

# USER'S MANUAL

LG PROGRAMMABLE LOGIC CONTROLLER

***GLOFA-GM***

**Instruction / Programming**

**LG Industrial Systems**

## Ch 1. Overview

1.1. Characteristics of IEC 1131-3 Language .....	1-1
1.2. Type of Language .....	1-1

## Ch 2. The Structure of Software

2.1. Overview.....	2-1
2.2. Project .....	2-1
2.3. Configuration .....	2-1
2.3.1. Resource .....	2-2
2.3.1.1. Program .....	2-2
2.3.1.2. Resource Global Variable .....	2-2
2.3.1.3. Task .....	2-3
2.3.2. Configuration Global Variable.....	2-4
2.3.3. Access Variable .....	2-4

## Ch 3. Common Elements

3.1. Expression.....	3-1
3.1.1. Identifiers .....	3-1
3.1.2. Data Expression .....	3-1
3.1.2.1. Numbers .....	3-2
3.1.2.2. Character String.....	3-2
3.1.2.3. Time Letters .....	3-2
3.1.2.3.1. Duration.....	3-2
3.1.2.3.2. Time of Day and Date .....	3-3
3.2. Data Type .....	3-4
3.2.1. Basic Data Type.....	3-4
3.2.2. Data Type Hierarchy Chart .....	3-5
3.2.3. Initial Value .....	3-5
3.2.4. Data Type Structure.....	3-6
3.3. Variable .....	3-8
3.3.1. Variable Expression.....	3-8
3.3.2. Variable Declaration.....	3-9
3.3.3. Reserved Variable.....	3-12

# Table of Contents

3.4. Reserved Word .....	3-17
3.5. Program Type.....	3-18
3.5.1. Function .....	3-18
3.5.2. Function Block .....	3-19
3.5.3. Program .....	3-20

## Ch 4. SFC (Sequential Function Chart)

4.1. Overview.....	4-1
4.2. SFC Structure.....	4-1
4.2.1. Step.....	4-1
4.2.2. Transition .....	4-2
4.2.3. Action .....	4-2
4.2.4. Action Qualifier .....	4-3
4.3. Extension Regulation.....	4-8
4.3.1. Serial Connection.....	4-8
4.3.2. Selection Branch.....	4-8
4.3.3. Parallel Branch (Simultaneous Branch).....	4-9
4.3.4. Jump .....	4-9

## Ch 5. IL (Instruction List)

5.1. Overview .....	5-1
5.2. Current Result: CR.....	5-1
5.3. Instructions.....	5-2
5.3.1. Label.....	5-2
5.3.2. Modifier.....	5-2
5.3.3. Basic Operation.....	5-3
5.3.3.1. Basic Operator.....	5-5
5.4. Calling of Function and Function Block.....	5-24

## Ch 6. LD (Ladder Diagram)

6.1. Overview.....	6-1
6.2. Bus Line.....	6-1
6.3. Connection Line.....	6-2
6.4. Contact .....	6-3
6.5. Coil .....	6-4
6.6. Calling of Function and Function Block .....	6-5

## Ch 7. Function and Function Block

7.1. Function.....	7-1
7.1.1. Type Conversion Function .....	7-1
7.1.2. Arithmetic Function .....	7-8
7.1.2.1. Numerical Operation Function with One Input .....	7-8
7.1.2.2. Basic Arithmetic Function .....	7-8
7.1.3. Bit Array Function .....	7-9
7.1.3.1. Bit-shift Function .....	7-9
7.1.3.2. Bit Operation Function .....	7-9
7.1.4. Selection Function .....	7-9
7.1.5. Data Exchange Function .....	7-9
7.1.6. Comparison Function.....	7-10
7.1.7. Character String Function.....	7-10
7.1.8. Time/Time of Day/Date and Time of Day Function .....	7-11
7.1.9. System Control Function .....	7-11
7.1.10. Data Manipulation Function .....	7-12
7.1.11. Stack Operation Function .....	7-12
7.2. MK (MASTER-K) Function .....	7-13
7.3. Array Operation Function .....	7-13
7.4. Basic Function Block .....	7-13
7.4.1. Bistable Function Block .....	7-13
7.4.2. Edge Detection Function Block.....	7-13
7.4.3. Counter .....	7-14
7.4.4. Timer.....	7-14
7.4.5. Other Function Block.....	7-14

# Table of Contents

## Ch 8. Basic Function/Function Block Library

8.1 Basic Function Library.....	8-1
ABS .....	8-2
ACOS .....	8-3
ADD .....	8-4
ADD_TIME .....	8-5
AND.....	8-6
ARY_TO_STRING .....	8-7
ASIN .....	8-8
ATAN.....	8-9
BCD_TO_*** .....	8-10
BOOL_TO_*** .....	8-11
BYTE_TO_*** .....	8-12
CONCAT .....	8-13
CONCAT_TIME .....	8-14
COS.....	8-15
DATE_TO_*** .....	8-16
DELETE .....	8-17
DI.....	8-18
DINT_TO_*** .....	8-20
DIREC_IN.....	8-22
DIREC_O .....	8-25
DIV .....	8-27
DIV_TIME.....	8-28
DT_TO_*** .....	8-29
DWORD_TO_*** .....	8-30
EI .....	8-32
EQ .....	8-33
ESTOP .....	8-34
EXP .....	8-35
EXPT .....	8-36
FIND .....	8-37
GE .....	8-38
GT .....	8-39
INSERT .....	8-40
INT_TO_*** .....	8-41
LE .....	8-43
LEFT.....	8-44
LEN .....	8-45
LIMIT .....	8-46
LINT_TO_*** .....	8-47

# Table of Contents

LN.....	8-49
LOG.....	8-50
LREAL_TO_***.....	8-51
LT.....	8-53
LWORD_TO_***.....	8-54
MAX.....	8-56
MID.....	8-57
MIN.....	8-58
MOD.....	8-59
MOVE.....	8-60
MUL.....	8-61
MUL_TIME.....	8-62
MUX.....	8-63
NE.....	8-64
NOT.....	8-65
NUM_TO_STRING.....	8-66
OR.....	8-67
REAL_TO_***.....	8-68
REPLACE.....	8-70
RIGHT.....	8-72
ROL.....	8-73
ROR.....	8-74
SEL.....	8-75
SHL.....	8-76
SHR.....	8-77
SIN.....	8-78
SINT_TO_***.....	8-79
SQRT.....	8-81
STOP.....	8-82
STRING_TO_***.....	8-83
STRING_TO_ARY.....	8-85
SUB.....	8-86
SUB_DATE.....	8-87
SUB_DT.....	8-88
SUB_TIME.....	8-89
SUB_TOD.....	8-90
TAN.....	8-91
TIME_TO_***.....	8-92
TOD_TO_***.....	8-93
TRUNC.....	8-94
UDINT_TO_***.....	8-95
UINT_TO_***.....	8-97

# Table of Contents

ULINT_TO_***	8-99
USINT_TO_***	8-101
WDT_RST	8-103
WORD_TO_***	8-105
XOR	8-106
8.2 Application Function Library	8-107
ARY_ASC_TO_BCD	8-108
ARY_ASC_TO_BYTE	8-110
ARY_AVE_***	8-112
ARY_BCD_TO_ASC	8-114
ARY_BYTE_TO_ASC	8-116
ARY_CMP_***	8-118
ARY_FLL_***	8-120
ARY_MOVE	8-122
ARY_ROT_C_***	8-124
ARY_SCH_***	8-126
ARY_SFT_C_***	8-128
ARY_SWAP_***	8-130
ASC_TO_BCD	8-132
ASC_TO_BYTE	8-133
BCD_TO_ASC	8-134
BIT_BYTE	8-135
BMOV_***	8-136
BSUM_***	8-138
BYTE_BIT	8-139
BYTE_TO_ASC	8-140
BYTE_WORD	8-141
DEC_***	8-142
DECO_***	8-143
DEG_***	8-144
DIS_***	8-145
DWORD_LWORD	8-147
DWORD_WORD	8-148
ENCO_B,W,D,L	8-149
GET_CHAR	8-150
INC_B,W,D,L	8-151
LWORD_DWORD	8-152
MCS	8-153
MCSCLR	8-155
MEQ	8-156
PUT_CHAR	8-158

# Table of Contents

RAD_***	8-159
ROTATE_A_***	8-160
ROTATE_C_***	8-163
RTC_SET	8-165
SEG	8-168
SHIFT_A_***	8-170
SHIFT_C_***	8-173
SWAP_***	8-175
UNI_***	8-176
WORD_BYTE	8-178
WORD_DWORD	8-179
XCHG_***	8-180
8.3 Basic Function Block Library	8-182
CTD	8-183
CTU	8-185
CTUD	8-187
F_TRIG	8-189
RS	8-190
R_TRIG	8-191
SEMA	8-192
SR	8-195
TOF	8-196
TON	8-198
TP	8-200
8.4 Application Function Block Library	8-202
CTR	8-203
DUTY	8-205
FIFO_***	8-207
LIFO_***	8-211
SCON	8-215
TMR	8-218
TMR_FLK	8-220
TMR_UINT	8-222
TOF_RST	8-224
TOF_UINT	8-226
TON_UINT	8-228
TP_RST	8-230
TP_UINT	8-232
TRTG	8-234
TRTG_UINT	8-236



## 1. Overview

This instruction describes languages that support GM1~GM7 (GLOFA PLC).

GLOFA PLC is based on the standard language of IEC (International Electrotechnical Commission).

### 1.1 Characteristics of IEC 1131-3 Language

The characteristics of IEC language newly introduced are as follows:

Available to support several data types.

The introduction of program elements such as functions, function blocks etc. enables the bottom-up design and top-down design and the structural creation of PLC program.

The program created by the user shall be stored like as a library system so that it can be used in other environment, which enables to reuse the software.

Available to support various languages so that the user can select the optimal language suitable for the environment to apply.

### 1.2 Type of Language

The PLC language standardized by IEC consists of two illustrated languages, two character languages and SFC.

Illustrated languages

**a) LD (Ladder Diagram):** A graphical language that is based on the relay ladder logic

**b) FBD (Function Block Diagram):** A graphical language for depicting signal and data flows through function blocks - re-usable software elements

Character language

**a) IL (Instruction List):** A low-level 'assembler like' language that is based on similar instruction list languages.

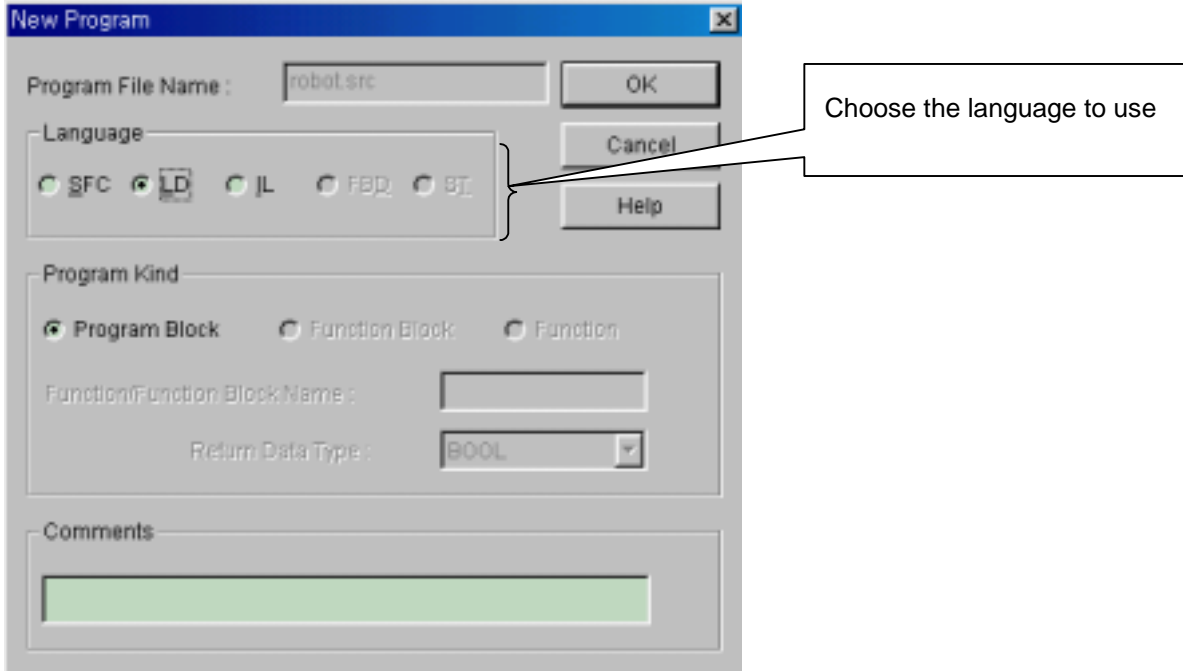
**b) ST (Structured Text):** A high-level language of PASCAL type

SFC (Sequential Function Chart): A graphical language for depicting sequential behavior of a control system. It is used for defining control sequences that are time- and event-driven.

## 1. Overview

---

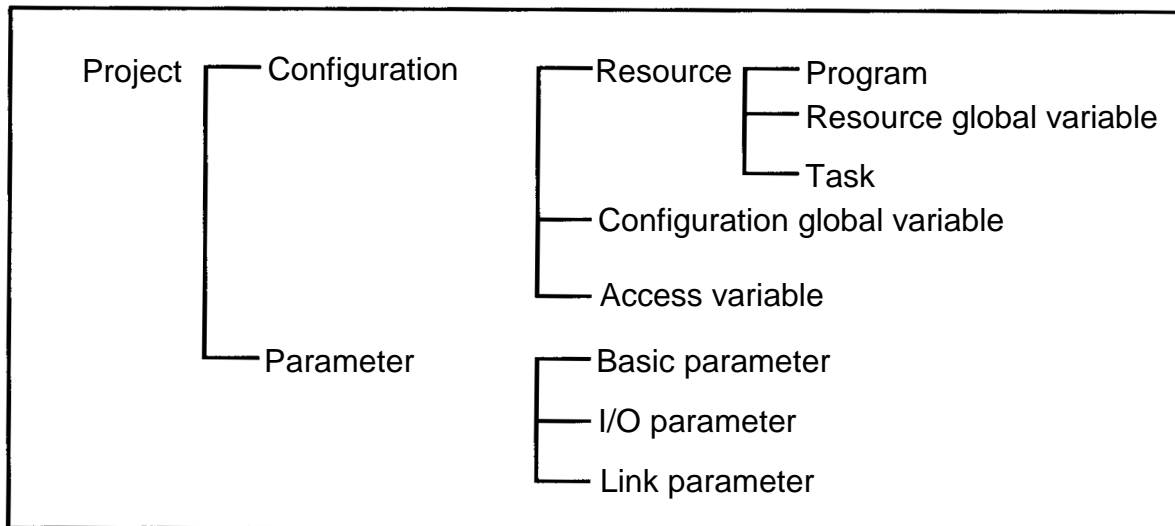
The languages supported by GLOFA PLC at present are IL, LD and SFC.



## 2. The Structure of Software

### 2.1 Overview

Before making a PLC program, you should have an overall PLC system mapped out in the aspect of software. The overall PLC system is defined as one project in GLOFA PLC. In the project, all composition elements necessary for the PLC system are defined hierarchically.



### 2.2 Project

For a GLOFA PLC program, the first priority should be given to project configuration. To make one project means that all the elements necessary for a PLC system (scan programs, task definitions, basic parameters, I/O parameters, etc.) are programmed.

A project is divided into two groups: configuration and parameter. Configuration part is for several definitions of a PLC program such as global variable, program, task definition and their interrelation. Parameter part is for setting parameters necessary for a PLC system operation. In this book, we deal with “Configuration part.” For parameter part, please refer to “GMWIN User’s Manual.”

### 2.3 Configuration

Configuration means a PLC system. It consists of a base, a CPU module, I/O modules and special modules and so on. Generally one PLC system has one CPU module; 4 CPU modules can be installed in GM1.

A PLC system has its own name called Configuration name. This becomes its unique name during communicating between PLCs. Configuration name is limited up to maximum 8 letters in alphabet and for more information, please refer to 3.1.1 Identifiers.

Configuration contains resource, configuration global variables and access variables.

## **2. The Structure of Software**

---

### **2.3.1 Resource**

Resource means one CPU module. And it is available to define 4 resources in the GM1 Configuration. For GM2 ~ GM5, only one resource is available to define. This resource has its own name that is also used for communication. The resource name is limited up to 8 letters in alphabet and it complies with 3.1.1 Identifiers.

Resource has programs, resource global variables and task definitions.

#### **2.3.1.1 Program**

It is an application program that is actually executed on PLC. In GLOFA PLC, it is available to create several application programs for one resource and set program conditions to run. For example, you can make programs as follows: program A is a general scan program; program B is a program executed once in a second; program C is an event program that is executed with certain inputs. These conditions to execute the program are called "Task." Users should make an application program as well as set the conditions (task definitions). Unless task definitions are set, this program will be regarded as a scan program.

##### **Reference**

Scan program: application program that repeats a series of execution from the start to the end after reading input data from input modules, and writing the results in output modules.

A program has its instance name. This instance contains data to be executed in this program.

##### **Reference**

For the instance, refer to 3.5.2. Function Block.

#### **2.3.1.2 Resource Global Variable**

The variables defined in resource global variable can be used in any program of the resource. All the data to be shared among programs are defined in resource global variables.

If users want to use resource global variables in their programs, variables are supposed to be declared as VAR\_EXTERNAL.

##### **Reference**

For a variable type, refer to 3.3.2 Variable Declaration.

### 2.3.1.3 Task

Task means a condition to execute a program. Task definitions contain designation of program execution condition and priority.

There are 3 types of program execution conditions as follows:

- 1) **Single**: executes once if the setting condition is satisfied. The condition is set as a name of **BOOL** variable.
- 2) **Interval**: executes periodically per a setting time. The condition is set as elapsed time value. Refer to '3.1.2.3.1 Duration' for how to set the elapsed time value.
- 3) **Interrupt**: executes once if the contact of an interrupt card is ON. The condition is set as the contact number of an interrupt card.

Execution conditions	Setting	Description
Single	%IX0.0.1	Executes once if input contact point %IX0.0.1 is ON.
Interval	T#1S	Executes per second
Interrupt	4	Executes once if the contact (#4) of an interrupt card is ON.

The priority is from 0 to 7. Priority 0 is the highest priority. When scheduling, the task with the highest priority is executed first. And if there are some tasks with the same priority, they're executed in execution-condition-occur order.

The task used by the reservation in system contains `_ERR_SYS`, `_H_INIT` and `_INIT` task.

`_ERR_SYS`: System Error (available in GM1, 2)

`_H_INIT`: Hot Restart

`_INIT`: Cold/Warm Restart

## **2. The Structure of Software**

---

### **2.3.2 Configuration Global Variable**

The variables defined in Configuration Global Variables can be used in any resource program. All the data to be shared among resources are defined in Configuration Global Variable.

If users want to use configuration global variables in their programs, variables are supposed to be declared as VAR\_EXTERNAL.

#### **Reference**

For a variable type, refer to 3.3.2 Variable Declaration.

Configuration global variable can be defined only in GM1 that can have several resources.

### **2.3.3 Access Variable**

The variable defined in Access Variable can be used in other PLC system.

#### **Reference**

For the use of access variable, refer to the User's Manual (Communication part).

### 3. Common Elements

The elements of **GLOFA PLC** program (programs, functions, function blocks) can be programmed in other languages such as **IL**, **LD**, **SFC**, etc., respectively. Those languages, however, have grammar elements in common.

#### 3.1. Expression

##### 3.1.1. Identifiers

Alphabet and all letters starting with underline (  ), and all the mixed letters with numbers and underlines can be identifiers.

Identifiers are used as variable names.

Blank (space) is not allowed in identifiers.

In case of variables, identifiers are generally 16 letters of the alphabet while input/output variable and instance, 8 letters of the alphabet.

There's no difference between small letters and capitals in alphabet; all the letters of the alphabet are recognized as capitals.

Types	Examples
Capital letters and numbers	IW210, IW215Z, QX75, IDENT
Capital letters, numbers and underline	LIM_SW_2, LIMSW5, ABCD, AB_CD
Capital letters and numbers starting with the underline ( <u>  </u> )	_MAIN, _12V7, _ABCD

##### 3.1.2. Data Expression

The data in **GLOFA PLC** is: numbers, a string of characters, time letters, etc.

Types	Examples
Integer	-12, 0, 123_456, +986
Real number	-12.0, 0.0, 0.456, 3.14159_26
Real number with an exponent	-1.34E-12, 1.0E+6, 1.234E6
Binary number	2#1111_1111, 2#1110_0000
Octal number	8#377 (decimal 255) 8#340 (decimal 224)
Hexadecimal number	16#FF (decimal 255) 16#E0 (decimal 224)
BOOL data	0, 1, TRUE, FALSE

## 3. Common Elements

---

### 3.1.2.1. Numbers

There are integer and real numbers.

Discontinuous underline ( \_ ) can be placed between numbers and it doesn't have any meaning.

Decimal complies with general decimal literal expression and if there is a decimal point, this will be real numbers.

In case of expressing exponent, plus/minus signs can be used. The letter 'E' standing for the exponent does not distinguish capitals from small letters.

When using real numbers with exponents, the followings are not allowed.

**Ex)** 12E-5 ( x )    12.0E-5 ( )

Integer includes binary, octal, hexadecimal numbers, not to mention decimal, which can be distinguished by placing # in front of each number.

0 ~ 9 and A ~ F are used (including small letters a ~ f) in expressing hexadecimal.

Not available to have plus/minus signs in expressing hexadecimal.

Boolean data may be expressed as an integer 0 or 1.

### 3.1.2.2. Character String

Character string covers all the letters surrounded with single inverted commas.

The length is limited up to 16 letters in case of character string constant and for an initialization case it does within 30 letters.

**Ex)**

'CONVEYER'

### 3.1.2.3. Time Letters

Time letters are classified into these: 1) Duration data which is calculating and controlling the elapsed time of a controlling event; 2) Time of Day and Date data which is displaying the time of the starting/ending point of a controlling event.

#### 3.1.2.3.1. Duration

Duration data starts with the reserved word, 'T#' or 't#'.

Several data types such as date (d), hour (h), minute (m), second (s) and millisecond (ms) should be written in order and duration date can start with any unit among them. Millisecond (ms), the minimum unit can be omitted but don't skip the medium unit between duration units.

Not allowed to use the underline ( \_ ).

Duration data can overflow at the maximum unit, if any, and the data with a decimal point is available except 'ms'. It does not exceed T#49d17h2m47s295ms (32bits by 'ms' unit).

The data is limited to the third decimal place in the second unit (s).

Decimal point is not available at 'ms' unit.



Capital and small letters are both available.

Content	Examples
Duration (no underline)	T#14ms, T#14.7s, T#14.7m, T#14.7h t#14.7d, t#25h15m, t#5d14h12m18s356ms

**3.1.2.3.2. Time of Day and Date**

There are three types expressing 'Time of Day and Date' as follows: Date; Time of Day; Date and Time.

Content	Prefix as a reserved word
Date prefix	D#
Time of Day prefix	TOD#
Date and Time prefix	DT#

The starting point of date is January 1st, 1984.

There's a limit on 'Time of Day' and 'Date and Time', which is up to the third decimal place in the 'ms' unit.

The overflow is not allowed for all the units when expressing 'Time of Day' and 'Date and Time'.

Content	Examples
Date	D#1984-06-25 d#1984-06-25
Time of Day	TOD#15:36:55.36 tod#15:36:55.369
Date and Time	DT#1984-06-25-15:36:55.36 dt#1984-06-25-15:36:55.369

### 3. Common Elements

---

#### 3.2. Data Type

Data has a data type to show its character.

##### 3.2.1. Basic Data Type

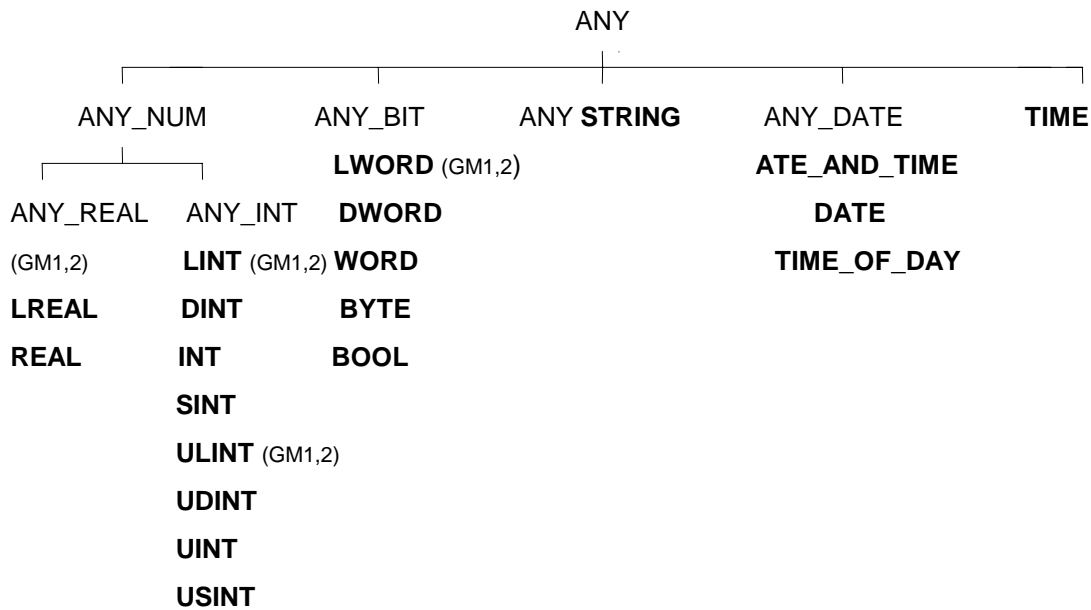
GLOFA PLC supports the following basic data types.

No	Reserved Word	Data Type	Size (bits)	Range
1	SINT	Short Integer	8	-128 ~ 127
2	INT	Integer	16	-32768 ~ 32767
3	DINT	Double Integer	32	-2147483648 ~ 2147483647
4	LINT	Long Integer	64	$-2^{63} \sim 2^{63}-1$
5	USINT	Unsigned Short Integer	8	0 ~ 255
6	UINT	Unsigned Integer	16	0 ~ 65535
7	UDINT	Unsigned Double Integer	32	0 ~ 4294967295
8	ULINT	Unsigned Long Integer	64	$0 \sim 2^{64}-1$
9	REAL	Real Numbers	32	-3.402823E38 ~ -1.401298E-45 1.401298E-45 ~ 3.402823E38
10	LREAL	Long Real Numbers	64	-1.7976931E308 ~ -4.9406564E-324 4.9406564E-324 ~ 1.7976931E308
11	TIME	Duration	32	T#0S ~ T#49D17H2M47S295MS
12	DATE	Date	16	D#1984-01-01 ~ D#2163-6-6
13	TIME_OF_DAY	Time of Day	32	TOD#00:00:00 ~ TOD#23:59:59.999
14	DATE_AND_TIME	Date and Time	64	DT#1984-01-01-00:00:00 ~ DT#2163-12-31-23:59:59.999
15	STRING	Character String	30*8	Limited within 30 letters.
16	BOOL	Boolean	1	0, 1
17	BYTE	Bit String of Length 8	8	16#0 ~ 16#FF
18	WORD	Bit String of Length 16	16	16#0 ~ 16#FFFF
19	DWORD	Bit String of Length 32	32	16#0 ~ 16#FFFFFFFF
20	LWORD	Bit String of Length 64	64	16#0 ~ 16#FFFFFFFFFFFFFFFF

LINT, ULINT, REAL, LREAL, LWORD are available in GM1 and GM2 only.

### 3.2.2. Data Type Hierarchy Chart

Data types used in **GLOFA PLC** are as follows:



LINT, ULINT, LWORD and ANY\_REAL (LREAL, REAL) are available in **GM1** and **GM2** only.

Data expressed as ANY\_NUM includes LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT hereafter.

For example, if a data type is expressed as ANY\_BIT in GM3, it can use one of the following data types: DWORD, WORD, BYTE and BOOL.

### 3.2.3. Initial Value

If an initial value of data were not assigned, it would be automatically assigned as below.

Data Type	Initial Value
SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BOOL, BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T#0s
DATE	D#1984-01-01
TIME_OF_DAY	TOD#00:00:00
DATE_AND_TIME	DT#1984-01-01-00:00:00
STRING	' ' (empty string)


### 3. Common Elements

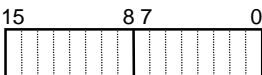
---

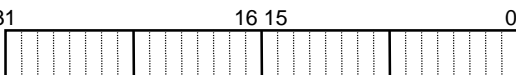
#### 3.2.4. Data Type Structure

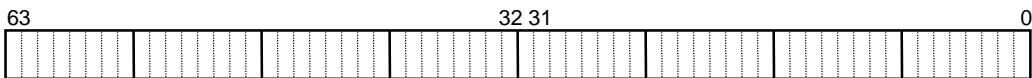
##### # Bit String

**BOOL**  1 bit, range: 0, 1

**BYTE**  8 bits, range: 2#0000\_0000 ~ 2#1111\_1111, 16#00 ~ 16#FF

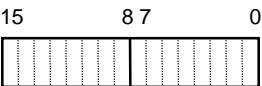
**WORD**  16 bits, range: 2#0000\_0000\_0000\_0000 ~ 2#1111\_1111\_1111\_1111  
16#0000 ~ 16#FFFF

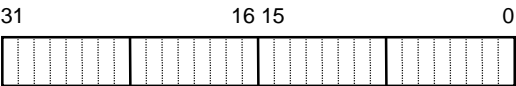
**DWORD**  32 bits, range: 2#0000\_...000 ~ 2#1111\_...111  
16#00000000 ~ 16#FFFFFFFF

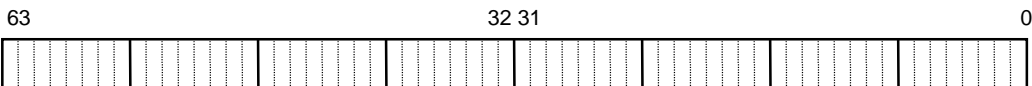
**LWORD**  64 bits, range: 2#0000\_...000 ~ 2#1111\_...111, 16#0000000000000000 ~ 16#FFFFFFFFFFFFFFFF

##### # Unsigned Integer

**USINT**  8 bits, range: 0 ~ 255

**UINT**  16 bits, range: 0 ~ 65,535

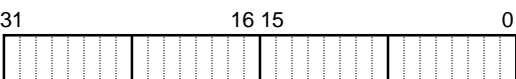
**UDINT**  32 bits, range: 0 ~ 4,294,967,295

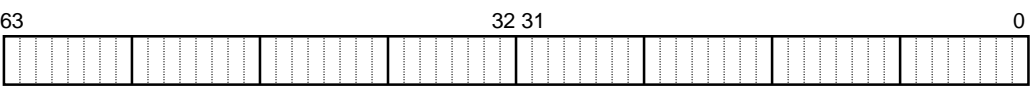
**ULINT**  64 bits, range: 0 ~ 2<sup>64</sup>-1

##### # Integer (Negative number is expressed as 2's Complement.)

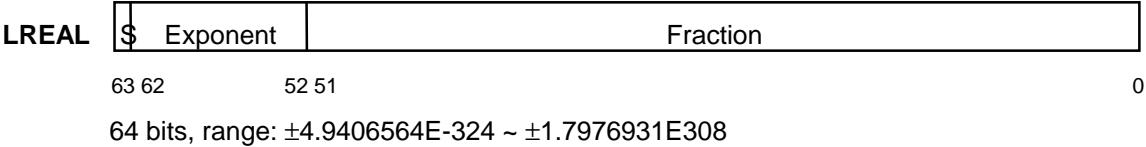
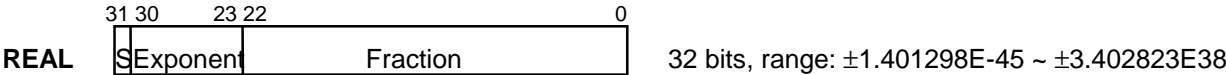
**SINT**  8 bits, range: -128 ~ 127

**INT**  16 bits, range: -32,768 ~ 32,767

**DINT**  32 bits, range: -2,147,483,648 ~ 2,147,483,647

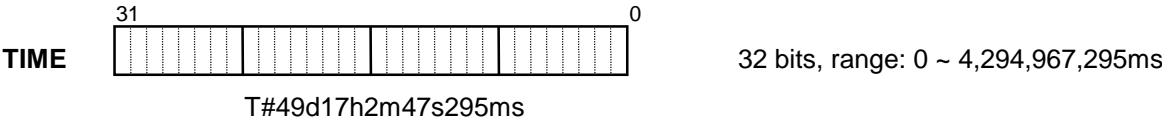
**LINT**  64 bits, range: -2<sup>63</sup> ~ 2<sup>63</sup>-1

# Real (based on the IEEE Standard 754-1984)

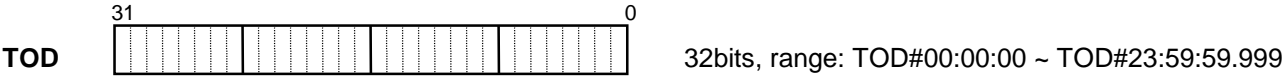
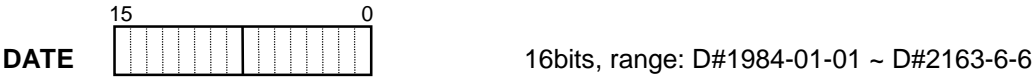
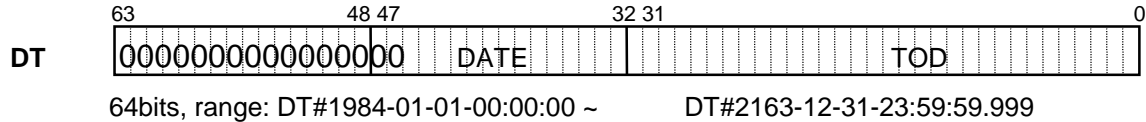


- **S:** sign (If it's 0, the data is a positive number; otherwise, a negative number).
- **Exponent:** exponent of 2 ( $2^{e-127}$ : for REAL,  $e=b_{30}b_{29}...b_{23}$ ; for LREAL,  $e=b_{62}b_{61}...b_{52}$ ).
- **Fraction:** a decimal fraction (Fraction: for REAL,  $f=b_{22}b_{21}...b_0$ ; for LREAL,  $e=b_{51}b_{52}...b_0$ ).

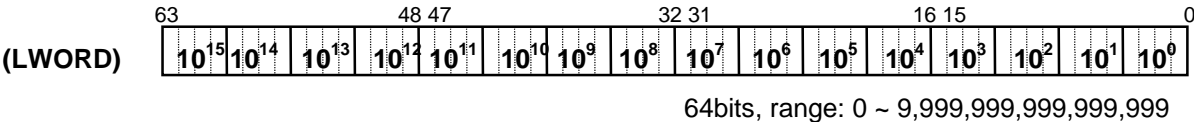
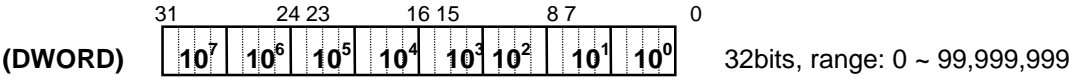
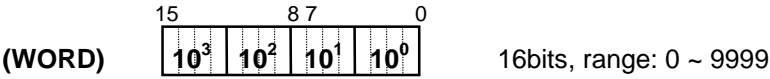
# Time



# Date



#BCD



### 3. Common Elements

---

#### 3.3. Variable

A variable, data used in the program, has its own value. 'Variable' means something that can vary such as an input/output of **PLC**, memory, etc.

##### 3.3.1. Variable Expression

Variables can be expressed in two ways: one is to give a name to a data element using an identifier (Variable by Identifier) and the other is to directly assign a memory address or an input/output of PLC to a data element (Direct Variable).

A variable by identifier should be unique within its 'effective scope' (program area where the variable was declared) in order to distinguish it from other variables.

A direct variable is expressed as one, which starts with the percent sign (%) followed by the 'location prefix', a prefix of the data size, and more than one unsigned integer numbers divided by a period (.). The prefix are shown as below:

Location prefix

No.	Prefix	Meaning
1		Input Location
2	Q	Output Location
3	M	Memory Location

Size prefix

No.	Prefix	Meaning
1	X	1 bit size
2	None	1 bit size
3	B	1 BYTE (8 bits) size
4	W	1 WORD (16 bits) size
5	D	1 DOUBLE WORD (32 bits) size
6	L	1 LONG WORD (64 bits) size

Expression format

**%[Location Prefix][Size Prefix] n1.n2.n3**

No.	, Q	M
n1	Base number (starting from "0")	n1 data according to [size prefix] (starting from "0")
n2	Slot number (starting from "0")	n2 bit of n1 data (starting from "0"): available to omit
n3	n3 data according to the [size prefix] (starting from "0")	Not used.

**Examples**

- %QX3.1.4** or **%Q3.1.4**      4<sup>th</sup> output of no.1 slot on no.3 base (1bit)
  - % W2.4.1**                      1<sup>st</sup> word input of no.4 slot on no.2 base (16bits)
  - %MD48**                          48<sup>th</sup> double word memory
  - %MW40.3**                      3<sup>rd</sup> bit of 40<sup>th</sup> word memory
- (Internal memory doesn't have a base or slot number.)

Small letter is not allowed as a prefix.

A variable without a size prefix is treated as 1 bit.

Direct variables are available to use without a variable declaration.

**3.3.2. Variable Declaration**

Program elements (programs, functions, function blocks, etc) have declaration parts to edit their variables to use.

Users should declare variables first to use them in the program elements.

The contents of a variable declaration are as follows:

1) Variable types: how to declare variables?

Variable types	Description
<b>VAR</b>	General variable available to read/write
<b>VAR_RETAIN</b>	Retaining (data-keeping) variable
<b>VAR_CONSTANT</b>	Read Only Variable
<b>VAR_EXTERNAL</b>	Declaration to use the variable declared as <b>VAR_GLOBAL</b>

**Reference**

When declaring **Resource Global Variable** and **Configuration Global Variable**, variable formats are **VAR\_GLOBAL**, **VAR\_GLOBAL\_RETAIN**, and **VAR\_GLOBAL\_CONSTANT**; **VAR\_EXTERNAL** is not available for them.

2) Data type: sets a variable data type.

3) Memory allocation: assigns memory for a variable.

Auto: the compiler sets a variable location automatically (Automatic Allocation Variable).

Assign (AT): a user sets a variable location, using a direct variable (Direct Variable).

### 3. Common Elements

---

#### Reference

The location of Automatic Allocation Variable is not fixed. If variable **VAL1**, for example, was declared as **BOOL**, it is not fixed in the internal memory; the compiler and linker fix its location. If the program is compiled again after modification, the location may change.

The merit of Automatic Allocation Variable is that users don't have to care the location of the internal variables because its location is not overlapped as long as a variable name is different from others.

It is recommended not to use Direct Variable except % and %Q because the location of a variable is fixed and it could be overlapped in a wrong-used case.

Initial Value Assignment: assigns an initial value. A variable is set with an initial value as is shown in '3.2.3. Initial Value' if not assigned.

#### Reference

The initial value is not assigned when it comes to **VAR\_EXTERNAL**.

In case of 'Variable Declaration', you cannot assign an initial value to % or %Q variables.

You can declare variable **VAR\_RETAIN** that keeps its data in case of power failure. Rules are:

- 1) 'Retention Variable' retains its data when the system is set as 'Warm Restart'.
- 2) In case of 'Cold Restart', variables are initialized as the initial values set by users or the basic initial values as are shown in '3.2.3 Initial Value'.

Variables, which are not declared as **VAR\_RETAIN**, are to be initialized as the initial values set by a user or the basic initial values in case of **Warm** or **Cold Restart**'.

#### Reference

Variables, which are assigned as %I or %Q, are not to be declared as **VAR\_RETAIN** or **VAR\_CONSTANT**.

Users can declare variables 'Array' with Elementary Data Type. When declaring the Array Variable, users are supposed to set Data Type and Array Size; 'String' among Elementary Data Type is not allowed.

Effective scope of variable declaration, the area which is available to use the variable, is limited to the program where variables are declared. And users can't use variables declared in other program in the above area. On the contrary, users can get an access to 'Global Variable' from other program elements by declaring it as '**VAR\_EXTERNAL**': 'Configuration Global Variable' can be used in all program elements of all resources; 'Resource Global Variable' can be used in all program elements of the very resource.



Examples of Variable Declaration

Variable Name	Variable Kind	Data Type	Initial Value	Memory Allocation
<b>I_VAL</b>	<b>VAR</b>	INT	<b>1234</b>	Auto
<b>BIPOLAR</b>	<b>VAR_RETAIN</b>	REAL		Auto
<b>LIMIT_SW</b>	<b>VAR</b>	BOOL		%IX1.0.2
<b>GLO_SW</b>	<b>VAR_EXTERNAL</b>	DWORD		Auto
<b>READ_BUF</b>	<b>VAR</b>	ARRAY OF INT[10]		Auto

### 3. Common Elements

---

#### 3.3.3. Reserved Variable

'Reserved Variable' is the variables previously declared in the system. These variables are used for special purposes and users cannot declare other variables with the Reserved Variable names.

Users can use these reserved variables without variable declaration.

For further information, please refer to 'User's Manual'.

##### 1) User Flag

Reserved Variable	Data Type	Description
_ERR	BOOL	Operation error contact
_LER	BOOL	Operation error latch contact
_T20MS	BOOL	20ms clock contact
_T100MS	BOOL	100ms clock contact
_T200MS	BOOL	200ms clock contact
_T1S	BOOL	1 sec. clock contact
_T2S	BOOL	2 sec. clock contact
_T10S	BOOL	10 sec. clock contact
_T20S	BOOL	20 sec. clock contact
_T60S	BOOL	60 sec. clock contact
_ON	BOOL	All time ON contact
_OFF	BOOL	All time OFF contact
_1ON	BOOL	1 scan ON contact
_1OFF	BOOL	1 scan OFF contact
_STOG	BOOL	Reversal at every scanning
_INIT_DONE	BOOL	Initial program completion
_RTC_DATE	DATE	Current date of RTC
_RTC_TOD	TOD	Current time of RTC
_RTC_WEEK	UINT	Current day of RTC

2) System Error Flag

Reserved Variable	Data Type	Description
_CNF_ER	WORD	System error (Heavy trouble)
_CPU_ER	BOOL	CPU configuration error
_IO_TYER	BOOL	Module type inconsistency error
_IO_DEER	BOOL	Module installation error
_FUUSE_ER	BOOL	Fuse shortage error
_IO_RWER	BOOL	I/O module read/write error (trouble)
_SP_IFER	BOOL	Special/communication module interface error (trouble)
_ANNUN_ER	BOOL	Heavy trouble detection error of external device
_WD_ER	BOOL	Scan Watch-Dog error
_CODE_ER	BOOL	Program code error
_STACK_ER	BOOL	Stack Overflow error
_P_BCK_ER	BOOL	Program error

3) System Error Release Flag

Reserved Variable	Data Type	Description
_CNF_ER_M	BYTE	System error (heavy trouble) release

4) System Alarm Flag

Reserved variable	Data type	Description
_CNF_WAR	WORD	System Alarm (Alarm message)
_RTC_ERR	BOOL	RTC data error
_D_BCK_ER	BOOL	Data backup error
_H_BCK_ER	BOOL	Hot restart unable error
_AB_SD_ER	BOOL	Abnormal Shutdown
_TASK_ERR	BOOL	Task conflict (normal cycle, external task)
_BAT_ERR	BOOL	Battery error
_ANNUN_WR	BOOL	Light trouble detection of external device
_HSPMT1_ER	BOOL	Over high-speed link parameter 1
_HSPMT2_ER	BOOL	Over high-speed link parameter 2
_HSPMT3_ER	BOOL	Over high-speed link parameter 3
_HSPMT4_ER	BOOL	Over high-speed link parameter 4

### 3. Common Elements

---

#### 5) Detailed System Error Flag

Reserved variable	Data type	Description
_IO_TYER_N	UINT	Module type inconsistency slot number
_IO_TYERR	ARRAY OF BYTE	Module type inconsistency location
_IO_DEER_N	UINT	Module installation slot number
_IO_DEERR	ARRAY OF BYTE	Module installation location
_FUSE_ER_N	UINT	Fuse shortage slot number
_FUSE_ERR	ARRAY OF BYTE	Fuse shortage slot location
_IO_RWER_N	UINT	I/O module read/write error slot number
_IO_RWERR	ARRAY OF BYTE	I/O module read/write error slot location
_ANC_ERR	ARRAY OF UINT	Heavy trouble detection of external device
_ANC_WAR	ARRAY OF UINT	Light trouble detection of external device
_ANC_WB	ARRAY OF BOOL	Alarm message detection bit map of external device
_TC_BMAP	ARRAY OF BOOL	Task conflict mark
_TC_CNT	ARRAY OF UINT	Task conflict counter
_BAT_ER_TM	DT	Battery voltage drop-down time
_AC_F_CNT	UINT	Shutdown counter
_AC_F_TM	ARRAY OF DT	Instantaneous service interruption history

#### 6) Information of System Operation Status

Reserved variable	Data type	Description
CPU_TYPE	UINT	System Type
_VER_NUM	UINT	PLC O/S Version number
_MEM_TYPE	UINT	Memory module type
_SYS_STATE	WORD	PLC mode and status
_RST_TY	BYTE	Restart mode information
_INIT_RUN	BIT	Initializing
_SCAN_MAX	UINT	Max. scan time (ms)
_SCAN_MIN	UINT	Min. scan time (ms)
_SCAN_CUR	UINT	Current scan time (ms)
_STSK_NUM	UINT	Task number requiring execution time check
_STSK_MAX	UINT	Max. task execution time (ms)
_STSK_MIN	UINT	Min. task execution time (ms)
_STSK_CUR	UINT	Current task execution time (ms)
_RTC_TIME	ARRAY OF BYTE	Current time
_SYS_ERR	UINT	Error type

#### 7) Communication Module Information Flag [n is a slot number where a communication module is installed (n = 0 ~ 7)]

Reserved variable	Data type	Description
_CnVERNO	UINT	Communication module version number
_CnTXECNT	UINT	Communication transmit error
_CnRXECNT	UINT	Communication receive error
_CnSVCFCNT	UINT	Communication service process error
_CnSCANMX	UINT	Max. communication scan time (1ms unit)
_CnSCANAV	UINT	Average communication scan time (1ms unit)
_CnSCANMN	UINT	Minimum communication scan time (1ms unit)
_CnLINF	UINT	Communication module system information
_CnCRDER	BOOL	Communication module system error (Error = 1)
_CnSVBSY	BOOL	Lack of common RAM resource (Lack = 1)
_CnIFERR	BOOL	Interface error (error = 1)
_CnINRING	BOOL	Communication in ring (IN_RING = 1)

### 3. Common Elements

---

8) Remote I/O Control Flag [**m** is a slot number where a communication module is installed (**m = 0 ~ 7**)]

Reserved variable	Data type	Description
<b>_FSMm_RESET</b>	<b>BOOL</b> (able to write)	Remote I/O station reset control (reset = 1)
<b>_FSMm_IO_RESET</b>	<b>BOOL</b> (able to write)	Output reset control of remote I/O station (reset = 1)
<b>_FSMm_st_no</b>	<b>USINT</b> (able to write)	Station number of corresponding remote I/O station

9) Detailed High-speed Link Information Flag [**m** is a high-speed link parameter number (**m = 1, 2, 3, 4**)]

Reserved variable	Data type	Description
<b>_HSmRLINK</b>	<b>BOOL</b>	HS RUN_LINK information
<b>_HSmLTRBL</b>	<b>BOOL</b>	Abnormal information of HS (Link Trouble)
<b>_HSmSTATE</b>	<b>ARRAY OF BOOL</b>	General communication status information of k data block
<b>_HSmMOD</b>	<b>ARRAY OF BOOL</b>	Station mode information of k data block at HS link parameter (Run = 1, Other = 0)
<b>_HSmTRX</b>	<b>ARRAY OF BOOL</b>	Communication status information of k data block at HS link parameter (Normal = 1, Abnormal = 0)
<b>_HSmERR</b>	<b>ARRAY OF BOOL</b>	Station status information of k data block at HS link parameter (Normal = 0, Error = 1)

### 3.4. Reserved Word

Reserved words are previously defined words to use in the system. And these reserved words cannot be used as an identifier.

Reserved words
<b>ACTION ... END_ACTION</b>
<b>ARRAY ... OF</b>
<b>AT</b>
<b>CASE ... OF ... ELSE ... END_CASE</b>
<b>CONFIGURATION ... END_CONFIGURATION</b>
Name of data type
<b>DATE#, D#</b>
<b>DATE_AND_TIME#, DT#</b>
<b>EXIT</b>
<b>FOR ... TO ... BY ... DO ... END_FOR</b>
<b>FUNCTION ... END_FUNCTION</b>
<b>FUNCTION_BLOCK ... END_FUNCTION_BLOCK</b>
Name of function block
<b>IF ... THEN ... ELSIF ... ELSE ... END_IF</b>
<b>OK</b>
Operator ( <b>IL</b> language)
Operator ( <b>ST</b> language)
<b>PROGRAM</b>
<b>PROGRAM ... END_PROGRAM</b>
<b>REPEAT ... UNTIL ... END_REPEAT</b>
<b>RESOURCE ... END_RESOURCE</b>
<b>RETAIN</b>
<b>RETURN</b>
<b>STEP ... END_STEP</b>
<b>STRUCTURE ... END_STRUCTURE</b>
<b>T#</b>
<b>TASK ... WITH</b>
<b>TIME_OF_DAY#, TOD#</b>
<b>TRANSITION ... FROM... TO ... END_TRANSITION</b>
<b>TYPE ... END_TYPE</b>
<b>VAR ... END_VAR</b>
<b>VAR_INPUT ... END_VAR</b>
<b>VAR_OUTPUT ... END_VAR</b>
<b>VAR_IN_OUT ... END_VAR</b>
<b>VAR_EXTERNAL ... END_VAR</b>
<b>VAR_ACCESS ... END_VAR</b>
<b>VAR_GLOBAL ... END_VAR</b>
<b>WHILE ... DO ... END_WHILE</b>
<b>WITH</b>

## 3. Common Elements

---

### 3.5. Program Type

There are three types of program: function, function block and program.

It is not available to call its own program in the program (reflexive call is prohibited).

#### 3.5.1. Function

A function has one output.

##### **Example**

If there is function A that is to add input IN1 and IN2 and then add 100 to the sum of IN1 and IN2. and the output  $1 \leq IN1 + IN2 + 100$ , this function will be correct. However, if the above function has one more output (output  $2 \leq IN1 + IN2 * 100$ ), this will not be a function because it has 2 outputs: output 1 and output 2.

A function does not have data to preserve its state inside. This means if an input is constant, an output value should be constant, which is a function.

##### **Example**

If there is function B whose contents are

Output  $1 \leq IN1 + IN2 + Val$

$Val \leq output1$  (where, Val is an internal variable),

This cannot be a function as there is internal variable Val. To have an internal variable means that an output will be different even if there is a same input. Output 1 value is subject to change because of Val variable even if the value of IN1 and IN2 are constant as is shown on the above. Compared with the above function A, function A will have output 1 value (150) when IN1 is 20 and IN2 is 30. This shows that the output value will be constant if inputs are constant.

An internal variable of a function is not available to have an initial value.

Users can't declare a function as VAR\_EXTERNAL and use it.

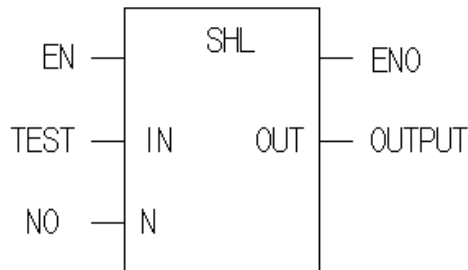
It is not available to use direct variables inside the function.

A function will be called by program elements and used.

Data transfer from program composition elements, which call the function, to the function will be executed through an input of a function.



#### Example



SHL function is a basic function that shifts input IN to the left as many as N bit number and produces it as an output. Program composition elements call SHL function, assigning a value of TEST variable to input IN and a value of NO variable to input N. The result will be stored in OUTPUT variable.

A function is inserted into a library for use.

It is not available to call a function block or a program inside the function.

A function has a variable whose name is the same as that of the function and whose data type is the same as the data type of the result of the function. This variable is automatically created when making a function, and the result value of the function will be written in the output.

#### Example

If a function name is WEIGH and a data type of a result value is WORD, a variable whose name is WEIGH and whose data type is WORD will be automatically created inside the function. Users can store the result of function in variable WEIGH.

ST WEIGH (example in IL)

### 3.5.2 Function Block

A function block has several outputs.

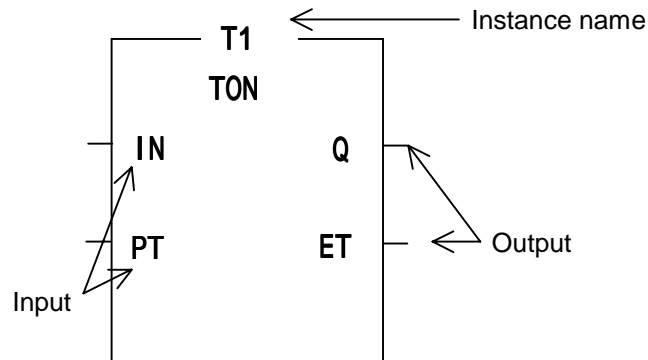
A function block has data inside. A function block should declare the instance as it declares variables before using them. Instance is a set of variables used in a function block. A function block should have its data memory to preserve the output value as well as variables used inside, which is called as "instance." A program is a kind of a function block and also needs to declare "instance." However, users cannot call a program inside a program or a function block for use, contrary to a function block.

In order to use the output value of a function block, it is required to place a period (.) between the name of instance and the output name.

### 3. Common Elements

---

#### Example



General examples of a function block are Timer and Counter. On-delay timer function block is TON and this is executed if IN is ON after users declare T1 as “instance.” In order to use timer output contact and duration value, it is required to place a period (.) between the name of instance and the output name. In case of a timer function block, the output contact and the elapsed time value for the instance are T1.Q and T1.ET respectively because the output contact name is Q and the elapsed time contact name is ET. The output value of a function is a return value by calling a function while the output value of a function block is fixed for the instance.

Users cannot declare a direct variable inside a function block. However, users can use a direct variable declared as Global Variable and allocated according to ‘Assign (AT)’ after declaring it as VAR\_EXTERNAL.

A function block is inserted into a library for use.

It is not available to call a program inside the function block.

#### 3.5.3 Program

Users can use a program after declaring an instance like a function block.

It is available to use direct variables in the program.

A program does not have input/output variables.

The calling of a program is defined in the resource.

## 4. SFC (Sequential Function Chart)

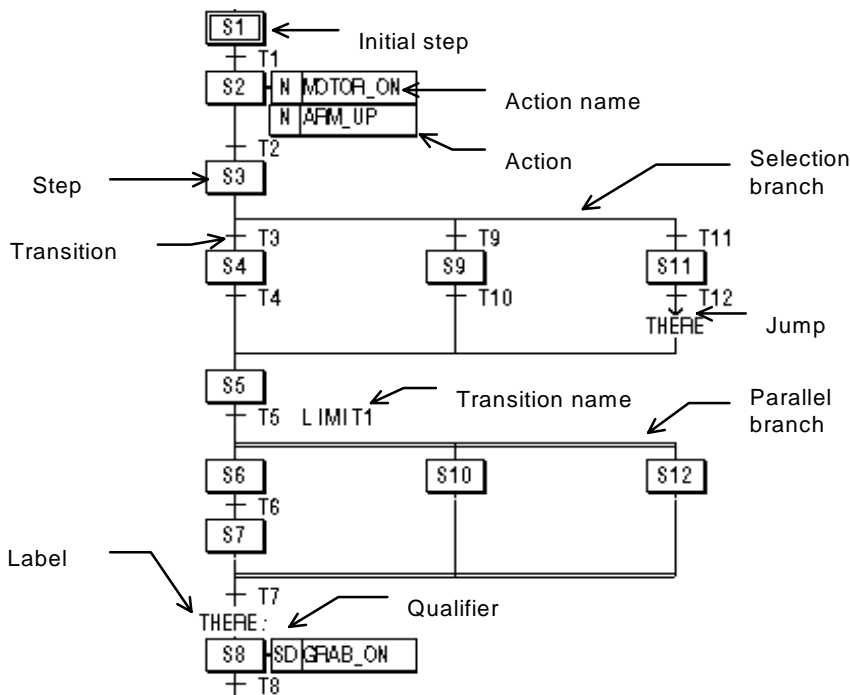
### 4.1. Overview

SFC is a structured language that extends an application program in the form of flow chart according to the processing sequence, using a PLC language.

SFC splits an application program into step and transition, and provides how to connect them each other. Each step is related to action and each transition is related to transition condition.

As SFC should contain the state information, only program and function block among program types are available to apply this SFC.

Type



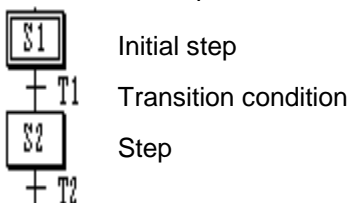
### 4.2. SFC Structure

#### 4.2.1. Step

Step indicates a sequence control unit by connecting the action.

When step is in an active state, the attached content of action will be executed.

The initial step is one to be activated first.



If a next transition condition of activated initial step (S1) is established, step 1 (S1) that is currently activated becomes deactivated and Step 2 (S2) connected to S1 becomes activated.

## 4. SFC

---

### 4.2.2. Transition

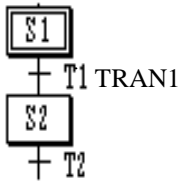
Transition indicates the execution condition between steps.

A transition condition should be described as a PLC language such as IL or LD.

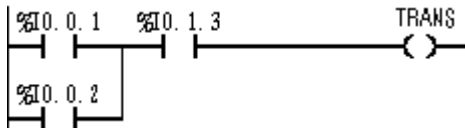
The result of a transition condition should always be a BOOL type and the variable name should be TRANS for any transition.

In case that the result of transition condition is 1, the current step is deactivated and the next step is activated.

There must be a transition between step and step.



The content of TRAN1



When TRANS is on, S1 will be deactivated and S2 activated.

TRANS is the internally declared variable.

A transition condition of all transition should be output in TRANS variable.

### 4.2.3. Action

Each step is able to connect up to two actions.

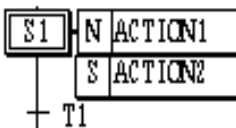
The step without action is regarded as a waiting action and it is required to wait until the next transition condition will be 1.

Action is composed of PLC language such as IL or LD and the content of action will be executed while the step is activated.

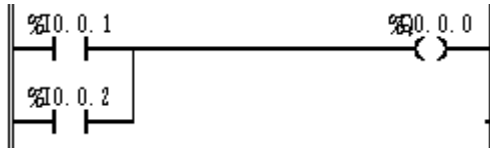
Action qualifier will be used to control action.

When action becomes deactivated state after activating, the contact output in action will be 0.

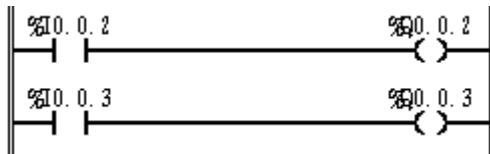
However, S, R, function and function block output retain their state before they become non-activating.



The content of **ACTION1**



The content of **ACTION2**



- **ACTION1** will be executed only when **S1** is activated.
- **ACTION2** will be executed until **S1** meets **R** qualifier after activated.  
It goes on executing even if **S1** is deactivated.
- When action is deactivated, this action is Post Scanned and then passes to the next step.

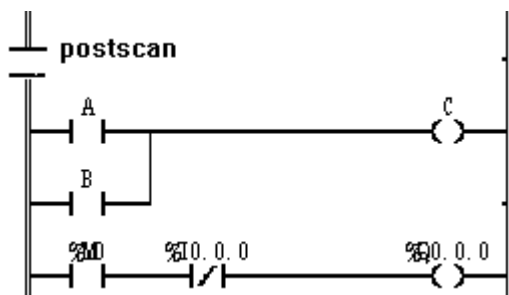
### Reference

#### **Post Scan**

When action is deactivated, this action is scanned again.

As it is scanned as if there were a contact (contact with the value of 0) in the beginning part of an action program, the program output, which is composed of contacts, will be 0.

Function, function block, **S**, **R** output etc., are not included.



In this figure, as the contact of **postscan** is 0, C and %Q0.0.0 will be 0.

#### **4.2.4. Action Qualifier**

Whenever action is used, action qualifier will be followed.

The action of step defines an executing point and time according to the assigned qualifier.

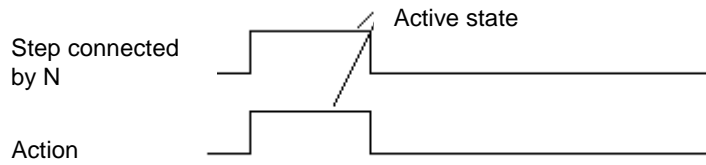
Types of action qualifier are as follows:

## 4. SFC

---

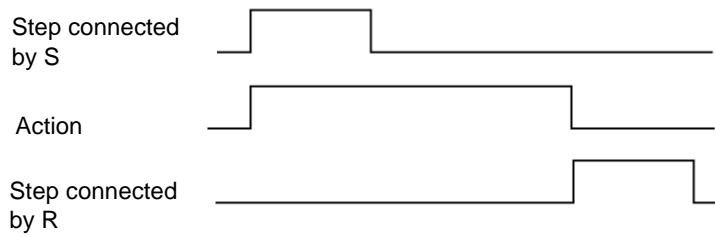
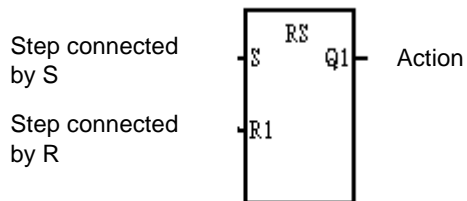
### 1) N (Non-Stored)

Action is executed only when the step is activated.



### 2) S (Set)

It continues the action after the step is deactivated (until the action is reset by R qualifier).

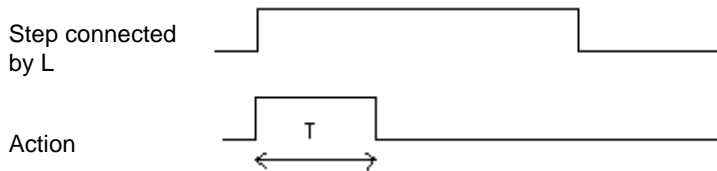
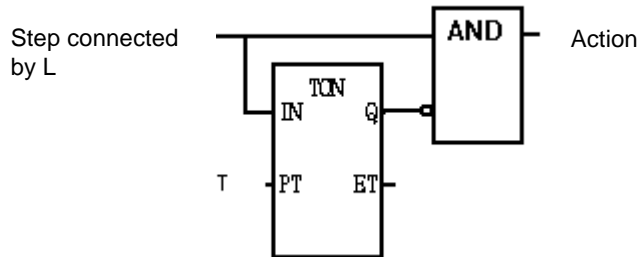


### 3) R (Overriding Reset)

It terminates the execution of an action previously started with the S, SD, SL or DS qualifier.

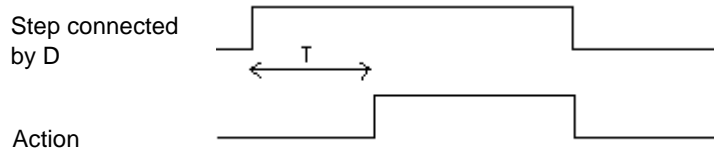
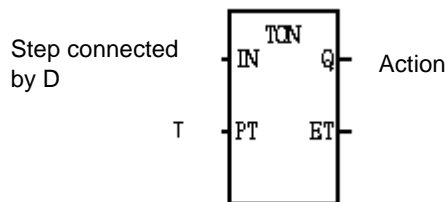
4) L (Time Limited)

It start the action when the step becomes active and continue until the step goes inactive or a set time elapses.



5) D (Time Delayed)

Start a delay timer when the step becomes active - after the time delay the action starts (if step still active) and continues until deactivated.

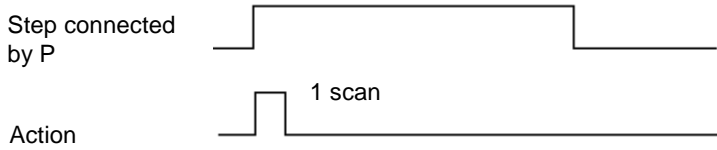
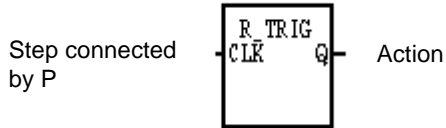


## 4. SFC

---

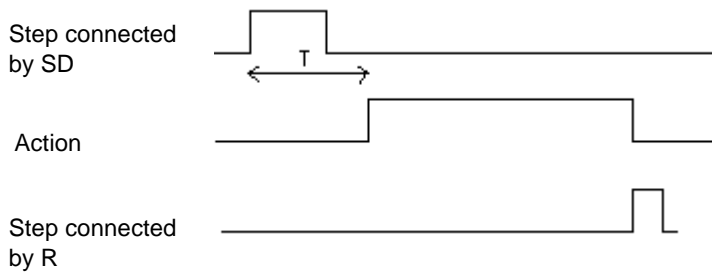
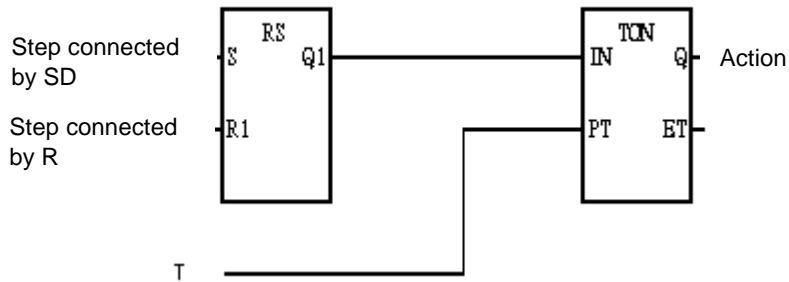
### 6) P (Pulse)

It starts the action when the step becomes active and executes the action only once.



### 7) SD (Stored & Time Delayed)

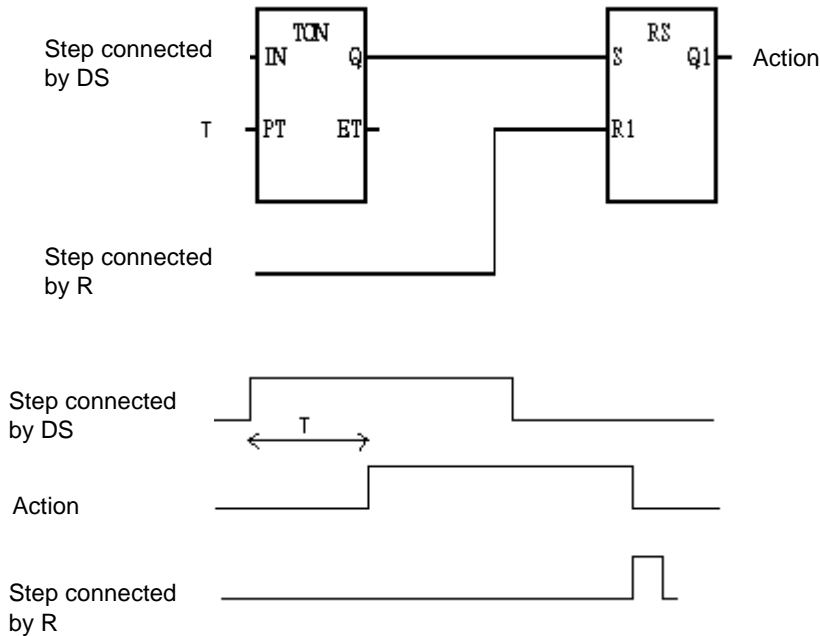
It starts a delay timer when the step becomes active - after the time delay, the action starts and continues until reset (regardless of step activation/deactivation).





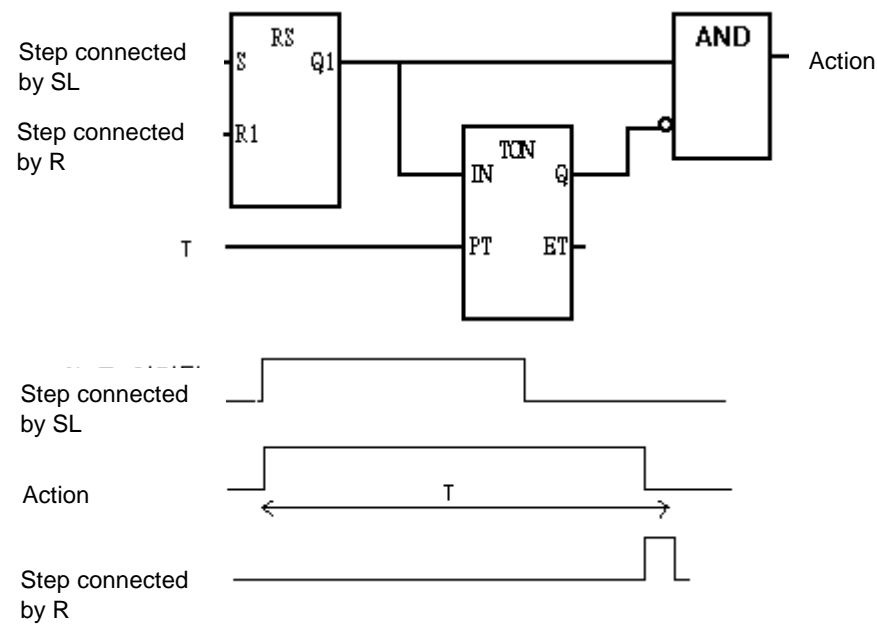
8) DS (Delayed & Stored)

It starts a delay timer when the step becomes active - after the time delay the action starts (if step still active) and continues until reset by R qualifier.



9) SL (Stored & Timed Limited)

It starts the action when the step becomes active and continues for a set time or until the action is reset (regardless of step activation/deactivation).



## 4. SFC

---

### 4.3. Extension Regulation

#### 4.3.1. Serial Connection

2 steps are always divided by transitions without connecting directly.

Step always divides 2 transitions without connecting directly.



[correct example]

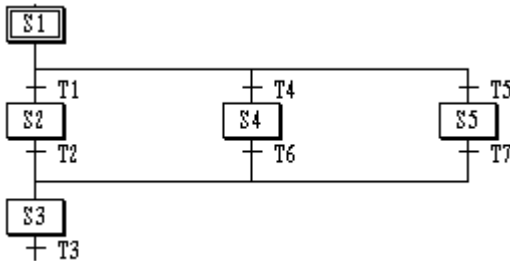
[wrong example]

For the transition between steps connected by serial, the lower step will be activated if the upper step is active and the transition condition connected to the next is 1.

#### 4.3.2. Selection Branch

When a processor executes a selection branch, the processor finds the first path with a true transition in the order of the program scan and executes the steps and transitions in that path. If more than one path in a selection branch goes true at the same time, the processor chooses the left-most path. The following example shows a typical scan sequence.

##### Example



\* In case that the transition condition of **T1** is 1,  
the order of activation will be **S1 -> S2 -> S3**.

\* In case that the transition condition of **T4** is 1,  
the order of activation will be **S1 -> S4 -> S3**.

\* In case that the transition condition of **T5** is 1,  
the order of activation will be **S1 -> S5 -> S3**.

If the transition conditions are 1 at the same time, the processor chooses the left-most path.

\* In case that the transition condition of **T1** and **T4** is 1 at the same time,  
the order of activation will be **S1 -> S2 -> S3**.

\* In case that the transition condition of **T4** and **T5** is 1 at the same time,  
the order of activation will be **S1 -> S4 -> S3**.

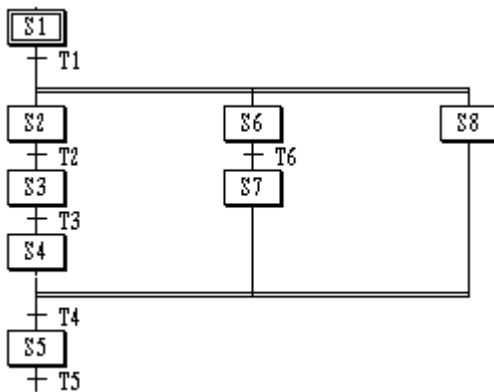
### 4.3.3. Parallel Branch (Simultaneous Branch)

When a processor executes the parallel (simultaneous) branch, the processor scans the branch from left-to-right, top-to-bottom. It appears that the processor executes each path in the branch simultaneously.

In case of connecting by parallel branch, if the transition condition connected to the next is 1, all steps tied to this transition will be activated. The extension of each branch will be the same as serial connection. At this time, the steps in the state of activation are as many as the number of branches.

In case of combining in parallel branch, if the transition condition is 1 when the state of all the last steps of each branch is activated, the step connected to the next will be activated.

#### Example



- If the transition condition of **T1** is 1 when **S1** is active, **S2**, **S6** and **S8** will be activated and **S1** will be deactivated.
- If the transition condition of **T4** is 1 when **S4**, **S7** and **S8** are activated, **S5** will be activated and **S4**, **S7** and **S8** will be deactivated.

\* The order of **activation**

```

S1-->S2-->S3-->S4-->S5
  +->S6-->S7-----+
  +->S8-----+
  
```

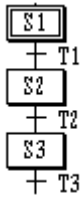
### 4.3.4. Jump

If the transition condition connected to the next is 1 after the last step of SFC is activated, the initial step of SFC will be activated.

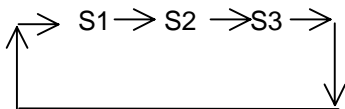
## 4. SFC

---

### Example



\* The order of activation



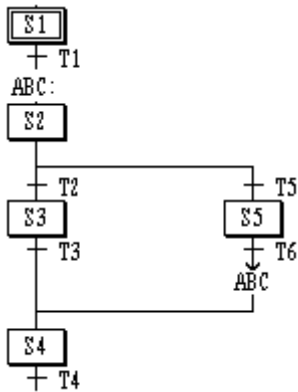
It is possible to extend to the place using a jump.

Jump can only be placed at the end of SFC program or the end of a selection branch.

It is not allowed to jump into the inside or outside of parallel branch; it is allowed to jump within parallel branch.

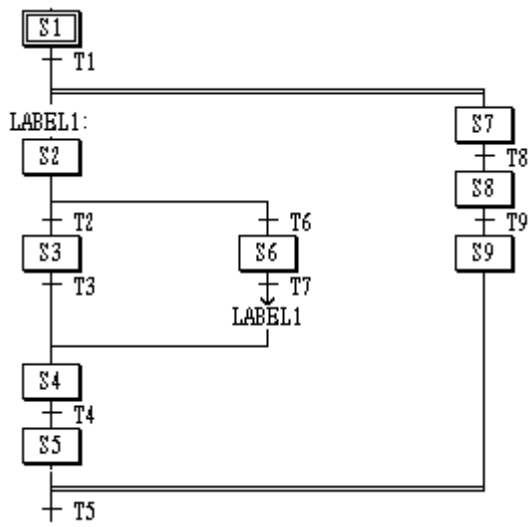
### Example

1) Jump at the end of selection branch

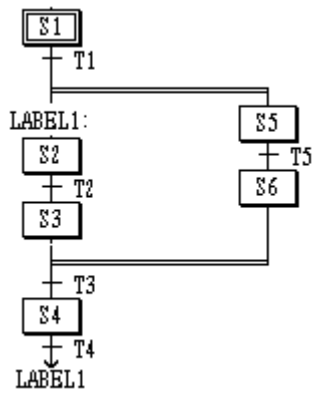


- **S2** will be activated after **S5**.

2) Jump within parallel branch



3) Not available to jump into the inside of parallel branch..



MEMO



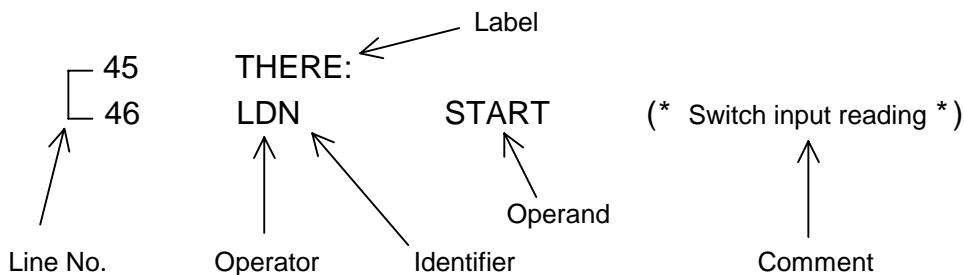
## 5. IL (Instruction List)

### 5.1. Overview

IL is a low-level 'assembler like' language.

IL is applicable to simple PLC systems.

Type



### 5.2. Current Result: CR

In IL, there is a register that stores an operation result by that time, which is called "CR (current result)".

Only one CR exists in IL.

CR is able to be any data type.

The operator that puts a certain value to CR and determines its data type is LD (Load).

#### **Example**

LD %IX0.0.0 is to put the value of %IX0.0.0 to the CR. Now, the data type of CR is BOOL because the data type expressed as X is BOOL. If variable VAL is declared as INT and is written as LD VAL, it writes the value of VAL to CR and the data type of CR is INT.

ST operator stores the current result (CR) in a variable.

#### **Example**

If variable VAL is declared as INT and is written as ST VAL, this means that CR is stored in variable VAL. At this time, the data type of CR should be INT. Unless CR is an INT type, an error occurs when compiling.

## 5. IL

---

Please read the following:

```
LD      %IX0.0.0
```

```
ST      VAL  (assume that variable VAL is declared as INT)
```

CR is assigned as BOOL in the first row and declared as INT in the second row, which results in an error when compiling.

```
LD      %IX0.0.0
```

```
ST      START
```

```
LD      20
```

```
ST      VAL  (assume that variable START is declared as BOOL and variable VAL as INT)
```

The above example is executed normally because the data type to store CR respectively is the same.

### 5.3. Instructions

IL is a list of instructions.

Each instruction must begin on a new line, and must contain an operator, completed with optional modifiers and, if necessary, for the specific operation, one or more operands, separated with commas (',').

#### 5.3.1. Label

A label followed by a colon (':') may precede the instruction.

Labels are used as operands for some operations such as jumps.

#### 5.3.2. Modifier

The modifier character must complete the name of the operator, with no blank characters between them. There're three types of modifiers: N, (, C.

The N modifier indicates a Boolean negation of the operand.

#### **Example**

ANDN %IX2.0.0 is interpreted as:

```
CR <= CR AND NOT %IX2.0.0
```

When N is attached to JMP, CAL and RET with no blank character between them, this means it executes the instruction when CR is BOOL 0.

Modifier '(' delays the operation of an operator until it meets operator ')'.  
As there is only one CR in IL, it is available to execute the delayed operation: CR is kept while other operations are executed and after that, operation will be done with the stored CR value.

Type	Characteristic	Semantics
(	Modifier	Operation is delayed.
)	Operator	Evaluation deferred operation used with '('



**Example**

```
AND( %IX1.0.0
OR   %IX2.0.0 )
CR <= CR AND (%IX1.0.0 OR %IX2.0.0)
```

This means that the execution of AND will be delayed until ')' appears. After the operation inside the parentheses, %IX1.0.0 OR %IX2.0.0, is executed, the operation with the result will be done.

Modifier 'C' indicates that the attached instruction must be executed only if the current result has the Boolean value 1 (TRUE).

**Example**

```
JMPC   THERE
If CR is BOOL 1, jump to THERE.
```

**5.3.3. Basic Operator**

Basic operators are as follows:

No.	Operator	Modifier	Operand	Semantics
1	LD	N	Data	Set current results equal to operand
2	ST	N	Data	Store current results to operand
3	S		BOOL	If CR is BOOL 1, set Boolean Operand to 1
	R		BOOL	If CR is BOOL 1, set Boolean Operand to 0
4	AND	N,(	Data	Boolean AND operation
5	OR	N,(	Data	Boolean OR operation
6	XOR	N,(	Data	Boolean XOR operation
7	ADD	(	Data	Addition operation
8	SUB	(	Data	Subtraction operation
9	MUL	(	Data	Multiplication operation
10	DIV	(	Data	Division operation
11	GT	(	Data	Comparison operation: > (greater than)
12	GE	(	Data	Comparison operation: >= (greater than or equal to)
13	EQ	(	Data	Comparison operation: = (equal to)
14	NE	(	Data	Comparison operation: <> (not equal)
15	LE	(	Data	Comparison operation: <= (less than or equal to)
16	LT	(	Data	Comparison operation: < (less than)
17	JMP	C, N	Label	Jump to label
18	CAL	C, N	Name	Call a function or function block
19	RET	C, N		Return from a function or function block
20	)			Evaluation deferred operation used with '('

Operators from no. 4 to 16 execute the following functions:

### **CR <== CR Operation Operand**

After executing the operation made between CR and operand value is done, it stores the result in CR.

#### **Example**

AND %IX1.0.0 is interpreted as follows:

CR <= CR AND %IX1.0.0

Comparison operator stores its Boolean result in CR after a comparison operation made between CR and the right operand.

#### **Example**

For GT %MW10, if CR is greater than the value of internal memory word 10, the value of CR will be BOOL 1. Otherwise it will be 0.

The data type of CR is not modified by most of the operation instructions. However, in case of comparison operators, a data type of CR is changed.

#### **Example**

```
LD    VAL    (a)
EQ    GROSS  (b)
AND   %IX0.0.0 (c)
ST    START  (d)
```

(assume that variable START is declared as BOOL, and variable VAL and GROSS as INT)

At (a) row, the INT value of VAL is put in CR. At (b) row, after comparing the CR to INT value of GROSS, if the value is same, it puts BOOL 1 in CR; if not, CR is BOOL 0. At this time, a data type of CR changes from INT to BOOL. Accordingly, instructions of (c) and (d) rows are normal without making an error.

### 5.3.3.1. Basic Operator

#### (1) LD

Meaning	It loads a value in the current result. A data type of CR changes according to the operand data type.	
Modifier	N: If the operand is BOOL, it negates its value and loads it in CR.	
Operand	All the data types including constant are available.	
Examples	LD TRUE	The value of BOOL 1 is loaded in CR. The data type of CR is BOOL.
	LD INT_VALUE	The value of INT_VALUE is loaded in CR. The data type of CR is INT.
	LD T#1S	T#1S, time constant, is loaded in CR. The data type of CR is TIME.
	LDN B_VALUE	The value of B_VALUE is negated and is loaded in CR. The data type of CR is BOOL.

#### (2) ST

Meaning	It stores the current result (CR) in a variable (operand). The data type of both CR and operand should be the same. The current result is not modified by this operation.	
Modifier	N: If CR is BOOL, it negates its value and stores it in the operand. At this time, the value of CR does not change.	
Operand	All the data types except constant are available. Its data type should be the same as that of CR.	
Examples	LD FALSE	The value of BOOL 0 is loaded in CR. The data type of CR is BOOL.
	ST B_VALUE1	Stores the value of CR in variable B_VALUE1 of which data type is BOOL.
	STN B_VALUE2	Negates the value of CR and stores it in B_VALUE2 of which data type is BOOL.
	LD INT_VALUE	The value of INT_VALUE that is INT variable is loaded in CR. The data type of CR is INT.
	ST I_VALUE1	Stores the value of CR in variable I_VALUE1 of which data type is INT.
	LD D#1995-12-25	Date constant D#1995-12-25 is loaded in CR. At this time, a data type of CR is DATE.
	ST D_VALUE1	Stores the value of CR in variable D_VALUE1 of which data type is DATE.

(3) S (Set)

Meaning	<p>If CR is BOOL 1, the operand value of which data type is BOOL will be 1.                  No operation is processed if CR is BOOL 0.                  The current result is not modified by this operation.</p>	
Modifier	None	
Operand	<p>Only BOOL data type is available.                  Constant is not available.</p>	
Examples	<p>LD FALSE</p> <p>S B_VALUE1</p> <p>LD TRUE</p> <p>S B_VALUE2</p>	<p>The value of BOOL 0 is loaded in CR. At this time, a data type of CR is BOOL.</p> <p>No operation is processed because CR is 0.                  The value of B_VALUE1 does not change.</p> <p>The value of BOOL 1 is loaded in CR. At this time, a data type of CR is BOOL.</p> <p>As CR is 1, the value of B_VALUE2 whose data type is BOOL will be 1.</p>

(4) R (Reset)

Meaning	<p>If CR is BOOL 1, the operand value whose data type is BOOL will be 0.                  No operation is processed if CR is BOOL 0.                  The current result is not modified by this operation.</p>	
Modifier	None	
Operand	<p>Only BOOL data type is available.                  Constant is not available.</p>	
Examples	<p>LD FALSE</p> <p>R B_VALUE1</p> <p>LD TRUE</p> <p>R B_VALUE2</p> <p>ST B_VALUE3</p>	<p>The value of BOOL 0 is loaded in CR. At this time, a data type of CR is BOOL.</p> <p>No operation is processed because CR is 0.                  The value of B_VALUE1 does not change.</p> <p>The value of BOOL 1 is loaded in CR. At this time, a data type of CR is BOOL.</p> <p>As CR is 1, the value of B_VALUE2 whose data type is BOOL will be 0. The value of CR does not change.</p> <p>The value of CR (Boolean 1) is stored in B_VALUE3 whose data type is BOOL.</p>

## (5) AND

Meaning	After logical AND operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	N: If the operand data type is BOOL, logical AND operation is made between the operand value and CR after negating the operand value. (: If a data type of operand is BOOL, moves CR value in other place for a while and stores the operand value in CR (deferred operation).	
Operand	Only BOOL, BYTE, WORD, DWORD, DWORD, LWORD data types are available. Constant is also available.	
Examples	LD B_VALUE1 AND B_VALUE2 ANDN B_VALUE3 ST B_VALUE4  LD W_VALUE1 AND W_VALUE2 ST W_VALUE3  LD B_VALUE1 AND( B_VALUE2 OR B_VALUE3 ) ST B_VALUE4	The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL. After logical AND operation for CR and the value of B_VALUE2 whose data type is BOOL, stores the result in CR. After negating the value of B_VALUE3, logical AND operation is made between CR and the value of B_VALUE3 whose data type is BOOL. Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 AND B_VALUE2 AND NOT (B_VALUE3)  The value of W_VALUE1 whose data type is WORD is loaded in CR. At this time, a data type of CR is WORD. After logical AND operation for CR and the value of W_VALUE2 whose data type is WORD, stores the result in CR. Stores CR value in W_VALUE3 whose data type is WORD. W_VALUE3 <== W_VALUE1 AND W_VALUE2  The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL. Moves CR value in other place and stores the value of B_VALUE2 whose data type is BOOL in CR. After logical OR operation for CR and the value of B_VALUE3 whose data type is BOOL, stores the result in CR. After logical AND operation for the current CR value and the moved CR value stored in other place, stores the result in CR. Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 AND (B_VALUE2 OR B_VALUE3)

(6) OR

Meaning	After logical OR operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	N: If the operand data type is BOOL, logical AND operation is made between the operand value and CR after negating the operand value. (: If a data type of operand is BOOL, moves CR value in other place for a while and stores the operand value in CR (deferred operation).	
Operand	Only BOOL, BYTE, WORD, DWORD, LWORD data types are available. Constant is also available.	
Examples	LD B_VALUE1  OR B_VALUE2  ORN B_VALUE3  ST B_VALUE4  LD W_VALUE1  OR W_VALUE2  ST W_VALUE3  LD B_VALUE1  OR( B_VALUE2  AND B_VALUE3  )  ST B_VALUE4	The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL.  After logical OR operation for CR and the value of B_VALUE2 whose data type is BOOL, stores the result in CR.  After negating the value of B_VALUE3, logical OR operation is made between CR and the value of B_VALUE3 whose data type is BOOL.  Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 OR B_VALUE2 OR NOT (B_VALUE3)  The value of W_VALUE1 whose data type is WORD is loaded in CR. At this time, a data type of CR is WORD.  After logical AND operation for CR and the value of W_VALUE2 whose data type is WORD, stores the result in CR.  Stores CR value in W_VALUE3 whose data type is WORD. W_VALUE3 <== W_VALUE1 OR W_VALUE2  The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL.  Moves CR value in other place and stores the value of B_VALUE2 whose data type is BOOL in CR.  After logical AND operation for CR and the value of B_VALUE3 whose data type is BOOL, stores the result in CR.  After logical OR operation for the current CR value and the moved CR value stored in other place, stores the result in CR.  Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 OR (B_VALUE2 AND B_VALUE3)

## (7) XOR

Meaning	After logical XOR operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	N: If the operand data type is BOOL, logical AND operation is made between the operand value and CR after negating the operand value. (: If a data type of operand is BOOL, moves CR value in other place for a while and stores the operand value in CR (deferred operation).	
Operand	Only BOOL, BYTE, WORD, DWORD, LWORD data types are available. Constant is also available.	
Examples	LD B_VALUE1	The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL.
	XOR B_VALUE2	After logical XOR operation for CR and the value of B_VALUE2 whose data type is BOOL, stores the result in CR.
	XORN B_VALUE3	After negating the value of B_VALUE3, logical XOR operation is made between CR and the value of B_VALUE3 whose data type is BOOL.
	ST B_VALUE4	Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 XOR B_VALUE2 XOR NOT (B_VALUE3)
	LD W_VALUE1	The value of W_VALUE1 whose data type is WORD is loaded in CR. At this time, a data type of CR is WORD.
	XOR W_VALUE2	After logical XOR operation for CR and the value of W_VALUE2 whose data type is WORD, stores the result in CR.
	ST W_VALUE3	Stores CR value in W_VALUE3 whose data type is WORD. W_VALUE3 <== W_VALUE1 XOR W_VALUE2
	LD B_VALUE1	The value of B_VALUE1 whose data type is BOOL is loaded in CR. At this time, a data type of CR is BOOL.
	XOR( B_VALUE2	Moves CR value in other place and stores the value of B_VALUE2 whose data type is BOOL in CR.
	AND B_VALUE3	After logical AND operation for CR and the value of B_VALUE3 whose data type is BOOL, stores the result in CR.
	)	After logical XOR operation for the current CR value and the moved CR value stored in other place, stores the result in CR.
	ST B_VALUE4	Stores CR value in B_VALUE4 whose data type is BOOL. B_VALUE4 <== B_VALUE1 XOR (B_VALUE2 AND B_VALUE3)

(8) ADD

Meaning	After addition operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	(: Moves CR value in other place for a while and stores the operand value in CR (deferred operation)).	
Operand	Only SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL data types are available. Constant is also available.	
Examples	LD I_VALUE1  ADD I_VALUE2  ST I_VALUE3  LD D_VALUE1  ADD( D_VALUE2  DIV D_VALUE3  )  ST D_VALUE4	The value of I_VALUE1 whose data type is INT is loaded in CR. At this time, a data type of CR is INT.  After ADD operation for CR and the value of I_VALUE2 whose data type is INT, stores the result in CR.  Stores CR value in I_VALUE3 whose data type is INT. I_VALUE3 <== I_VALUE1 + I_VALUE2  The value of D_VALUE1 whose data type is DINT is loaded in CR. At this time, a data type of CR is DINT.  Moves CR value in other place and stores the value of D_VALUE2 whose data type is DINT in CR.  After DIV operation for CR and the value of D_VALUE3 whose data type is DINT, stores the result in CR.  After ADD operation for the current CR value and the moved CR value stored in other place, stores the result in CR.  Stores the CR value in D_VALUE4 whose data type is DINT. D_VALUE4 <== D_VALUE1 + (D_VALUE2 / D_VALUE3)



## (9) SUB

Meaning	After subtraction operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	(: Moves CR value in other place for a while and stores the operand value in CR (deferred operation)).	
Operand	Only SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL data types are available. Constant is also available.	
Examples	LD I_VALUE1  SUB I_VALUE2  ST I_VALUE3  LD D_VALUE1 SUB( D_VALUE2 MUL D_VALUE3 ) ST D_VALUE4	The value of I_VALUE1 whose data type is INT is loaded in CR. At this time, a data type of CR is INT. After SUB operation for CR and the value of I_VALUE2 whose data type is INT, stores the result in CR. Stores CR value in I_VALUE3 whose data type is INT. $I\_VALUE3 \leftarrow I\_VALUE1 - I\_VALUE2$ The value of D_VALUE1 whose data type is DINT is loaded in CR. At this time, a data type of CR is DINT. Moves CR value in other place and stores the value of D_VALUE2 whose data type is DINT in CR. After MUL operation for CR and the value of D_VALUE3 whose data type is DINT, stores the result in CR. After SUB operation for the current CR value and the moved CR value stored in other place, stores the result in CR. Stores the CR value in D_VALUE4 whose data type is DINT. $D\_VALUE4 \leftarrow D\_VALUE1 - (D\_VALUE2 \times D\_VALUE3)$

(10) MUL

Meaning	After multiplication operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	(: Moves CR value in other place for a while and stores the operand value in CR (deferred operation).	
Operand	Only SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL data types are available. Constant is also available.	
Examples	LD I_VALUE1  MUL I_VALUE2  ST I_VALUE3  LD D_VALUE1  MUL( D_VALUE2  SUB D_VALUE3  )  ST D_VALUE4	The value of I_VALUE1 whose data type is INT is loaded in CR. At this time, a data type of CR is INT.  After MUL operation for CR and the value of I_VALUE2 whose data type is INT, stores the result in CR.  Stores CR value in I_VALUE3 whose data type is INT. I_VALUE3 <== I_VALUE1 X I_VALUE2  The value of D_VALUE1 whose data type is DINT is loaded in CR. At this time, a data type of CR is DINT.  Moves CR value in other place and stores the value of D_VALUE2 whose data type is DINT in CR.  After SUB operation for CR and the value of D_VALUE3 whose data type is DINT, stores the result in CR.  After MUL operation for the current CR value and the moved CR value stored in other place, stores the result in CR.  Stores the CR value in D_VALUE4 whose data type is DINT. D_VALUE4 <== D_VALUE1 X (D_VALUE2 - D_VALUE3)

## (11) DIV

Meaning	After division operation for CR and the operand value, stores the operation result in CR. At this time, a data type of both CR and the operand should be the same. The operand value does not change.	
Modifier	(: Moves CR value in other place for a while and stores the operand value in CR (deferred operation)).	
Operand	Only SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL data types are available. Constant is also available.	
Examples	LD I_VALUE1	The value of I_VALUE1 whose data type is INT is loaded in CR. At this time, a data type of CR is INT.
	DIV I_VALUE2	After DIV operation for CR and the value of I_VALUE2 whose data type is INT, stores the result in CR.
	ST I_VALUE3	Stores CR value in I_VALUE3 whose data type is INT. $I\_VALUE3 \leftarrow I\_VALUE1 / I\_VALUE2$
	LD D_VALUE1	The value of D_VALUE1 whose data type is DINT is loaded in CR. At this time, a data type of CR is DINT.
	DIV( D_VALUE2	Moves CR value in other place and stores the value of D_VALUE2 whose data type is DINT in CR.
	ADD D_VALUE3	After ADD operation for CR and the value of D_VALUE3 whose data type is DINT, stores the result in CR.
	)	After DIV operation for the current CR value and the moved CR value stored in other place, stores the result in CR.
	ST D_VALUE4	Stores the CR value in D_VALUE4 whose data type is DINT. $D\_VALUE4 \leftarrow D\_VALUE1 / (D\_VALUE2 + D\_VALUE3)$

(12) GT

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All the data types except ARRAY are available. Constant is also available.	
Examples		In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 70,
	LD I_VAL1	The value of I_VAL1 whose data type is INT is loaded in CR.
	GT I_VAL2	After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 < I_VAL2, CR will be 0)
	ST B_VAL1	Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 <== FALSE
	LD I_VAL2	The value of I_VAL2 whose data type is INT is loaded in CR.
	GT I_VAL1	After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 < I_VAL2, CR will be 1)
	ST B_VAL2	Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 <== TRUE
	LD I_VAL1	The value of I_VAL1 whose data type is INT is loaded in CR.
	GT( I_VAL2	Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR.
	SUB I_VAL3	After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR.
	)	After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR > current CR, CR will be 1)
	ST B_VAL3	Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL3 <== TRUE

## (13) GE

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All the data types except ARRAY are available. Constant is also available.	
Examples		<p>In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 70,</p> <p>LD I_VAL1 The value of I_VAL1 whose data type is INT is loaded in CR.</p> <p>GE I_VAL2 After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 0)</p> <p>ST B_VAL1 Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 &lt;== FALSE</p> <p>LD I_VAL2 The value of I_VAL2 whose data type is INT is loaded in CR.</p> <p>GE I_VAL1 After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 1)</p> <p>ST B_VAL2 Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 &lt;== TRUE</p> <p>LD I_VAL1 The value of I_VAL1 whose data type is INT is loaded in CR.</p> <p>GE( I_VAL2 Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR.</p> <p>SUB I_VAL3 After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR.</p> <p>) After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR &gt; current CR, CR will be 1)</p> <p>ST B_VAL3 Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL3 &lt;== TRUE</p>

(14) EQ

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All the data types except ARRAY are available. Constant is also available.	
Examples	<pre>LD   I_VAL1 EQ   I_VAL2  ST   B_VAL1  LD   I_VAL1 EQ(  I_VAL3  ST   B_VAL2  LD   I_VAL1 EQ(  I_VAL2  SUB  I_VAL3 )  ST   B_VAL3</pre>	<p>In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 50,</p> <p>The value of I_VAL1 whose data type is INT is loaded in CR. After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 0)</p> <p>Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 &lt;== FALSE</p> <p>The value of I_VAL2 whose data type is INT is loaded in CR. After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 = I_VAL3, CR will be 1)</p> <p>Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 &lt;== TRUE</p> <p>The value of I_VAL1 whose data type is INT is loaded in CR. Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR. After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR. After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR = current CR, CR will be 1)</p> <p>Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL3 &lt;== TRUE</p>

(15) NE

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All data types except ARRAY are available. Constant is also available.	
Examples		In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 50,
	LD I_VAL1	The value of I_VAL1 whose data type is INT is loaded in CR.
	NE I_VAL3	After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 = I_VAL3, CR will be 0)
	ST B_VAL1	Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 <== FALSE
	LD I_VAL1	The value of I_VAL2 whose data type is INT is loaded in CR.
	NE I_VAL2	After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 <> I_VAL2, CR will be 1)
	ST B_VAL2	Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 <== TRUE
	LD I_VAL1	The value of I_VAL1 whose data type is INT is loaded in CR.
	NE( I_VAL2	Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR.
	SUB I_VAL3	After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR.
	)	After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR = current CR, CR will be 0)
	ST B_VA3	Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL2 <== FALSE

(16) LE

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All data types except ARRAY are available. Constant is also available.	
Examples		<p>In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 70,</p> <p>LD I_VAL2 LE I_VAL1 ST B_VAL1</p> <p>The value of I_VAL2 whose data type is INT is loaded in CR. After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 0) Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 &lt;== FALSE</p> <p>LD I_VAL1 LE I_VAL2 ST B_VAL2</p> <p>The value of I_VAL1 whose data type is INT is loaded in CR. After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 1) Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 &lt;== TRUE</p> <p>LD I_VAL1 LE( I_VAL2 SUB I_VAL3 ) ST B_VA3</p> <p>The value of I_VAL1 whose data type is INT is loaded in CR. Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR. After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR. After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR &gt; current CR, CR will be 0) Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL2 &lt;== FALSE</p>



(17) LT

Meaning	After comparison operation for CR and the operand value, stores the BOOL result in CR. CR will be 1 only if CR is greater than operand. A data type of both CR and the operand should be the same. The operand value does not change. After operation, a data type of CR will be BOOL regardless of the operand data type.	
Modifier	(: Moves CR value in other place for a while and stores the value of operand in CR (deferred operation)).	
Operand	All data types except ARRAY are available. Constant is also available.	
Examples		<p>In case that I_VAL1 = 50, I_VAL2 = 100 I_VAL3 = 70,</p> <p>LD I_VAL2 The value of I_VAL2 whose data type is INT is loaded in CR.</p> <p>LT I_VAL1 After comparison operation for CR and the value of I_VAL1 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 0)</p> <p>ST B_VAL1 Stores CR value in B_VAL1 whose data type is BOOL. B_VAL1 &lt;== FALSE</p> <p>LD I_VAL1 The value of I_VAL1 whose data type is INT is loaded in CR.</p> <p>LT I_VAL2 After comparison operation for CR and the value of I_VAL2 whose data type is INT, stores the result in CR. (As I_VAL1 &lt; I_VAL2, CR will be 1)</p> <p>ST B_VAL2 Stores CR value in B_VAL2 whose data type is BOOL. B_VAL2 &lt;== TRUE</p> <p>LD I_VAL1 The value of I_VAL1 whose data type is INT is loaded in CR.</p> <p>LT( I_VAL2 Moves CR value in other place and stores the value of I_VAL2 whose data type is INT in CR.</p> <p>SUB I_VAL3 After SUB operation for CR and the value of I_VAL3 whose data type is INT, stores the result in CR.</p> <p>) After comparison operation for the current CR value and the moved CR value stored in other place, stores the result in CR. (As the stored CR &gt; current CR, CR will be 0)</p> <p>ST B_VA3 Stores the CR value in B_VAL3 whose data type is BOOL. B_VAL2 &lt;== FALSE</p>

(18) JMP

Meaning	Jumps to the specified label.	
Modifier	<p>C: If CR whose data type is BOOL is TRUE (1), it jumps to the specified label.          If CR whose data type is BOOL is FALSE (0), it does not jump to the specified label but executes the next instruction.</p> <p>N: If CR whose data type is BOOL is FALSE (0), it jumps to the specified label.          If CR whose data type is BOOL is TRUE (1), it does not jump to the specified label but executes the next instruction.</p> <p>If there is no modifier, it jumps to the label regardless of CR value.</p>	
Operand	Label defined in the same IL program.	
Examples	<pre>LD      B_VAL1 JMP     THERE2 THERE1: LD      I_VAL2 THERE2: ST      I_VAL3  LD      B_VAL2 JMPN   THERE3 LD      B_VALUE SEL     G:=  CURRENT         RESULT     IN1:= I_VAL1     IN2:= I_VAL2 ST      I_VAL3 THERE3:</pre>	<p>This is a program that stores the value of I_VAL1 or I_VAL2 in I_VAL3 according to the value of B_VAL1 whose data type is BOOL.</p> <p>The value of B_VAL1 whose data type is BOOL is loaded in CR.</p> <p>If CR is 1, it jumps to THERE1 label; if CR is 0, it executes the next instruction.</p> <p>CR &lt;== I_VAL1</p> <p>Jumps to THERE2 label unconditionally.</p> <p>THERE1 label</p> <p>CR &lt;== I_VAL2</p> <p>THERE2 label</p> <p>I_VAL3 &lt;== CR</p> <p>This is a program that executes SEL function if the value of B_VAL2 whose data type is BOOL is 1.</p> <p>CR &lt;== B_VAL2</p> <p>If CR is 0 (FALSE), it jumps to THERE3 label.</p> <p>CR &lt;== B_VALUE</p> <p>Calls SEL function.</p> <p>I_VAL3 &lt;== CR</p> <p>THERE3 label</p>

## (19) CAL

Meaning	Calls the function block whose name is described in the operand section.	
Modifier	<p>C: if CR whose data type is BOOL is TRUE (1), it calls a function block.  If CR whose data type is BOOL is FALSE (0), it does not call a function block.</p> <p>N : if CR whose data type is BOOL is FALSE (0), it calls a function block.  If CR whose data type is BOOL is TRUE (1), it does not call a function block.</p> <p>If there is no modifier, it calls a function block regardless of CR.</p>	
Operand	Function block name	
Examples	<pre>LD      B_VAL1  CALC TON  TIMER1       IN:= T_INPUT       PT:= PRE_TIME  LD      B_VAL2  CALN CTU  COUNT1       CU:= B_UP       R:=  B_RESET       PV:= 100  CAL  CTD  COUNT2       CD:= B_DOWN       LD:= B_LDV       PV:= 300</pre>	<p>This is a program that if the value of B_VAL1 whose data type is BOOL is 1(TRUE), calls the TON (on-delay timer).  The value of B_VAL1 whose data type is BOOL is loaded in CR.  If CR is 1, it calls the on-delay timer, TON whose instance is TIMER1.</p> <p>This is a program that calls the CTU, (up counter), if the value of B_VAL2 whose data type is BOOL is 0 (FALSE).  The value of B_VAL2 whose data type is BOOL is loaded in CR.  If CR is 1, it calls the CTU (up counter) whose instance is COUNT1.</p> <p>This is a program that calls the CTD (down-counter) regardless of CR.  Calls the CTD (down-counter) whose instance is COUNT2.</p>

(20) RET

Meaning	Returns from a function or function block.	
Modifier	<p>C: if CR whose data type is BOOL is TRUE (1), it returns.              If CR whose data type is BOOL is FALSE (0), it does not return.</p> <p>N: if CR whose data type is BOOL is FALSE (0), it returns.              If CR whose data type is BOOL is TRUE (1), it does not return.</p> <p>If there is no modifier, it returns regardless of CR.</p>	
Operand	None	
Examples	<pre>LD    I_VAL1 MUL   I_VAL2 ST    I_VAL3 LD    _ERR RETN LD    0 ST    I_VAL3 RET</pre>	<p>This is a function that stores the result in I_VAL3 after MUL operation for the value of I_VAL1 whose data type is INT and the value of I_VAL2 whose data type is INT. At this time, if an operation error occurs in MUL operation, it returns after storing 0 in I_VAL3.</p> <p>CR &lt;== system error flag          If CR is 0, instance will return.</p> <p>I_VAL3 &lt;== 0          Returns unconditionally.</p>

(21) )

Meaning	Evaluation deferred operation used with '('.	
Modifier	None	
Operand	None	
Examples	<pre>LD    I_VAL1 ADD   I_VAL2 MUL   I_VAL3 ST    I_VAL4  LD    I_VAL1 ADD(  I_VAL2 MUL   I_VAL3 ) ST    I_VAL4  LD    L_VAL1 ADD(  L_VAL2 MUL(  L_VAL3 SUB   L_VAL4 ) ADD   L_VAL5 ) DIV   L_VAL6 ST    L_VAL7</pre>	<pre>I_VAL4 &lt;== (I_VAL1 + I_VAL2) X I_VAL3  I_VAL4 &lt;== I_VAL1 + (I_VAL2 X I_VAL3)  L_VAL7 &lt;== (L_VAL1 + (L_VAL2 X (L_VAL3 - L_VAL4 ) + L_VAL5)) / L_VAL6</pre>

#### 5.4. Calling of Function and Function Block

Calls a function using its name as an operator.

When calling a function, CR is stored as the first input.

If a function has more than one input, assign the input value and then call a function.

The output value of a function will be stored in CR.

A data type of CR will be the output data type a function.

##### Example

```
LD          VAL
SIN
ST          RESULT    (VAL and RESULT are regarded as a REAL data type)
```

If you store the value of VAL in CR at the first row and call SIN function at the second row, then the CR value will be stored in SIN function as a first input. And it does not need other inputs because SIN function has only one input, and the output value will be stored in CR after executing SIN function. At the third row, CR will be stored in RESULT variable.

```
LD          %IX0.0.0
SEL  G:=    CURRENT RESULT
          IN0:= VAL1
          IN1:= VAL2
ST  VAL3
```

This is the example of a function that has several inputs. CR is set at the first row and is loaded in SEL function as a first input value. If you assign each value for the rest inputs and call SEL function, the result will be stored in CR and CR value will be stored in variable VAL3.

JMP (JMPN, JMPC) instructions are used to call a function conditionally.

**Example**

```
LD          %IX0.0.0
JMPN       THERE
LD          I_VAL1
ADD  IN1:=  CURRENT RESULT
          IN2:=  I_VAL2
          IN3:=  I_VAL3
ST          I_VAL4
THERE:
```

%IX0.0.0 value is loaded in CR whose data type is BOOL at the first row. And if the value is 0 at the second row, it jumps to THERE: label. If %IX0.0.0 value is 1, it does not execute JMP instruction but does the next row.

When calling a function block, CAL is used as an operator and the instance name as an operand that is previously declared.

```
CAL  INSTANCE /* call a function block unconditionally. */
CALN INSTANCE /* if CR is BOOL 0, call a function block. */
CALC INSTANCE /* if CR is BOOL 1, call a function block. */
```

Here, INSTANCE should be previously declared as an instance of a function block.

CR is not loaded in a function block input. So it is required to assign all the input values necessary for a function block. Besides output value is not stored in CR.

**Example**

On-Delay Timer function block

```
LD          %IX0.0.0
CALC       TON  TIMER0
          IN:=  %IX0.1.2
          PT:=  T#200S
LD          TIMER0.Q
ST         %QX1.0.2
```

(assume that TIMER0 is declared as an instance of TON)

On-delay timer has 2 inputs and calls it after assigning its input values, respectively. If users want to use the result values, they can do it like the fifth row in the above program because the result values are stored in TIMER0.Q and TIMER0.ET respectively.

MEMO

The image shows a spiral-bound notebook page. At the top left, there is a wooden tab with the word "MEMO" written on it. The notebook has a green cover. The page is yellow and features horizontal dotted lines for writing. The spiral binding is on the left side.

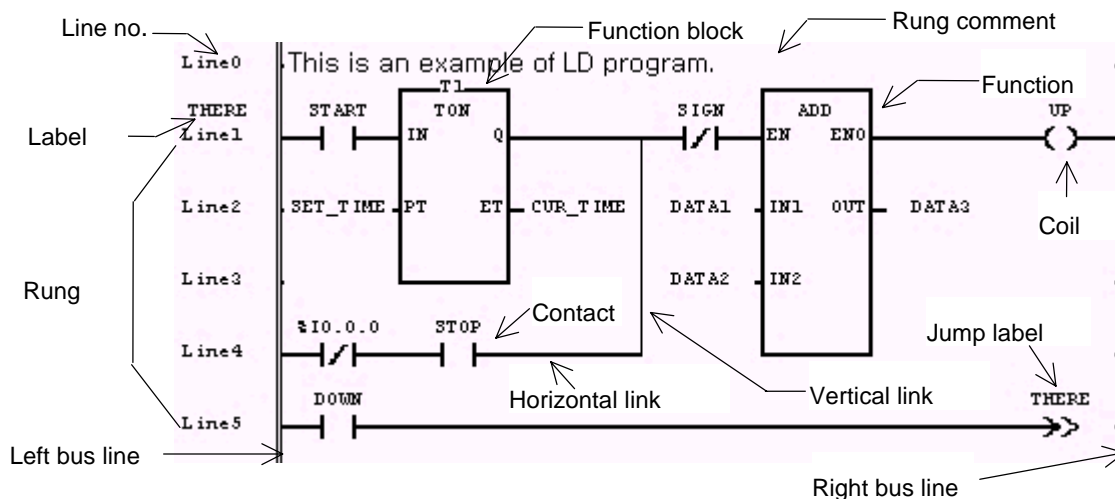


## 6. LD (Ladder Diagram)

### 6.1. Overview



LD program represents PLC program through graphic signs such as coil or contact used in relay logic diagram.

Configuration



### 6.2. Bus Line



Bus line as power line is placed vertically on both left and right sides on LD graphic diagram.

No.	Symbol	Description
1		Left bus line Its value is always 1 (BOOL).
2		Right bus line The value is not fixed.

### 6.3. Connection Line

The value (BOOL 1) of left bus line is transmitted to the right side by the ladder diagram. The line that has the transmitted value is called as 'power flow line' or 'connection line' which is connected to a contact or coil. Power flow line has always a BOOL value and there's only one power flow line in one rung that is connected by lines.

There are two types of a connection line of LD: horizontal connection line and vertical connection line.

No.	Symbol	Description
1		Horizontal connection line It transmits the left side value to the right side.
2		Vertical connection line It's logical OR of horizontal connection lines of its left side.

## 6.4. Contact

'Contact' transmits a value to the right horizontal connection line, which is the result of logical AND operation of these: the state of left horizontal connection line, Boolean input/output related to the current contact, or memory variables. It does not change the value of variable itself related to the contact. Standard contact symbols are as follows:

Static contact		
No.	Symbol	Description
1	$\begin{array}{c} \text{***} \\ \text{--- ---} \end{array}$	<p>Normally open contact</p> <p>When the addressed memory bit (marked with ***) is ON, the instruction is TRUE, which transmits the state of the left connection line to the right one. Otherwise the state of the right connection line is OFF.</p>
2	$\begin{array}{c} \text{***} \\ \text{--- /---} \end{array}$	<p>Normally closed contact</p> <p>When the addressed memory bit (marked with ***) is OFF, the instruction is TRUE, which transmits the state of the left connection line to the right one. Otherwise the state of the right connection line is OFF.</p>
State transition-sensing contact		
3	$\begin{array}{c} \text{***} \\ \text{--- P ---} \end{array}$	<p>Positive transition-sensing contact</p> <p>When the addressed memory bit (marked with ***) that was OFF in the previous scan is ON, it maintains ON state during one scan (current scan).</p>
4	$\begin{array}{c} \text{***} \\ \text{--- N ---} \end{array}$	<p>Negative transition-sensing contact</p> <p>When the addressed memory bit (marked with ***) that was ON in the previous scan is OFF, it maintains ON state during just one scan (current scan).</p>

6.5. Coil

Coil stores the state of the left connection line or the processing result of state transition in the associated BOOL variable. Standard coil symbols are as follows:

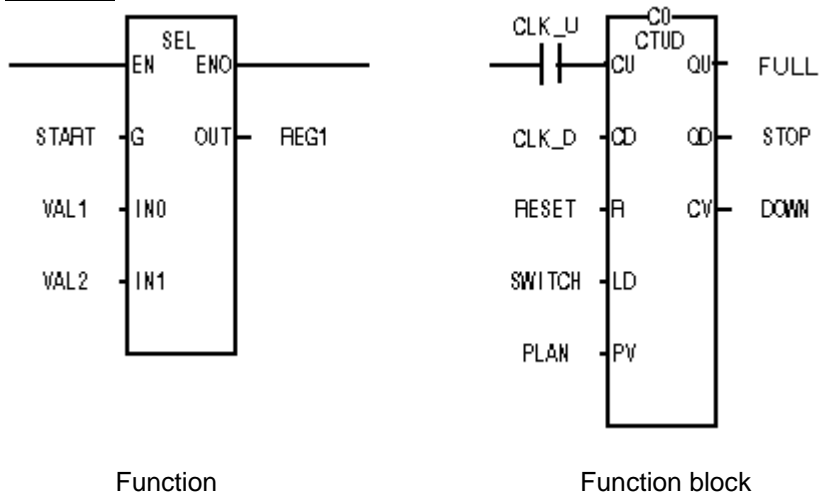
No.	Symbol	Description
Momentary Coils		
1	*** — ( ) —	Coil When the rung is TRUE, the addressed memory bit (marked with ***) is set ON. If the bit controls an output device, that output device will be ON.
2	*** — (/) —	Negated coil When the rung is TRUE, the addressed memory bit (marked with ***) is set OFF. That is, if the state of left connection line is OFF, the associated variable is ON and if the state of left connection line is ON, the associated variable is OFF. If the bit controls an output device, that output device will be OFF.
Latched Coils		
3	*** — (S) —	Set coil It sets the associated variable (marked with ***) to ON when the left link is in the ON state or TRUE and remains set until reset by a Reset coil. When the left link is OFF or FALSE, the associated variable is not affected by the Set coil element.
4	*** — (R) —	Reset coil It sets the associated variable (marked with ***) to OFF when the left link is in the ON state or TRUE and remains reset until set by a Set coil. When the left link is OFF or FALSE, the associated variable is not affected by the Reset coil element.
State Transition-sensing Coils		
5	*** — (P) —	Positive transition-sensing coil If the state of its left connection that was OFF in the previous scan is ON in the current scan, the associated variable (marked with ***) is ON during the current scan.
6	*** — (N) —	Negative transition-sensing coil If the state of its left connection that was ON in the previous scan is OFF in the current scan, the associated variable (marked with ***) is ON during the current scan.

Coils are placed in the rightmost side of LD, of which right side is a right bus line.

## 6.6. Calling of Function and Function Block

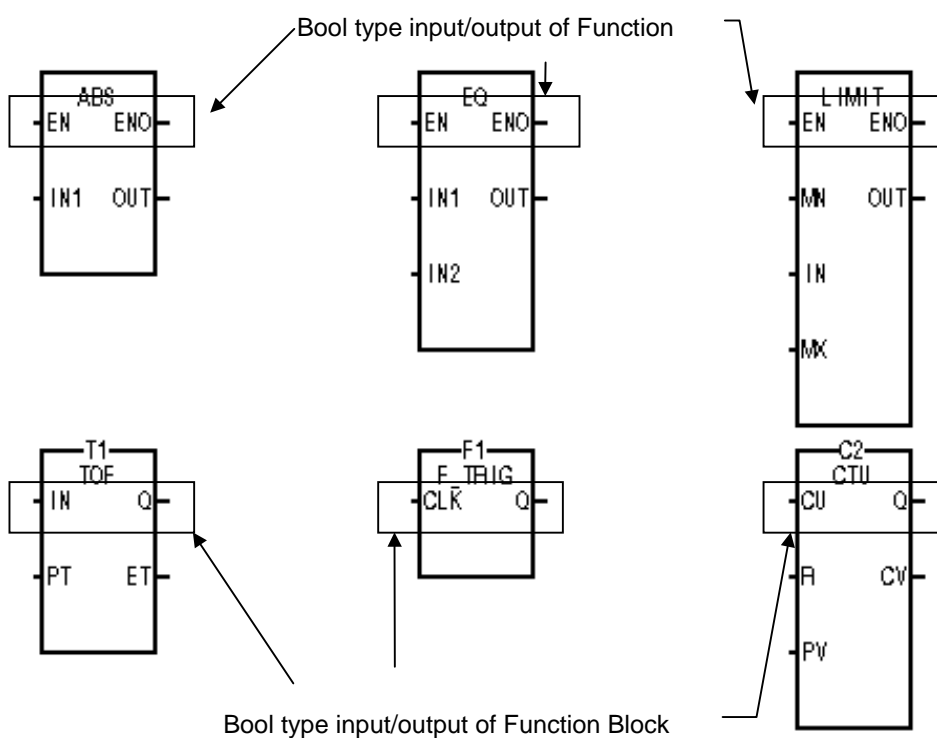
The connection to a function and function block will be done by putting suitable data or variable to their input/output.

### Example



There should be at least one BOOL-type input and BOOL-type output in a function or function block if you want to enable them. EN and ENO are BOOL-type input/output in a function while a data type of the first input and first output are BOOL-type in a function block.

### Example

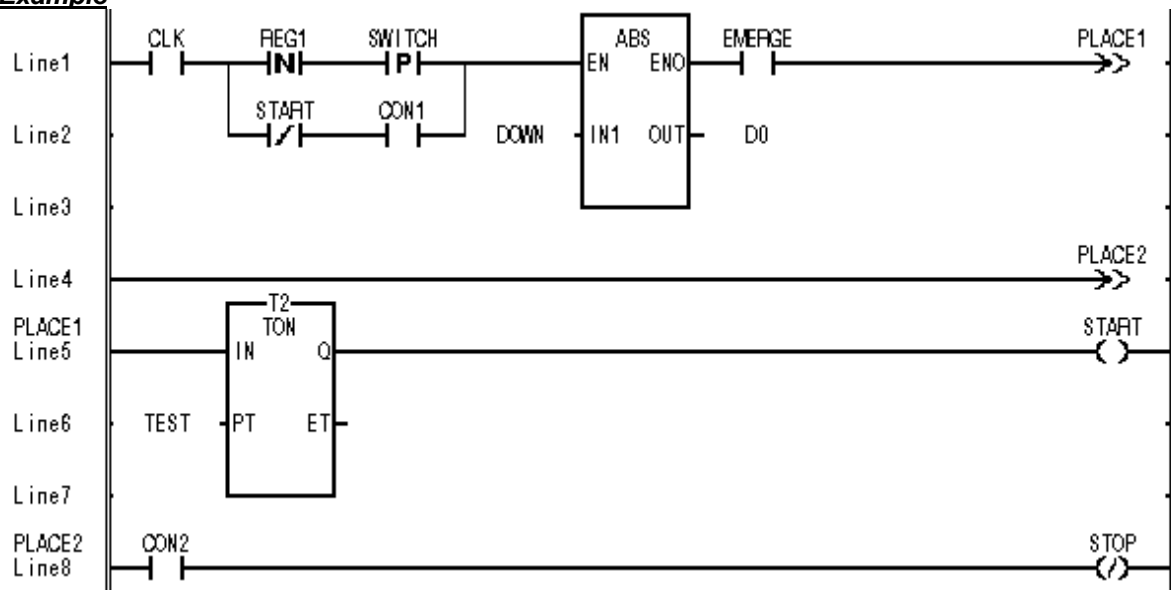


## 6. LD

Function in LD is different from that of IL. By convention the ladder logic connected Boolean input to a function is called EN and the corresponding output Boolean is called ENO, or enable out. If the value of EN is 1, then the function is executed, otherwise it is not executed. In all cases, the default is for the value of EN to be copied to the output ENO. If, for whatever reason, an error occurs in the execution of a function, the function is responsible to set ENO to FALSE (BOOL 0). EN is connected to the power flow line but ENO doesn't have to be connected to it. However, when connecting the power flow line to the function output instead of ENO, output data type should be a BOOL type. Note that only one power flow line can be connected to a function (when connecting the power flow line to the function output not ENO, do not connect anything to ENO output). All the inputs of a function are assigned by entering its data. The output of a function is stored at the output variable in the right side of it.

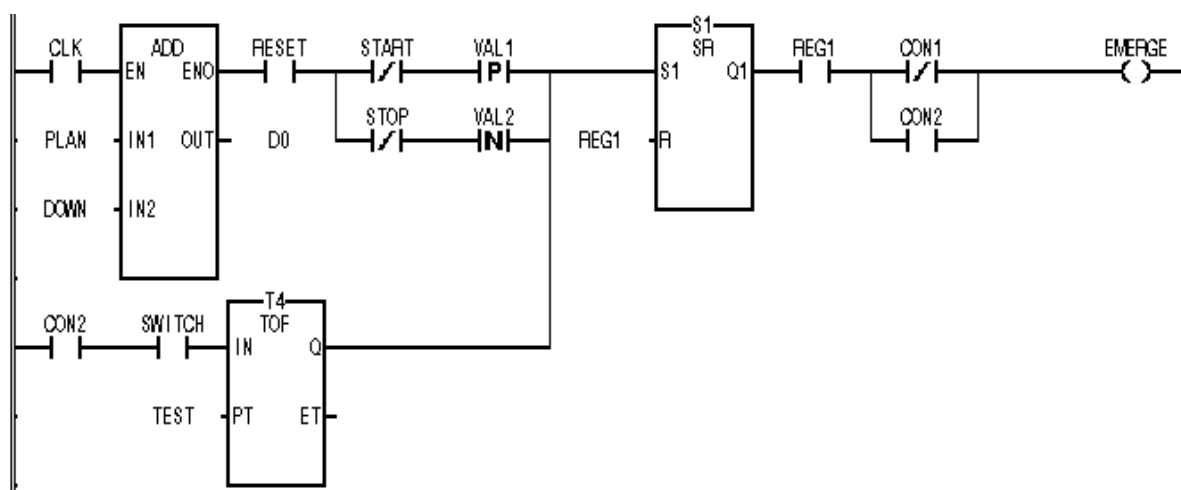
You can use a function block in LD as you do in IL. Inputs of a function block are assigned much the same as a function. A function block is called when the left link is TRUE and not called when the left link is FALSE. The value of the left link IN is copied to the right link Q for further processing. The name of the function block is the "instance" name, which can be user-defined and must be unique to LD in which the function block appears. You don't have to assign output variables because they are in the instance. If a function block is connected to the power flow line, it is always executed because there is neither EN nor ENO in it. Therefore, it is required to use Jump (-->>) to determine whether or not to execute a function block according to the logic result. When connecting the power flow line to the function block, it is required to connect it to the input/output of which data type is BOOL.

### Example



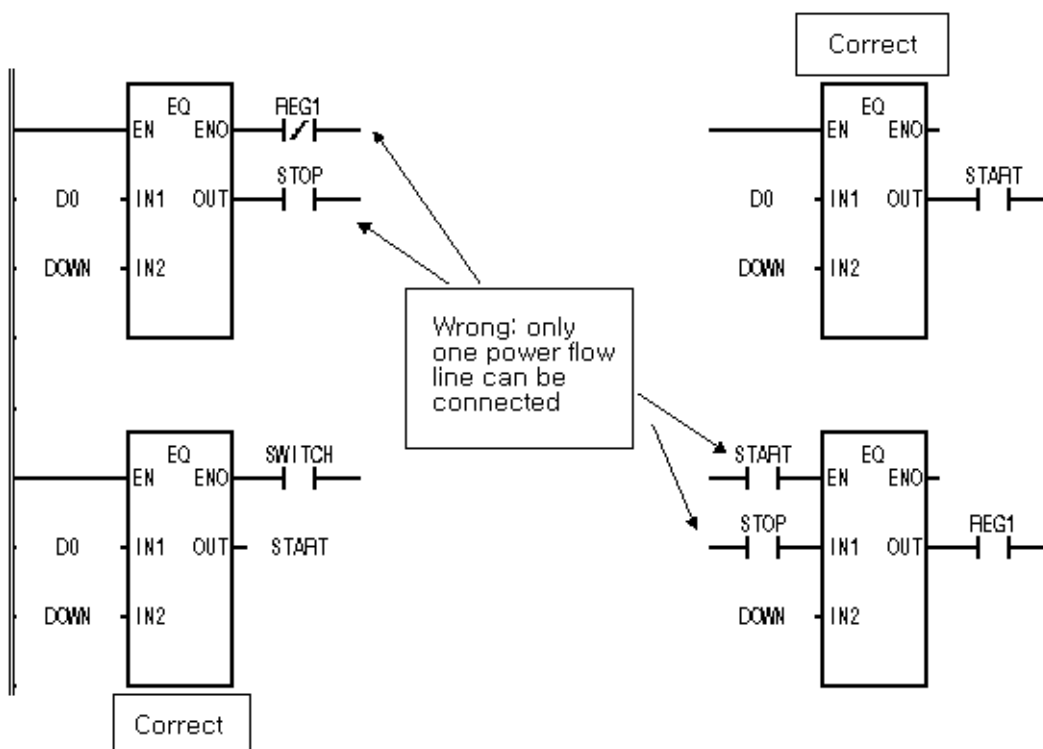
You can place a function and function block in any place of LD. It is available to make a program by connecting the power flow line to their output and then putting the contact to that.

**Example**

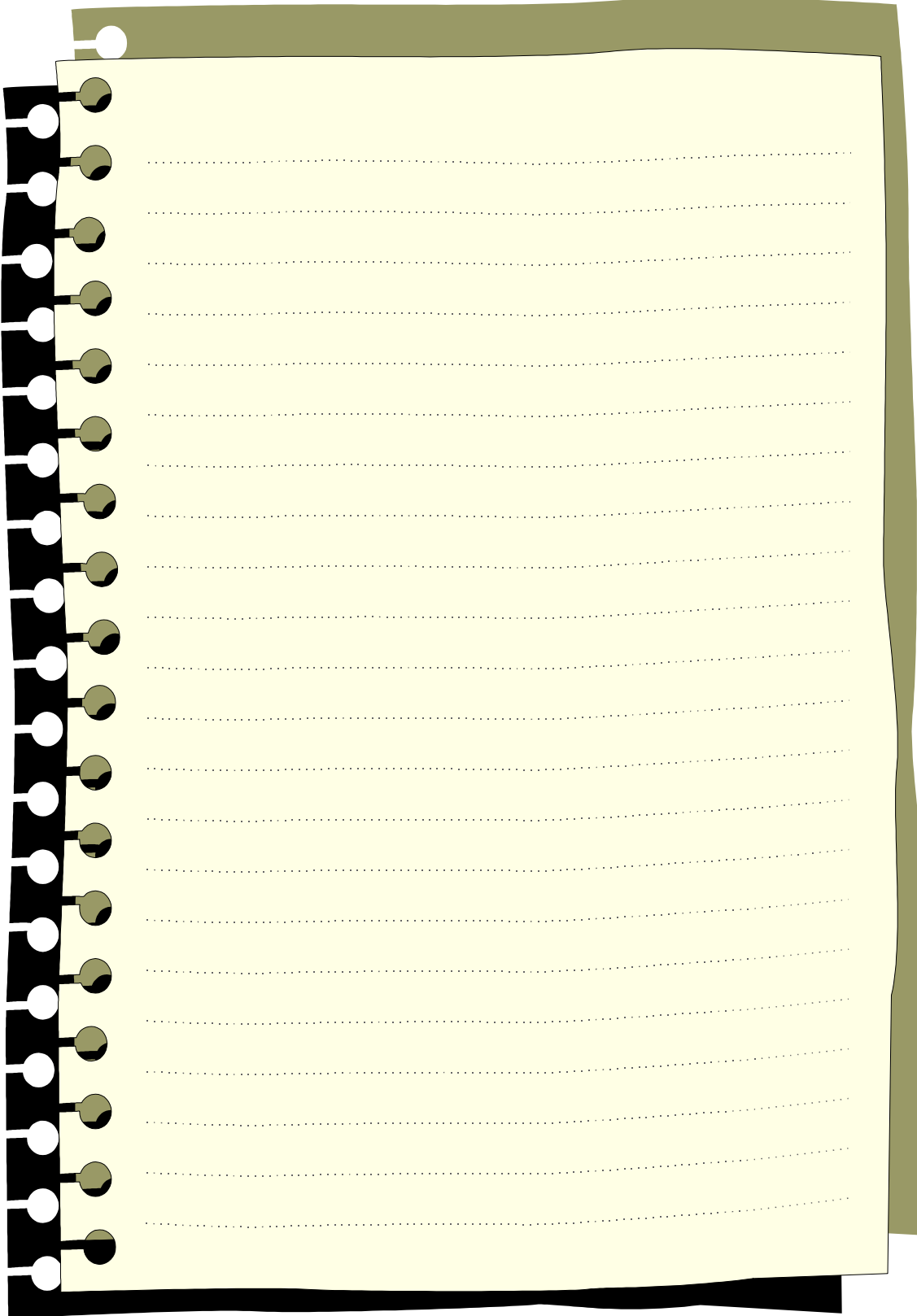


Only one power flow line can be connected to a function or function block.

**Example**



MEMO





## 7. Function and Function Block

It's a list of function and function block. For each function and function block, please refer to the next chapter.

### 7.1. Function

#### 7.1.1. Type Conversion Function

It converts each input data type into an output data type.

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
ARY_ASC_TO_***	ARY_ASC_TO_BYTE	WORD (ASCII)	BYTE			
	ARY_ASC_TO_BCD	WORD (ASCII)	BYTE (BCD)			
ARY_BYTE_TO_***	ARY_BYTE_TO_ASC	BYTE	WORD (ASCII)			
ARY_BCD_TO_***	ARY_BCD_TO_ASC	BYTE (BCD)	WORD (ASCII)			
ASC_TO_***	ASC_TO_BCD	BYTE (BCD)	USINT			
	ASC_TO_BYTE	WORD (BCD)	UINT			
BCD_TO_***	BCD_TO_SINT	BYTE (BCD)	SINT			
	BCD_TO_INT	WORD (BCD)	INT			
	BCD_TO_DINT	DWORD (BCD)	DINT			
	BCD_TO_LINT	LWORD (BCD)	LINT			
	BCD_TO_USINT	BYTE (BCD)	USINT			
	BCD_TO_UINT	WORD (BCD)	UINT			
	BCD_TO_UDINT	DWORD (BCD)	UDINT			
	BCD_TO_ULINT	LWORD (BCD)	ULINT			
TRUNC	TRUNC	REAL	DINT			
		LREAL	LINT			
REAL_TO_***	REAL_TO_SINT	REAL	SINT			
	REAL_TO_INT	REAL	INT			
	REAL_TO_DINT	REAL	DINT			
	REAL_TO_LINT	REAL	LINT			
	REAL_TO_USINT	REAL	USINT			
	REAL_TO_UINT	REAL	UINT			
	REAL_TO_UDINT	REAL	UDINT			
	REAL_TO_ULINT	REAL	ULINT			
	REAL_TO_DWORD	REAL	DWORD			
REAL_TO_LREAL	REAL	LREAL				
LREAL_TO_***	LREAL_TO_SINT	LREAL	SINT			
	LREAL_TO_INT	LREAL	INT			
	LREAL_TO_DINT	LREAL	DINT			
	LREAL_TO_LINT	LREAL	LINT			
	LREAL_TO_USINT	LREAL	USINT			

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
LREAL_TO_***	LREAL_TO_UINT	LREAL	UINT			
	LREAL_TO_UDINT	LREAL	UDINT			
	LREAL_TO_ULINT	LREAL	ULINT			
	LREAL_TO_LWORD	LREAL	LWORD			
	LREAL_TO_REAL	LREAL	REAL			
SINT_TO_***	SINT_TO_INT	SINT	INT			
	SINT_TO_DINT	SINT	DINT			
	SINT_TO_LINT	SINT	LINT			
	SINT_TO_USINT	SINT	USINT			
	SINT_TO_UINT	SINT	UINT			
	SINT_TO_UDINT	SINT	UDINT			
	SINT_TO_ULINT	SINT	ULINT			
	SINT_TO_BOOL	SINT	BOOL			
	SINT_TO_BYTE	SINT	BYTE			
	SINT_TO_WORD	SINT	WORD			
	SINT_TO_DWORD	SINT	DWORD			
	SINT_TO_LWORD	SINT	LWORD			
	SINT_TO_BCD	SINT	BYTE (BCD)			
	SINT_TO_REAL	SINT	REAL			
SINT_TO_LREAL	SINT	LREAL				
INT_TO_***	INT_TO_SINT	INT	SINT			
	INT_TO_DINT	INT	DINT			
	INT_TO_LINT	INT	LINT			
	INT_TO_USINT	INT	USINT			
	INT_TO_UINT	INT	UINT			
	INT_TO_UDINT	INT	UDINT			
	INT_TO_ULINT	INT	ULINT			
	INT_TO_BOOL	INT	BOOL			
	INT_TO_BYTE	INT	BYTE			
	INT_TO_WORD	INT	WORD			
	INT_TO_DWORD	INT	DWORD			
	INT_TO_LWORD	INT	LWORD			
	INT_TO_BCD	INT	WORD (BCD)			
	INT_TO_REAL	INT	REAL			
INT_TO_LREAL	INT	LREAL				

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
DINT_TO_***	DINT_TO_SINT	DINT	SINT			
	DINT_TO_INT	DINT	INT			
	DINT_TO_LINT	DINT	LINT			
	DINT_TO_USINT	DINT	USINT			
	DINT_TO_UINT	DINT	UINT			
	DINT_TO_UDINT	DINT	UDINT			
	DINT_TO_ULINT	DINT	ULINT			
	DINT_TO_BOOL	DINT	BOOL			
	DINT_TO_BYTE	DINT	BYTE			
	DINT_TO_WORD	DINT	WORD			
	DINT_TO_DWORD	DINT	DWORD			
	DINT_TO_LWORD	DINT	LWORD			
	DINT_TO_BCD	DINT	DWORD (BCD)			
	DINT_TO_REAL	DINT	REAL			
	DINT_TO_LREAL	DINT	LREAL			
LINT_TO_***	LINT_TO_SINT	LINT	SINT			
	LINT_TO_INT	LINT	INT			
	LINT_TO_DINT	LINT	DINT			
	LINT_TO_USINT	LINT	USINT			
	LINT_TO_UINT	LINT	UINT			
	LINT_TO_UDINT	LINT	UDINT			
	LINT_TO_ULINT	LINT	ULINT			
	LINT_TO_BOOL	LINT	BOOL			
	LINT_TO_BYTE	LINT	BYTE			
	LINT_TO_WORD	LINT	WORD			
	LINT_TO_DWORD	LINT	DWORD			
	LINT_TO_LWORD	LINT	LWORD			
	LINT_TO_BCD	LINT	LWORD (BCD)			
	LINT_TO_REAL	LINT	REAL			
	LINT_TO_LREAL	LINT	LREAL			
USINT_TO_***	USINT_TO_SINT	USINT	SINT			
	USINT_TO_INT	USINT	INT			
	USINT_TO_DINT	USINT	DINT			
	USINT_TO_LINT	USINT	LINT			
	USINT_TO_UINT	USINT	UINT			
	USINT_TO_UDINT	USINT	UDINT			
	USINT_TO_ULINT	USINT	ULINT			
	USINT_TO_BOOL	USINT	BOOL			
	USINT_TO_BYTE	USINT	BYTE			
	USINT_TO_WORD	USINT	WORD			
	USINT_TO_DWORD	USINT	DWORD			
	USINT_TO_LWORD	USINT	LWORD			

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
USINT_TO_***	USINT_TO_BCD	USINT	BYTE (BCD)			
	USINT_TO_REAL	USINT	REAL			
	USINT_TO_LREAL	USINT	LREAL			
UINT_TO_***	UINT_TO_SINT	UINT	SINT			
	UINT_TO_INT	UINT	INT			
	UINT_TO_DINT	UINT	DINT			
	UINT_TO_LINT	UINT	LINT			
	UINT_TO_USINT	UINT	USINT			
	UINT_TO_UDINT	UINT	UDINT			
	UINT_TO_ULINT	UINT	ULINT			
	UINT_TO_BOOL	UINT	BOOL			
	UINT_TO_BYTE	UINT	BYTE			
	UINT_TO_WORD	UINT	WORD			
	UINT_TO_DWORD	UINT	DWORD			
	UINT_TO_LWORD	UINT	LWORD			
	UINT_TO_BCD	UINT	WORD (BCD)			
	UINT_TO_REAL	UINT	REAL			
	UINT_TO_LREAL	UINT	LREAL			
	UINT_TO_DATE	UINT	DATE			
UDINT_TO_***	UDINT_TO_SINT	UDINT	SINT			
	UDINT_TO_INT	UDINT	INT			
	UDINT_TO_DINT	UDINT	DINT			
	UDINT_TO_LINT	UDINT	LINT			
	UDINT_TO_USINT	UDINT	USINT			
	UDINT_TO_UINT	UDINT	UINT			
	UDINT_TO_ULINT	UDINT	ULINT			
	UDINT_TO_BOOL	UDINT	BOOL			
	UDINT_TO_BYTE	UDINT	BYTE			
	UDINT_TO_WORD	UDINT	WORD			
	UDINT_TO_DWORD	UDINT	DWORD			
	UDINT_TO_LWORD	UDINT	LWORD			
	UDINT_TO_BCD	UDINT	DWORD (BCD)			
	UDINT_TO_REAL	UDINT	REAL			
	UDINT_TO_LREAL	UDINT	LREAL			
	UDINT_TO_TOD	UDINT	TOD			
UDINT_TO_TIME	UDINT	TIME				
ULINT_TO_***	ULINT_TO_SINT	ULINT	SINT			
	ULINT_TO_INT	ULINT	INT			
	ULINT_TO_DINT	ULINT	DINT			
	ULINT_TO_LINT	ULINT	LINT			
	ULINT_TO_USINT	ULINT	USINT			
ULINT_TO_UINT	ULINT	UINT				

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
ULINT_TO_***	ULINT_TO_UDINT	ULINT	UDINT			
	ULINT_TO_BOOL	ULINT	BOOL			
	ULINT_TO_BYTE	ULINT	BYTE			
	ULINT_TO_WORD	ULINT	WORD			
	ULINT_TO_DWORD	ULINT	DWORD			
	ULINT_TO_LWORD	ULINT	LWORD			
	ULINT_TO_BCD	ULINT	LWORD (BCD)			
	ULINT_TO_REAL	ULINT	REAL			
	ULINT_TO_LREAL	ULINT	LREAL			
BOOL_TO_***	BOOL_TO_SINT	BOOL	SINT			
	BOOL_TO_INT	BOOL	INT			
	BOOL_TO_DINT	BOOL	DINT			
	BOOL_TO_LINT	BOOL	LINT			
	BOOL_TO_USINT	BOOL	USINT			
	BOOL_TO_UINT	BOOL	UINT			
	BOOL_TO_UDINT	BOOL	UDINT			
	BOOL_TO_ULINT	BOOL	ULINT			
	BOOL_TO_BYTE	BOOL	BYTE			
	BOOL_TO_WORD	BOOL	WORD			
	BOOL_TO_DWORD	BOOL	DWORD			
	BOOL_TO_LWORD	BOOL	LWORD			
BOOL_TO_STRING	BOOL	STRING				
BYTE_TO_***	BYTE_TO_SINT	BYTE	SINT			
	BYTE_TO_INT	BYTE	INT			
	BYTE_TO_DINT	BYTE	DINT			
	BYTE_TO_LINT	BYTE	LINT			
	BYTE_TO_USINT	BYTE	USINT			
	BYTE_TO_UINT	BYTE	UINT			
	BYTE_TO_UDINT	BYTE	UDINT			
	BYTE_TO_ULINT	BYTE	ULINT			
	BYTE_TO_BOOL	BYTE	BOOL			
	BYTE_TO_WORD	BYTE	WORD			
	BYTE_TO_DWORD	BYTE	DWORD			
	BYTE_TO_LWORD	BYTE	LWORD			
	BYTE_TO_STRING	BYTE	STRING			
BYTE_TO_ASC	BYTE	WORD (ASCII)				
WORD_TO_***	WORD_TO_SINT	WORD	SINT			
	WORD_TO_INT	WORD	INT			
	WORD_TO_DINT	WORD	DINT			
	WORD_TO_LINT	WORD	LINT			
	WORD_TO_USINT	WORD	USINT			
	WORD_TO_UINT	WORD	UINT			

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
WORD_TO_***	WORD_TO_UDINT	WORD	UDINT			
	WORD_TO_ULINT	WORD	ULINT			
	WORD_TO_BOOL	WORD	BOOL			
	WORD_TO_BYTE	WORD	BYTE			
	WORD_TO_DWORD	WORD	DWORD			
	WORD_TO_LWORD	WORD	LWORD			
	WORD_TO_DATE	WORD	DATE			
	WORD_TO_STRING	WORD	STRING			
DWORD_TO_***	DWORD_TO_SINT	DWORD	SINT			
	DWORD_TO_INT	DWORD	INT			
	DWORD_TO_DINT	DWORD	DINT			
	DWORD_TO_LINT	DWORD	LINT			
	DWORD_TO_USINT	DWORD	USINT			
	DWORD_TO_UINT	DWORD	UINT			
	DWORD_TO_UDINT	DWORD	UDINT			
	DWORD_TO_ULINT	DWORD	ULINT			
	DWORD_TO_BOOL	DWORD	BOOL			
	DWORD_TO_BYTE	DWORD	BYTE			
	DWORD_TO_WORD	DWORD	WORD			
	DWORD_TO_LWORD	DWORD	LWORD			
	DWORD_TO_REAL	DWORD	REAL			
	DWORD_TO_TIME	DWORD	TIME			
	DWORD_TO_TOD	DWORD	TOD			
	DWORD_TO_STRING	DWORD	STRING			
LWORD_TO_***	LWORD_TO_SINT	LWORD	SINT			
	LWORD_TO_INT	LWORD	INT			
	LWORD_TO_DINT	LWORD	DINT			
	LWORD_TO_LINT	LWORD	LINT			
	LWORD_TO_USINT	LWORD	USINT			
	LWORD_TO_UINT	LWORD	UINT			
	LWORD_TO_UDINT	LWORD	UDINT			
	LWORD_TO_ULINT	LWORD	ULINT			
LWORD_TO_***	LWORD_TO_BOOL	LWORD	BOOL			
	LWORD_TO_BYTE	LWORD	BYTE			
	LWORD_TO_WORD	LWORD	WORD			
	LWORD_TO_DWORD	LWORD	DWORD			
	LWORD_TO_LREAL	LWORD	LREAL			
	LWORD_TO_DT	LWORD	DT			
	LWORD_TO_STRING	LWORD	STRING			
STRING_TO_***	STRING_TO_SINT	STRING	SINT			
	STRING_TO_INT	STRING	INT			
	STRING_TO_DINT	STRING	DINT			

## 7. Function and Function Block

Function group	Function	Input data type	Output data type	Application		
				GMR 2	GM3	GM4 7
STRING_TO_***	STRING_TO_LINT	STRING	LINT			
	STRING_TO_USINT	STRING	USINT			
	STRING_TO_UINT	STRING	UINT			
	STRING_TO_UDINT	STRING	UDINT			
	STRING_TO_ULINT	STRING	ULINT			
	STRING_TO_BOOL	STRING	BOOL			
	STRING_TO_BYTE	STRING	BYTE			
	STRING_TO_WORD	STRING	WORD			
	STRING_TO_DWORD	STRING	DWORD			
	STRING_TO_LWORD	STRING	LWORD			
	STRING_TO_REAL	STRING	REAL			
	STRING_TO_LREAL	STRING	LREAL			
	STRING_TO_DT	STRING	DT			
	STRING_TO_DATE	STRING	DATE			
	STRING_TO_TOD	STRING	TOD			
	STRING_TO_TIME	STRING	TIME			
NUM_TO_STRING	NUM_TO_STRING	ANY_NUM	STRING			
TIME_TO_***	TIME_TO_UDINT	TIME	UDINT			
	TIME_TO_DWORD	TIME	DWORD			
	TIME_TO_STRING	TIME	STRING			
DATE_TO_***	DATE_TO_UINT	DATE	UINT			
	DATE_TO_WORD	DATE	WORD			
	DATE_TO_STRING	DATE	STRING			
TOD_TO_***	TOD_TO_UDINT	TOD	UDINT			
	TOD_TO_DWORD	TOD	DWORD			
	TOD_TO_STRING	TOD	STRING			
DT_TO_***	DT_TO_LWORD	DT	LWORD			
	DT_TO_DATE	DT	DATE			
	DT_TO_TOD	DT	TOD			
	DT_TO_STRING	DT	STRING			

## 7. Function and Function Block

---

### 7.1.2. Arithmetic Function

#### 7.1.2.1. Numerical Operation Function with One Input

It supports GMR, GM1, GM2 (Note: ABS function supports GM3, GM4, GM6, GM7).

No.	Function	Description
General function		
1	ABS	Absolute value operation
2	SQRT	Calculate SQRT (Square root operation)
Logarithm		
3	LN	Natural logarithm operation
4	LOG	Base 10 logarithm operation
5	EXP	Natural exponential operation
Trigonometric function		
6	SIN	Sine operation
7	COS	Cosine operation
8	TAN	Tangent operation
9	ASIN	Arc Sine operation
10	ACOS	Arc Cosine operation
11	ATAN	Arc Tangent operation
Angle function		
12	RAD_REAL	Convert degree into radian
13	RAD_LREAL	
14	DEG_REAL	Convert radian into degree
15	DEG_LREAL	

#### 7.1.2.2. Basic Arithmetic Function

EXPT supports GMR, GM1, GM2 only; XCHG\_\*\*\* supports GM3, GM4, GM6, GM7.

No.	Function	Description
Operation function of which input number (n) can be extended up to 8.		
1	ADD	Addition ( $OUT \leq IN1 + IN2 + \dots + INn$ )
2	MUL	Multiplication ( $OUT \leq IN1 * IN2 * \dots * INn$ )
Operation function of which input number is fixed.		
3	SUB	Subtraction ( $OUT \leq IN1 - IN2$ )
4	DIV	Division ( $OUT \leq IN1 / IN2$ )
5	MOD	Calculate remainder ( $OUT \leq IN1 \text{ Modulo } IN2$ )
6	EXPT	Exponential operation ( $OUT \leq IN1^{IN2}$ )
7	MOVE	Copy data ( $OUT \leq IN$ )
Input data exchange		
8	XCHG_***	Exchanges two input data



### 7.1.3. Bit Array Function

#### 7.1.3.1. Bit-shift Function

No.	Function	Description
1	SHL	Shift left
2	SHR	Shift right
3	SHIFT_C_***	Shift with Carry
4	ROL	Rotate left
5	ROR	Rotate right
6	ROTATE_C_***	Rotates a designated direction

#### 7.1.3.2. Bit Operation Function

No.	Function	Description (n can be extended up to 8)
1	AND	Logical AND (OUT <= IN1 AND IN2 AND ... AND INn)
2	OR	Logical OR (OUT <= IN1 OR IN2 OR ... OR INn)
3	XOR	Exclusive OR (OUT <= IN1 XOR IN2 XOR ... XOR INn)
4	NOT	Reverse logic (OUT <= NOT IN1)

#### 7.1.4. Selection Function

No.	Function	Description (n can be extended up to 8)
1	SEL	Selection from two inputs
2	MAX	Produces a maximum value among input IN1, ..., INn
3	MIN	Produces a minimum value among input IN1, ..., INn
4	LIMIT	Limits upper and lower boundary
5	MUX	Selection from multiple inputs

#### 7.1.5. Data Exchange Function

No.	Function	Description
1	SWAP_BYTE	Swaps upper nibble for lower nibble data.
	SWAP_WORD	Swaps upper byte for lower byte data.
	SWAP_DWORD	Swaps upper word for lower word data.
	SWAP_LWORD	Swaps upper double word for lower double word data.
2	ARY_SWAP_BYTE	Swaps upper/lower nibble of byte elements.
	ARY_SWAP_WORD	Swaps upper/lower byte of WORD elements.
	ARY_SWAP_DWORD	Swaps upper/lower WORD of DWORD elements.
	ARY_SWAP_LWORD	Swaps upper/lower DWORD of LWORD elements.

## 7. Function and Function Block

---

### 7.1.6. Comparison Function

No.	Function	Description (n can be extended up to 8)
1	GT	'Greater than' comparison OUT <= (IN1>IN2) & (IN2>IN3) & ... & (INn-1 > INn)
2	GE	'Greater than or equal to' comparison OUT <= (IN1>=IN2) & (IN2>=IN3) & ... & (INn-1 >= INn)
3	EQ	'Equal to' comparison OUT <= (IN1=IN2) & (IN2=IN3) & ... & (INn-1 = INn)
4	LE	'Less than or equal to' comparison OUT <= (IN1<=IN2) & (IN2<=IN3) & ... & (INn-1 <= INn)
5	LT	'Less than' comparison OUT <= (IN1<IN2) & (IN2<IN3) & ... & (INn-1 < INn)
6	NE	'Not equal to' comparison OUT <= (IN1<>IN2) & (IN2<>IN3) & ... & (INn-1 <> INn)

### 7.1.7. Character String Function

No.	Function	Description
1	LEN	Find a length of a character string
2	LEFT	Take a left side of a string
3	RIGHT	Take a right side of a string
4	MID	Take a middle side of a string
5	CONCAT	Concatenate the input character string in order
6	INSERT	Insert a string
7	DELETE	Delete a string
8	REPLACE	Replace a string
9	FIND	Find a string

**7.1.8. Time/Time of Day/Date and Time of Day Function**

No.	Function	Description
1	ADD_TIME	Add time (Time/time of day/date and time addition)
2	SUB_TIME	Subtract time
	SUB_DATE	Subtract date
	SUB_TOD	Subtract TOD
	SUB_DT	Subtract DT
3	MUL_TIME	Multiply time
4	DIV_TIME	Divide time
5	CONCAT_TIME	Concatenate date with TOD

**7.1.9. System Control Function**

No.	Function	Description
1	DI	Invalidates interrupt (Not to permit task program starting)
2	EI	Permits running for a task program
3	STOP	Stop running by a task program
4	ESTOP	Emergency running stop by a program
5	DIREC_IN	Update input data (available for GM1 GM7)
6	DIREC_O	Updates output data (available in GM1 GM7)
7	WDT_RST	Initialize a timer of watchdog
8	MCS	Set MCS (Master Control)
9	MCCLR	Set MCCLR (Master Control Clear)

## 7. Function and Function Block

---

### 7.1.10. Data Manipulation Function

No.	Function	Description
1	MEQ_***	Compare whether two inputs are equal after masking
2	DIS_***	Data distribution
3	UNI_***	Unite data
4	BIT_BYTE	Combine 8 bits into one byte
5	BYTE_BIT	Divide one byte into 8 bits
6	BYTE_WORD	Combine two bytes into one WORD
7	WORD_BYTE	Divide one WORD into two bytes
8	WORD_DWORD	Combine two WORD data into DWORD
9	DWORD_WORD	Divide DWORD into 2 WORD data
10	DWORD_LWORD	Combine two DWORD data into LWORD
11	LWORD_DWORD	Divide LWORD into two DWORD data
12	GET_CHAR	Get one character from a character string
13	PUT_CHAR	Puts a character in a string
14	STRING_TO_ARY	Convert a string into a byte array
15	ARY_TO_STRING	Convert a byte array into a string

### 7.1.11. Stack Operation Function

No.	Function	Description
1	FIFO_***	First In First Out
2	LIFO_***	Last In First Out

## 7.2. MK (MASTER-K) Function

No.	Function	Description (n can be extended up to 8)
1	ENCO_***	Output a position of On bit by number
2	DECO_***	Turn a selected bit on
3	BSUM_***	Output a number of On bit
4	SEG	Convert BCD/HEX into 7-segment code
5	BMOV_***	Move part of a bit string
6	INC_***	Increase IN data
7	DEC_***	Decrease IN data

## 7.3. Array Operation Function

No.	Function	Description
1	ARY_MOVE	Copy array-typed data (OUT <= IN)
2	ARY_CMP_***	Array comparison
3	ARY_SCH_***	Array search
4	ARY_FLL_***	Filling an array with data
5	ARY_AVE_***	Find an average of an array
6	ARY_SFT_C_***	Array bit shift left with carry
7	ARY_ROT_C_***	Bit rotation of array with carry
8	SHIFT_A_***	Shift array elements
9	ROTATE_A_***	Rotates array elements

## 7.4. Basic Function Block

### 7.4.1. Bistable Function Block

No.	Function Block	Description
1	SR	Set preference bistable
2	RS	Reset preference bistable
3	SEMA	Semaphore

### 7.4.2. Edge Detection Function Block

No.	Function Block	Description
1	R_TRIG	Rising edge detector
2	F_TRIG	Falling edge detector

## **7. Function and Function Block**

---

### **7.4.3. Counter**

No.	Function Block	Description
1	CTU	Up Counter
2	CTD	Down Counter
3	CTUD	Up/Down Counter
4	CTR	Ring Counter

### **7.4.4. Timer**

No.	Function Block	Description
1	TP	Pulse Timer
2	TON	On-Delay Timer
3	TOF	Off-Delay Timer
4	TMR	Integrating Timer
5	TP_RST	TP with reset
6	TRTG	Retriggerable Timer
7	TOF_RST	TOF with reset
8	TON_UNIT	TON with integer setting
9	TOF_UNIT	TOF with integer setting
10	TP_UNIT	TP with integer setting
11	TMR_UNIT	TMR with integer setting

### **7.4.5. Other Function Block**

No.	Function Block	Description
1	SCON	Step Controller
2	DUTY	Scan setting On/Off

## 8. Function/Function Block Library

### 8.1 Basic Function Library

This chapter describes the basic function library respectively.

**POINT** When a function error occurs, please refer to the following instruction.

Function error

If an error occurs when a function is run, ENO will be 0 and, the error flag ( \_ERR, \_LER) will be 1.

Unless an error occurs, ENO will be equal to EN (EN and ENO are used in LD only).

Error flag

\_ERR (Error)

- After function execution of which error is described, \_ERR value will be changed as follows: (There's no change in \_ERR value as long as there's no function error.)

- In case of an operation error, it will be 1.

- In other cases, it will be 0.

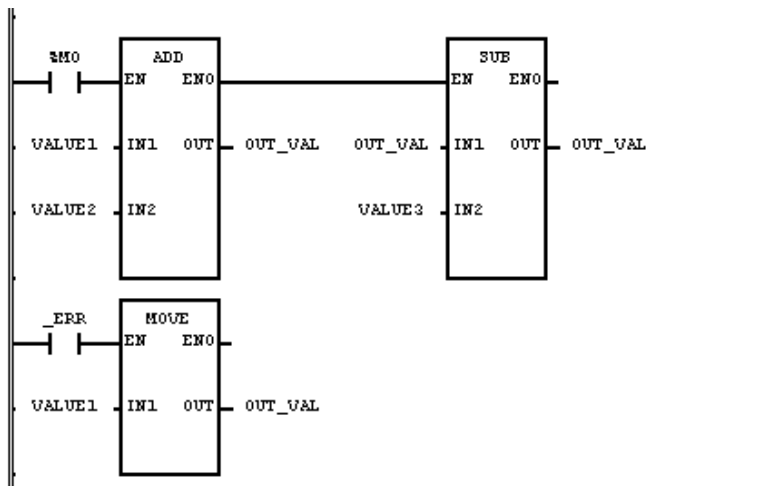
\_LER (Latched Error)

- In case of an error after execution, \_LER will be 1 and maintained until the end of the program.

- It is possible to write 0 in the program.

#### ■ Program Example

This is a program that moves VALUE1 data to OUT\_VAL without executing SUB function if an ADD function error occurs.



(1) An error occurs in ADD function when its two inputs are as follows:

Input (IN1): VALUE1 (SINT) = 100 (16#64)

Input (IN2): VALUE2 (SINT) = 50 (16#32)

Output (OUT): OUT\_VAL (SINT) = -106 (16#96)

(2) As an output value is out of range of its data type, the abnormal value will be stored in the OUT\_VAL (SINT). At this time, ENO of ADD function will be 0 and SUB function will not be executed, and the error flag ( \_ERR and \_LER) will be on.

(3) \_ERR will be on and MOVE function will be executed.

Input (IN1): VALUE1 (SINT) = 100 (16#64)

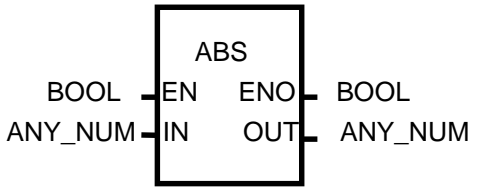
Output (OUT): OUT\_VAL (SINT) = 100 (16#64)

## 8. Basic Function/Function Block Library

### ABS

Absolute value operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of absolute value operation</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: absolute value IN, OUT should be the same data type.</p>

#### ■ Function

It converts input IN into its absolute value and produces output OUT.

$|X|$ , an absolute value of X is,

If  $X \geq 0$ ,  $|X| = X$ ,

If  $X < 0$ ,  $|X| = -X$ .

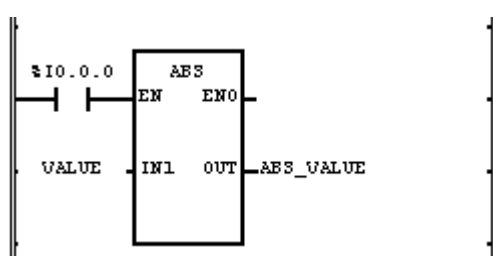
$OUT = |IN|$

#### ■ Error

\_ERR, \_LER flags are set when input IN is a minimum value.

Ex) If IN value is  $-128$  and its data type is SINT, an error occurs.

#### ■ Program Example

LD	IL
	<pre>LD      %I0.0.0 JMPN   AL LD      VALUE ABS ST      ABS_VALUE AL :</pre>

(1) If the transition condition (%I0.0.0) is on, ABS function will be executed.

(2) If  $VALUE = -7$ ,  $ABS\_VALUE = |-7| = 7$ .

If  $VALUE = 200$ ,  $ABS\_VALUE = |200| = 200$ .

Input (IN): VALUE (INT) = -7

1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
															(16#FFF9)	

(ABS)



Output (OUT): ABS\_VALUE (INT) = 7

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
															(16#0007)	

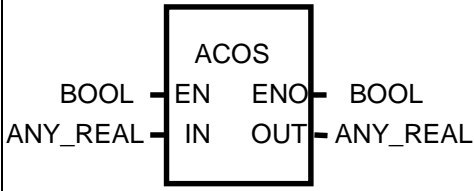
The negative number of INT type is represented as the 2's complement form (refer to 3.2.4. Data Type Structure)



## ACOS

Arc Cosine operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of Arc Cosine operation</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: Arc Cosine (radian)</p> <p>IN, OUT should be the same data type.</p>

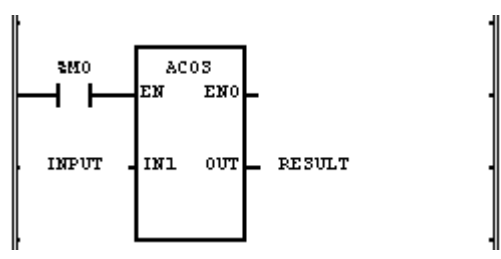
■ **Function**

It converts input IN into its Arc Cosine value and produces output OUT. The output range is between 0 and  $\pi$ .  
 $OUT = ACOS (IN)$ .

■ **Error**

Unless an IN value is between -1.0 and 1.0, \_ERR, \_LER flags are set.

■ **Program Example**

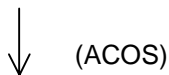
LD	IL
	<pre>LD      %M0 JMPN   LL LD      INPUT ACOS ST      RESULT LL :</pre>

- (1) If the transition condition (%M0) is on, ACOS function will be executed.
- (2) If INPUT is 0.8660... ( $\sqrt{3} / 2$ ), RESULT will be 0.5235... ( $\pi/6$  rad =  $30^\circ$ ).

$$ACOS (\sqrt{3} / 2) = \pi/6$$

$$(COS \pi/6 = \sqrt{3} / 2)$$

Input (IN1): INPUT (REAL) = 0.866



Output (OUT): RESULT (REAL) = 5.23499966E-01

REAL type representation is based on IEEE Standard 754-1984 (refer to 3.2.4. Data Type Structure).

## 8. Basic Function/Function Block Library

### ADD

Addition
----------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: value to be added            IN2: value to add            Input variable number can be extended up to 8</p> <p><b>Output</b> ENO: without an error, it will be 1            OUT: added value</p> <p>IN1, IN2, ..., OUT should be the same data type.</p>

#### ■ Function

It adds input variables up (IN1, IN2, ..., and INn, n: input number) and produces output OUT.  
 $OUT = IN1 + IN2 + \dots + INn$

#### ■ Error

When the output value is out of its data type, \_ERR, \_LER flags are set.

#### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   CA LD      VALUE1 ADD     IN1:= CURRENT RESULT         IN2:= VALUE2         IN3:= VALUE3 ST      OUT_VAL CA :           </pre>

(1) If the transition condition (%M0) is on, ADD function will be executed.

(2) If input variable VALUE1 = 300, VALUE2 = 200, and VALUE3 = 100,  
 output variable OUT\_VAL = 300 + 200 + 100 = 600.

Input (IN1): VALUE1 (INT) = 300 (16#012C)	0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0
	+ (ADD)
(IN2): VALUE2 (INT) = 200 (16#00C8)	0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
	+ (ADD)
(IN2): VALUE3 (INT) = 100 (16#0064)	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
	↓
(OUT): OUT_VAL (INT) = 600 (16#0258)	0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0

## ADD\_TIME

Time Addition
---------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: reference time, time of date IN2: time to add</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: added result of TOD or time</p> <p>IN1, IN2, and OUT should be the same data type: If IN1 type is TIME_OF_DAY, OUT type will be also TIME_OF_DAY.</p>

### Function

If IN1 is TIME, added TIME will be an output.

If IN1 is TIME\_OF\_DAY, it adds TIME to reference TIME\_OF\_DAY and produces output TIME\_OF\_DAY.

If IN1 is DATE\_AND\_TIME, the output data type will be DT (Date and Time of Day) adding the time to the standard date and time of day.

### Error

If an output value is out of range of related data type, \_ERR, \_LER flag will be set.

An error occurs: 1) when the result of adding the time and the time is out of range of TIME data type T#49D17H2M47S295MS; 2) the result of adding TOD (Time of Day) and the time exceeds 24hrs; 3) the result of adding the date and DT (Date and the Time of Day) exceeds the year, 2083.

### Program Example

LD	IL
	<pre>LD      %I0.1.0 JMPN   ABC LD      START_TIME ADD_TIME IN1:= CURRENT RESULT         IN2:= WORK_TIME  ST      END_TIME ABC :</pre>

(1) If the transition condition (%I0.1.0) is on, ADD\_TIME function will be executed.

(2) If START\_TIME is TOD#08:30:00 and WORK\_TIME is T#2H10M20S500MS, END\_TIME will be TOD#10:40:20.5.

Input (IN1): START\_TIME (TOD) = TOD#08:30:00  
+ (ADD\_TIME)

(IN2): WORK\_TIME (TIME) = T#2H10M20S500MS

Output (OUT): END\_TIME (TOD) = TOD#10:40:20.5

## 8. Basic Function/Function Block Library

### AND

Logical AND (Logical multiplication)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
<p>The diagram shows a rectangular function block labeled 'AND'. On the left side, there are three input terminals: 'EN' (type BOOL), 'IN1' (type ANY_BIT), and 'IN2' (type ANY_BIT). On the right side, there are two output terminals: 'ENO' (type BOOL) and 'OUT' (type ANY_BIT).</p>	<p><b>Input</b> EN: executes the function in case of 1            IN1: input 1            IN2: input 2            Input variables can be extended up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1            OUT: AND result            IN1, IN2, and OUT should be all the same data type.</p>

#### ■ Function

It performs logical AND operation on the input variables by bit and produces output OUT.

```

IN1  1111 ..... 0000
      &
IN2  1010 ..... 1010
OUT  1010 ..... 0000
  
```

#### ■ Program Example

LD	IL
<p>The ladder logic diagram shows a normally open contact labeled '%I0.1.1' connected to the 'EN' input of an 'AND' function block. The 'IN1' input is connected to a coil labeled '%MB10'. The 'IN2' input is connected to a coil labeled 'ABC'. The 'OUT' output of the function block is connected to a coil labeled '%QB0.0.0'.</p>	<pre> LD      %I0.1.1 JMPN   AA LD      %MB10 AND    IN1:= CURRENT RESULT        IN2:= ABC ST      %QB0.0.0 AA :   </pre>

- (1) If the transition condition (%I0.1.1) is on, AND function will be executed.
- (2) If IN1 = %MB10 and IN2 = ABC, the result of AND will be shown in OUT (%QB0.0.0).

Input (IN1): %MB10 (BYTE) = 16#CC

1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

& (AND)

(IN2): ABC (BYTE) = 16#F0

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

↓

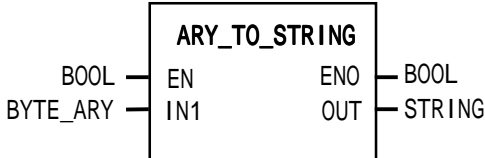
Output (OUT): %QB0.0.0 (BYTE) = 16#C0

1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

## ARY\_TO\_STRING

Converts a byte array into a string

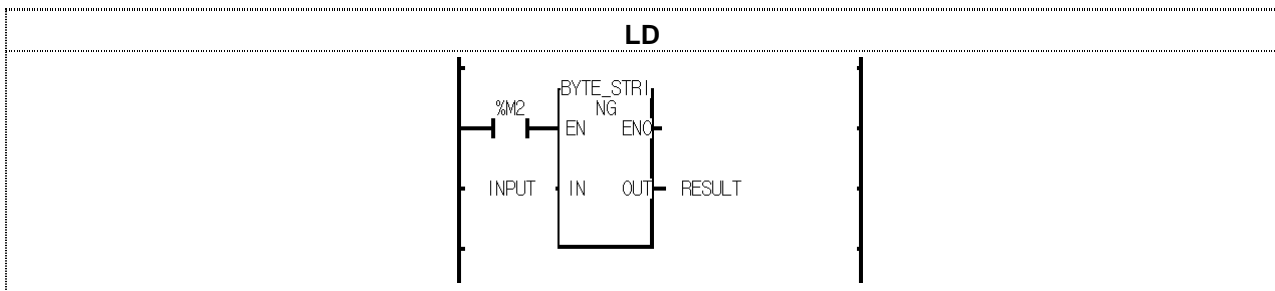
MODEL	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: byte array input</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: string output</p>

### ■ Function

It converts a byte array input into a string.

### ■ Program Example



(1) If the transition condition (%M2) is on, BYTE\_STRING function will be executed.

(2) Input variable INPUT is converted into string-type variable OUTPUT.

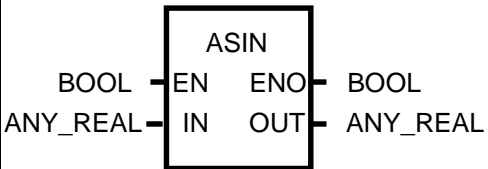
For example, if INPUT is 16#{22("), 47(G), 4D(M), 34(4), 2D(-), 43(C), 50(P), 55(U), 41(A), 22(")}, the RESULT will be "GM4-CPUA".

## 8. Basic Function/Function Block Library

### ASIN

Arc Sine operation
--------------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of Arc Sine operation</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: radian output value after operation IN and OUT should be the same data type.</p>

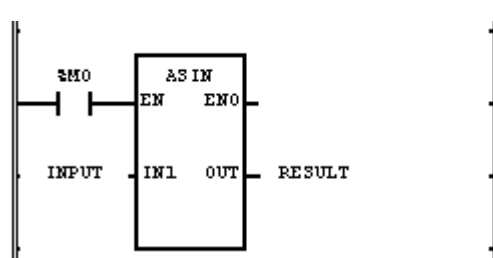
#### ■ Function

It produces an output (Arc Sine value) of IN. The output value is between  $-\pi/2$  and  $\pi/2$ .  
 $OUT = ASIN(IN)$

#### ■ Error

If an input value exceeds the range from -1.0 to 1.0, `_ERR` and `_LER` flags are set.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      INPUT ASIN ST      RESULT AAA :</pre>

(1) If the transition condition (%M0) is on, ASIN function will be executed.

(2) If INPUT variable is 0.8660.... ( $\sqrt{3}/2$ ), the RESULT will be 1.0471.... ( $\pi/3$  radian =  $60^\circ$ ).

$$ASIN(\sqrt{3}/2) = \pi/3$$

Therefore,  $SIN(\pi/3) = \sqrt{3}/2$

Input (IN1): INPUT (REAL) = 0.866

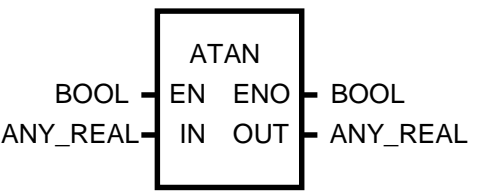
↓ (ASIN)

Output (OUT): RESULT (REAL) = 1.04714680E+00

## ATAN

Arc Tangent operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

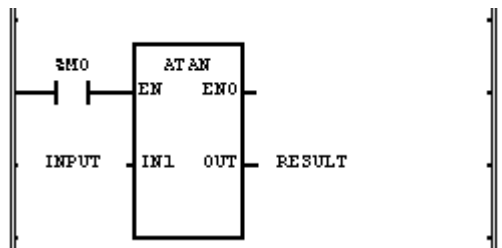
Function	Description
 <p>The diagram shows a rectangular function block labeled 'ATAN'. On the left side, there are two input terminals: 'EN' with data type 'BOOL' and 'IN' with data type 'ANY_REAL'. On the right side, there are two output terminals: 'ENO' with data type 'BOOL' and 'OUT' with data type 'ANY_REAL'.</p>	<p><b>Input</b> EN: executes the function in case of 1 IN: Input value of Arc Tangent operation</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: radian output value after operation IN, OUT should be the same data type.</p>

### ■ Function

It produces an output (Arc Tangent value) of IN value. The output value is between  $-\pi/2$  and  $\pi/2$ .

OUT = ATAN (IN)

### ■ Program Example

LD	IL
 <p>The diagram shows a single ladder logic rung. It starts with a normally open contact labeled '%M0'. This is followed by a function block labeled 'ATAN'. The 'EN' input of the block is connected to the '%M0' contact. The 'IN' input is labeled 'INPUT'. The 'ENO' output is connected to the right rail. The 'OUT' output is labeled 'RESULT'.</p>	<pre>LD      %M0 JMPN   AA LD      INPUT ATAN ST      RESULT AA :</pre>

(1) If the transition condition (%M0) is on, ATAN function will be executed.

(2) If INPUT = 1.0, then output RESULT will be:

RESULT =  $\pi/4 = 0.7853\dots$

ATAN (1) =  $\pi/4$

(TAN ( $\pi/4$ ) = 1)

Input (IN1): INPUT (REAL) = 1.0

↓ (ATAN)

Output (OUT): RESULT (REAL) = 7.85398185E-01

## 8. Basic Function/Function Block Library

### BCD\_TO\_\*\*\*

Converts BCD data into an integer number

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: ANY_BIT (BCD)</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

#### ■ Function

It converts input IN type and produces output OUT.

Function	Input type	Output type	Description
BCD_TO_SINT	BYTE	SINT	It converts BCD data into an output data type. It converts only when the input data type is a BCD value. If an input data type is WORD, only the part of its data (0 16#9999) will be normally converted.
BCD_TO_INT	WORD	INT	
BCD_TO_DINT	DWORD	DINT	
BCD_TO_LINT	LWORD	LINT	
BCD_TO_USINT	BYTE	USINT	
BCD_TO_UINT	WORD	UINT	
BCD_TO_UDINT	DWORD	UDINT	
BCD_TO_ULINT	LWORD	ULINT	

#### ■ Error

If IN is not a BCD data type, then the output will be 0 and \_ERR, \_LER flags will be set.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   ABC LD      BCD_VAL BCD_TO_SINT ST      OUT_VAL ABC :</pre>

(1) If the transition condition (%M0) is on, BCD\_TO\_\*\*\* function will be executed.

(2) If BCD\_VAL (BYTE) = 16#22 (2#0010\_0010),

then the output variable OUT\_VAL (SINT) = 22 (2#0001\_0110).

Input (IN1): BCD\_VAL (BYTE) = 16#22

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

√(BCD\_TO\_SINT)

Output (OUT): OUT\_VAL (SINT) = 22

0	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---



**BOOL\_TO\_\*\*\***

BOOL type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: bit to convert (1 bit)</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

■ **Function**

It converts input IN type and produces output OUT.

Function	Output type	Description
BOOL_TO_SINT	SINT	If the input value (BOOL) is 2#0, it produces the integer number '0' and if it is 2#1, it does the integer number '1' according to the output data type.
BOOL_TO_INT	INT	
BOOL_TO_DINT	DINT	
BOOL_TO_LINT	LINT	
BOOL_TO_USINT	USINT	
BOOL_TO_UINT	UINT	
BOOL_TO_UDINT	UDINT	
BOOL_TO_ULINT	ULINT	
BOOL_TO_BYTE	BYTE	It converts BOOL into the output data type of which upper bits are filled with 0.
BOOL_TO_WORD	WORD	
BOOL_TO_DWORD	DWORD	
BOOL_TO_LWORD	LWORD	
BOOL_TO_STRING	STRING	It converts BOOL into a STRING type, which will be '0' or '1'.

■ **Program Example**

LD	IL
	<pre> LD      %M0 JMPN   ABC LD      BOOL_VAL BOOL_TO_BYTE ST      OUT_VAL ABC :         </pre>

(1) If the transition condition (%M0) is on, BOOL\_TO\_\*\*\* function will be executed.

(2) If input BOOL\_VAL (BOOL) = 2#1, then output OUT\_VAL (BYTE) = 2#0000\_0001.

Input (IN1): BOOL\_VAL (BOOL) = 2#1

1



(BOOL\_TO\_SINT)

Output (OUT): OUT\_VAL (BYTE) = 16#1

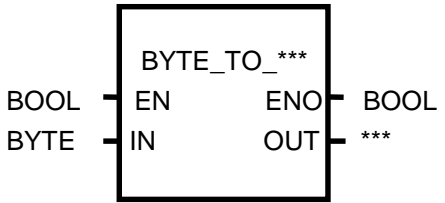
0 0 0 0 0 0 0 1

## 8. Basic Function/Function Block Library

### BYTE\_TO\_\*\*\*

BYTE type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

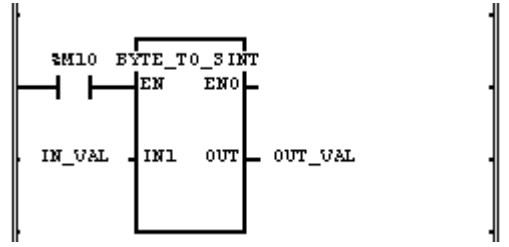
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: bit string to convert (8 bits)</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

It converts input IN type and produces output OUT.

Function	Output type	Description
BYTE_TO_SINT	SINT	Converts into SINT type without changing its internal bit array.
BYTE_TO_INT	INT	Converts into INT type filling the upper bits with 0.
BYTE_TO_DINT	DINT	Converts into DINT type filling the upper bits with 0.
BYTE_TO_LINT	LINT	Converts into LINT type filling the upper bits with 0.
BYTE_TO_USINT	USINT	Converts into USINT type without changing its internal bit array.
BYTE_TO_UINT	UINT	Converts into UINT type filling the upper bits with 0.
BYTE_TO_UDINT	UDINT	Converts into UDINT type filling the upper bits with 0.
BYTE_TO_ULINT	ULINT	Converts into ULINT type filling the upper bits with 0.
BYTE_TO_BOOL	BOOL	Takes the lower 1 bit and converts it into BOOL type.
BYTE_TO_WORD	WORD	Converts into WORD type filling the upper bits with 0.
BYTE_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
BYTE_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
BYTE_TO_STRING	STRING	Converts the input value into STRING type.

#### ■ Program Example

LD	IL
	<pre> LD      %M10 JMPN   LLL LD      IN_VAL BYTE_TO_SINT ST      OUT_VAL LLL : </pre>

(1) If the transition condition (%M10) is on, BYTE\_TO\_SINT function will be executed.

(2) If IN\_VAL (BYTE) = 2#0001\_1000, OUT\_VAL (SINT) = 24 (2#0001\_1000).

Input (IN1): IN\_VAL (BYTE) = 16#18

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

↓ (BYTE\_TO\_SINT)

Output (OUT): OUT\_VAL (SINT) = 24

0	0	0	1	1	0	0	0
---	---	---	---	---	---	---	---

## CONCAT

Concatenates a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: input character string            IN2: input character string            Input variable number can be extended up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: output character string</p>

### ■ Function

It concatenates the input character string IN1, IN2, IN3, ..., INn (n: input number) in order and produces output character string OUT.

### ■ Error

If the sum of character number of each input character string is greater than 30, then the output CONCAT is the concatenate string of each input character string (up to 30 letters), and \_ERR, \_LER flags will be set.

### ■ Program Example

LD	IL
	<pre> LD          %I0.2.1 JMPN      THERE LD          IN_TEXT1 CONCAT    IN1:= CURRENT RESULT           IN2:= IN_TEXT2 ST          OUT_TEXT THERE :           </pre>

(1) If the transition condition (%I0.2.1) is on, CONCAT function will be executed.

(2) If input variable IN\_TEXT1 = 'ABCD' and IN\_TEXT2 = 'DEF', then OUT\_TEXT = 'ABCDDEF'.

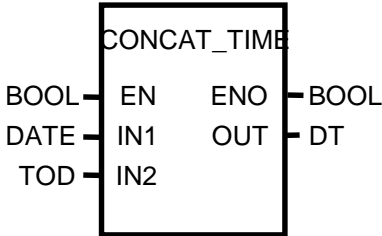
Input (IN1): IN\_TEXT1 (STRING) = 'ABCD'  
 (IN2): IN\_TEXT2 (STRING) = 'DEF'  
 ↓ (CONCAT)  
 Output (OUT): OUT\_TEXT (STRING) = 'ABCDDEF'

## 8. Basic Function/Function Block Library

### CONCAT\_TIME

Concatenates date and time of day

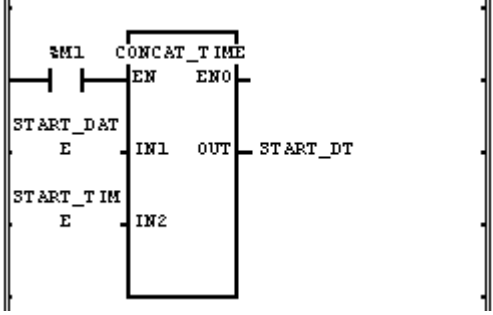
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: date data input IN2: Time of day data input</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: DT (Date and Time of Day) output</p>

#### ■ Function

It concatenates IN1 (date) and IN2 (time of day) and produces output OUT (DT).

#### ■ Program Example

LD	IL
	<pre> LD      %M1 JMPN   AA LD      START_DATE CONCAT_TIME  IN1:= CURRENT RESULT            IN2:= START_TIME ST      START_DT AA : </pre>

(1) If the transition condition (%M1) is on, CONCAT\_TIME function will be executed.

(2) If START\_DATE = D#1995-12-06 and START\_TIME = TOD#08:30:00,  
then, output START\_DT = DT#1995-12-06-08:30:00.

Input (IN1): START\_DATE1 (DATE) = D#1995-12-06  
(CONCAT\_TIME)  
(IN2): START\_TIME (TOD) = TOD#08:30:00



Output (OUT): START\_DT (DT) = DT#1995-12-06-08:30:00



## 8. Basic Function/Function Block Library

### DATE\_TO\_\*\*\*

Date type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: date data to convert</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

It converts an input IN type and produces output OUT.

Function	Output type	Description
DATE_TO_UINT	UINT	Converts DATE into UINT type.
DATE_TO_WORD	WORD	Converts DATE into WORD type.
DATE_TO_STRING	STRING	Converts DATE into STRING type.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   LL LD      IN_VAL DATE_TO_STRING ST      OUT_VAL LL :</pre>

(1) If the transition condition (%M0) is on, DATE\_TO\_STRING function will be executed.

(2) If IN\_VAL (DATE) = D#1995-12-01, OUT\_VAL (STRING) = D#1995-12-01.

Input (IN1): IN\_VAL (DATE) = D#1995-12-01

↓ (DATE\_TO\_STRING)

Output (OUT): OUT\_VAL (STRING) = 'D#1995-12-01'

## DELETE

Deletes a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <ul style="list-style-type: none"> <li>EN: executes the function in case of 1</li> <li>IN: input character string</li> <li>L: length of character string to delete</li> <li>P: position of character string to delete</li> </ul> <p><b>Output</b></p> <ul style="list-style-type: none"> <li>ENO: without an error, it will be 1.</li> <li>OUT: output character string</li> </ul>

### ■ Function

After deleting a character string (L) from the P character of IN, produces output OUT.

### ■ Error

If  $P \leq 0$  or  $L < 0$ , or

If  $P >$  character number of IN, \_ERR and \_LER flags will be set.

### ■ Program Example

LD	IL
	<pre> LD      %I0.0.0 JMPN   KKK LD      IN_TEXT DELETE IN:= CURRENT RESULT         L:= LENGTH         P:= POSITION ST      OUT_TEXT KKK : </pre>

(1) If the transition condition (%I0.0.0) is ON, DELETE function will be executed.

(2) If input variable IN\_TEXT = 'ABCDEF', LENGTH = 3, and POSITION = 3, then OUT\_TEXT (STRING) will be 'ABF'.

Input (IN): IN\_TEXT (STRING) = 'ABCDEF'

(L): LENGTH (INT) = 3

(P): POSITION (INT) = 3

↓ (DELETE)

Output (OUT): OUT\_VAL (STRING) = 'ABF'

## 8. Basic Function/Function Block Library

### DI

Invalidates task program (Not to permit task program starting)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 REQ: requires to invalidate task program starting</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: If DI is executed, it will be 1.</p>

#### ■ Function

If EN = 1 and REQ = 1, it stops a task program (single, interval, interrupt).

Once DI function is executed, a task program does not start even if REQ input is 0.

In order to start a task program normally, please use 'EI' function.

If you want to partially stop the task program for the troubled part, (otherwise, miss the continuity of operation process due to the execution of other task program), it is available to use this function.

The task programs created while its execution is not invalidated will be executed according to task program types as follows:

- Single task: it will be executed after 'EI' function or current-running task program execution. In his case, it repeats a task program as many as the state of single variable changes.
- Interval task, interrupt: Interval task, interrupt: the task occurred when it is not permitted to execute will be executed after 'EI' function or the current-running task program execution. But, if it occurs more than 2 times, TASK\_ERR is ON and TC\_CNT (the number of task collision) is counted.



■ Program Example

This is the program that controls the task program increasing the value per second by using DI (Invalidates task program) and EI (permits running for task program).

LD	IL
<p>(1) Scan program (TASK program control)</p>	<p>(1) Scan program (TASK program control)</p> <pre>LDN      %M100 JMPN    KK LD      %I0.1.14 DI ST      DI_OK KK :</pre> <pre>LDN      %M100 JMPN    LL LD      %I0.1.15 EI ST      EI_OK LL :</pre>
<p>(2) Task program increasing by executing per second.</p>	<p>(2) Task program increasing by executing per second</p> <pre>LDN      %M1 JMPN    MM LD      %IW0.0.0 MOVE ST      %MW100 MM :</pre>

- (1) If REQ (assigned as direct variable %I0.1.14) of DI is on, DI function will be executed and output DI\_OK will be 1.
- (2) If DI function is executed, the task program to be executed per second stops.
- (3) If REQ (assigned as direct variable %I0.1.15) of EI is on, EI function will be executed and output EI\_OK will be 1.
- (4) If EI function is executed, the task program stopped due to function DI will restart.

## 8. Basic Function/Function Block Library

### DINT\_TO\_\*\*\*

Invalidates task program (Not to permit task program starting)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: double integer value to convert</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

It converts Input IN type and produces output OUT.

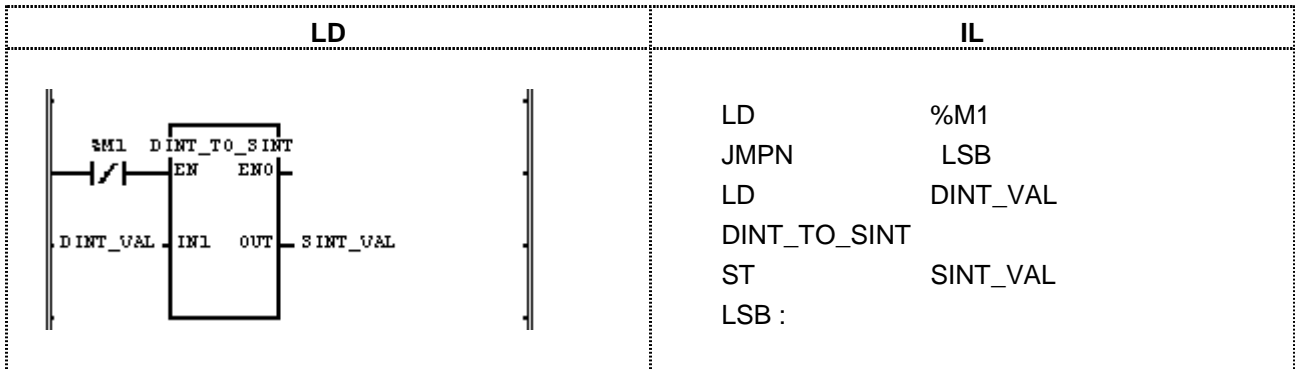
Function	Output type	Description
DINT_TO_SINT	SINT	If input is -128 127, normal conversion. Except this, an error occurs.
DINT_TO_INT	INT	If input is -32768 32767, normal conversion. Except this, an error occurs.
DINT_TO_LINT	LINT	Converts normally into LINT type.
DINT_TO_USINT	USINT	If input is 0 255, normal conversion. Except this, an error occurs.
DINT_TO_UINT	UINT	If input is 0 65535, normal conversion. Except this, an error occurs.
DINT_TO_UDINT	UDINT	If input is 0 2147483647, normal conversion. Except this, an error occurs.
DINT_TO_ULINT	ULINT	If input is 0 2147483647, normal conversion. Except this, an error occurs.
DINT_TO_BOOL	BOOL	Takes the low 1 bit and converts into BOOL type.
DINT_TO_BYTE	BYTE	Takes the low 8 bit and converts into BYTE type.
DINT_TO_WORD	WORD	Takes the low 18 bit and converts into WORD type.
DINT_TO_DWORD	DWORD	Converts into DWORD type without changing the internal bit array.
DINT_TO_LWORD	LWORD	Converts into LWORD type filling the upper bytes with 0.
DINT_TO_BCD	DWORD	If input is 0 99,999,999, normal conversion. Except this, an error occurs.
DINT_TO_REAL	REAL	Converts DINT into REAL type. During conversion, an error caused by the precision may occur.
DINT_TO_LREAL	LREAL	Converts DINT into LREAL type. During conversion, an error caused by the precision may occur.

#### ■ Error

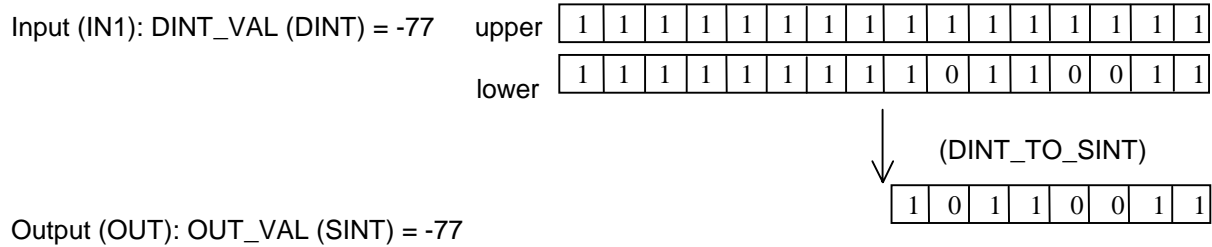
If a conversion error occurs, \_ERR, \_LER flags will be set.

When an error occurs, it takes as many lower bits as the bit number of the output type and produces an output without changing the internal bit array.

■ Program Example



- (1) If the transition condition (%M1) is on, DINT\_TO\_SINT function will be executed.
- (2) If IN1 = DINT\_VAL (DINT) = -77, SINT\_VAL (SINT) = -77.

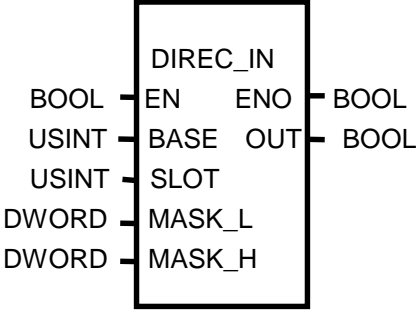


## 8. Basic Function/Function Block Library

### DIREC\_IN

Update input data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            BASE: base number of an input module installed            SLOT: slot number of an input module installed            MASK_L: designates bits not to be updated among lower 32-bit data of input            MASK_H: designates bits not to be updated among upper 32-bit data of input</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: if update is completed, output will be 1.</p>

#### ■ Function

If EN is 1 during the scan, DIREC\_IN function reads 64-bit data of an input module from the designated position of BASE and SLOT and updates them.

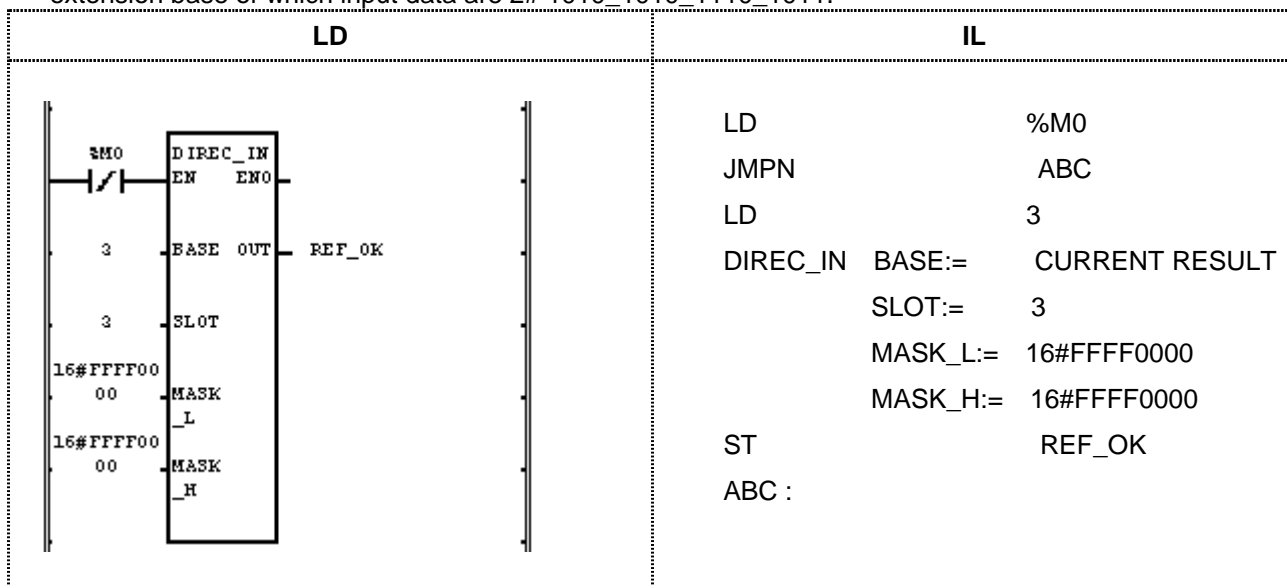
At this time, only the actual contacts of an input module will be updated in the image scope.

DIREC\_IN function is available to use when you want to change the ON/OFF state of input (%I) during the scan.

Generally, it's impossible to update input data during 1 scan (executing a scan program) because a scan-synchronized batch processing mode executes the batch processing to read input data and produce output data after a scan program. It's available to update related input data, if you use DIREC\_IN function during program execution.

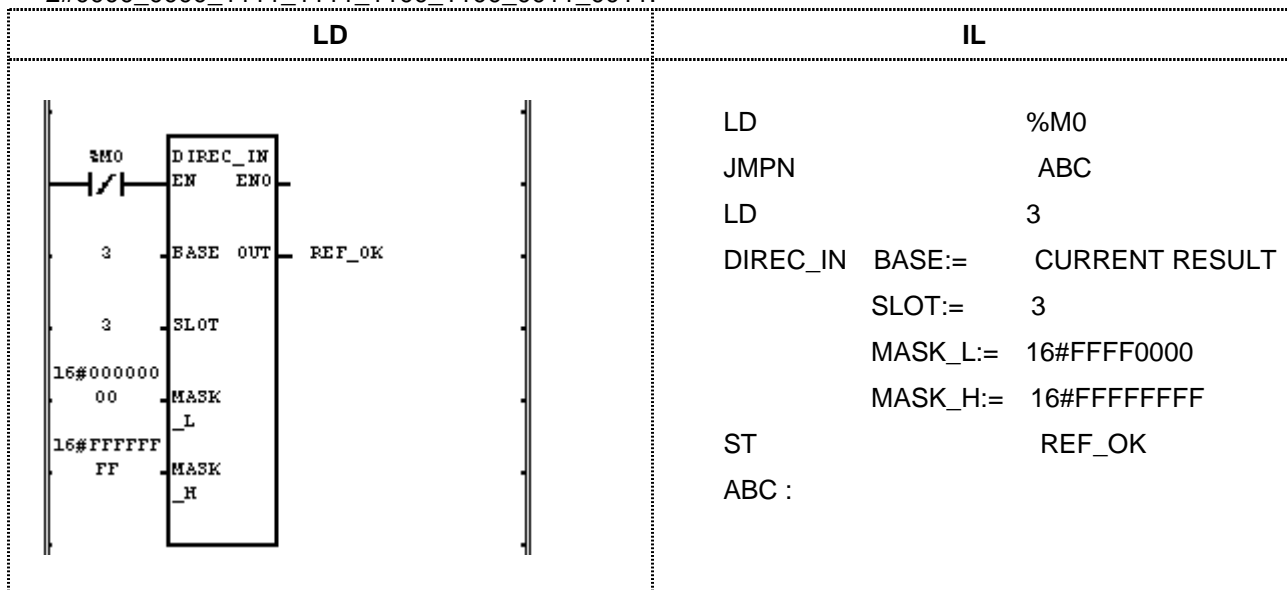
■ Program Example

1. This is the program that updates a 16-contact module installed in the 4th slot (slot number is 3) of the 3rd extension base of which input data are 2# 1010\_1010\_1110\_1011.



- (1) If the input condition (%M0) is on, function DIREC\_IN will be executed.
- (2) The image scope to update will be %IW3.3.0 and %IW3.3.0 will be updated with 2#1010\_1010\_1110\_1011 during the scan because a 16-contact module is installed and the lower 16-bit data update is allowed (MASK\_L = 16#FFFF0000).
- (3) It doesn't matter what data are set in MASK\_H because a 16-contact module is installed.

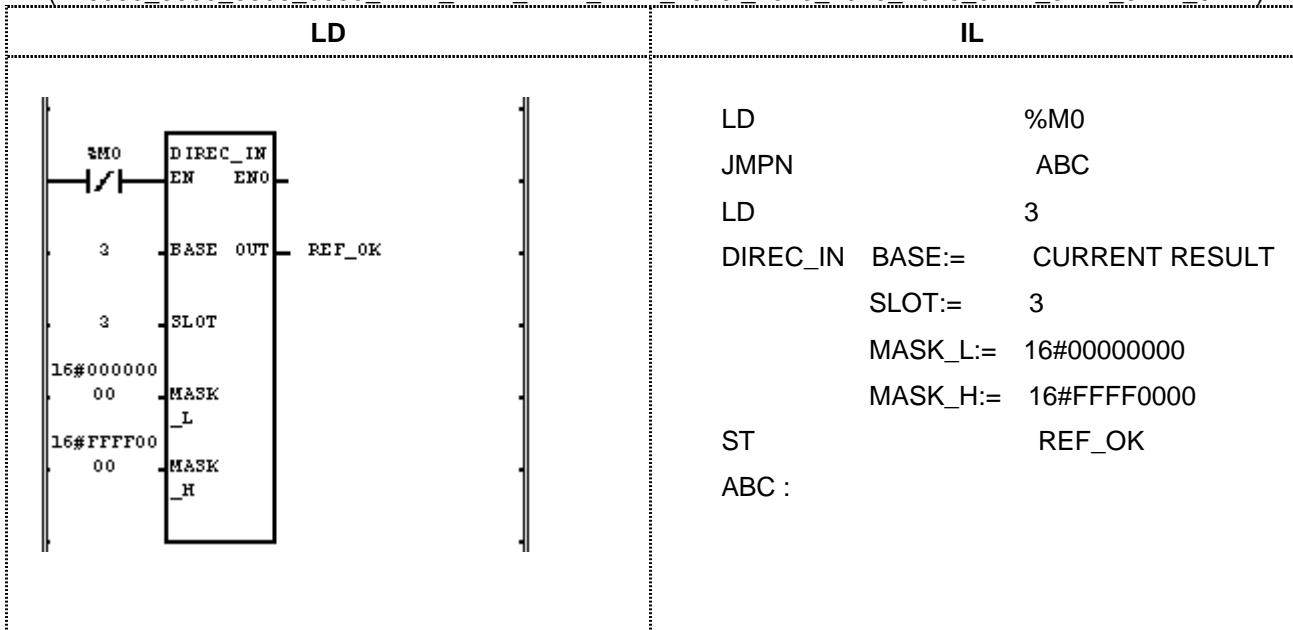
2. This is the program that updates the lower 16-bit data of the 32-contact module installed in the 4th slot (slot number is 3) of the 3rd extension base of which input data are 2#0000\_0000\_1111\_1111\_1100\_1100\_0011\_0011.



- (1) If input condition (%M0) is on, function DIREC\_IN will be executed.
- (2) The image scope to update will be %ID3.3.0 but only %IW3.3.0 will be updated with 2#1100\_1100\_0011\_0011 during the scan because a 16-contact module is installed and the lower 16-bit data update is allowed (MASK\_L = 16#FFFF0000).

## 8. Basic Function/Function Block Library

3. This is the program that updates the lower 48-bit data of the 64-contact module installed in the 4th slot (slot number is 3) of the 3rd extension base of which input data are 16#0000\_FFFF\_AAAA\_7777 (2#0000\_0000\_0000\_0000\_1111\_1111\_1111\_1111\_1010\_1010\_1010\_1010\_0111\_0111\_0111\_0111).



- (1) If the input condition (%M0) is on, function DIREC\_IN will be executed.
- (2) The installed module is a 64-contact module and the image scope to update will be %IL3.3.0 (%ID3.3.0 and ID3.3.1).  
 %ID3.3.0 will be updated because the lower 32-bit data update is allowed (MASK\_L = 16#00000000).  
 %IW3.3.2 of %ID3.3.1 will be updated because only the lower 16-bit data update (among upper 32 bits) is allowed (MASK\_H = 16#FFFF0000).

Accordingly, the data update of the image scope is as follows:



- (3) If the input update is completed, output REF\_OK will be 1.

## DIREC\_O

Update output data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            BASE: base number of an input module installed            SLOT: slot number of an input module installed            MASK_L: designates bits not to be updated among lower 32-bit data of output            MASK_H: designates a bit not to update among upper 32-bit data of output</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1.            OUT: If update is completed, output will be 1.</p>

### Function

If EN is 1 during the scan, DIREC\_O function reads 64-bit data of an output module from the designated position of BASE and SLOT and updates the unmasked (MASK (0)) data.

DIREC\_O is available to use when you want to change the ON/OFF state of output (%Q) during the scan.

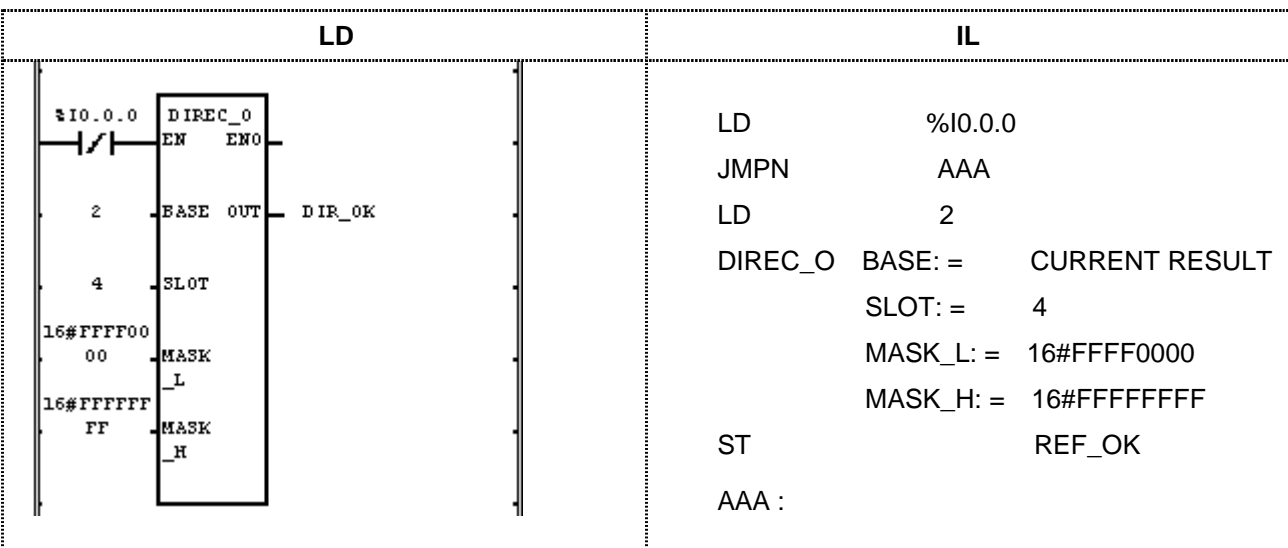
Generally, it's impossible to update input data during 1 scan (executing a scan program) because a scan-synchronized batch processing mode executes the batch processing to read input data and produce output data after a scan program.

It's available to update related output data, if you use DIREC\_O function during program execution.

If the base/slot number is wrong or it is not available to write data normally in an output module, ENO and OUT are '1' (without an error, it will be 1).

### Program Example

1. This is the program that produces output data 2#0111\_0111\_0111\_0111 in a 16-contact relay output module installed in the 5th slot (slot number is 4) of the 2nd extension base.



(1) Input the slot and base number in which an output module installed.

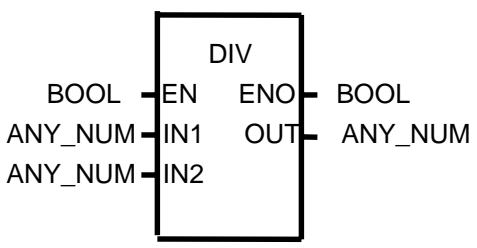




## DIV

Division
----------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: the value to be divided (dividend) IN2: the value to divide (divisor)</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the divided result (quotient)</p> <p>The variable connected to IN1, IN2 and OUT should be all the same data type.</p>

### Function

It divides IN1 by IN2 and produces an output omitting decimal fraction from the quotient.

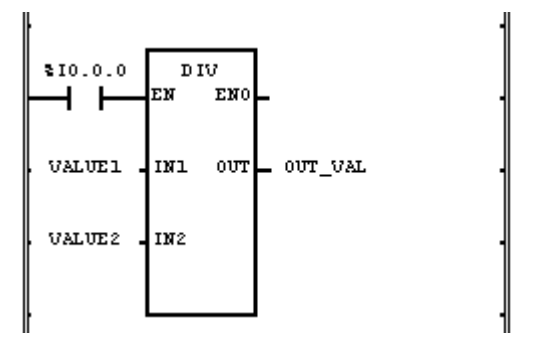
$$OUT = IN1/IN2$$

IN1	IN2	OUT	Remarks
7	2	3	Decimal fraction omitted.
7	-2	-3	
-7	2	-3	
-7	-2	3	
7	0	x	Error

### Error

If the value to divide (divisor) is '0', \_ERR, \_LER flags will be set.

### Program Example

LD	IL
	<pre>LD      %I0.0.0 JMPN   LL LD      VALUE1 DIV     IN1:= CURRENT RESULT         IN2:= VALUE2 ST      OUT_VAL LL :</pre>

(1) If the transition condition (%I0.0.0) is on, DIV function will be executed.

(2) If input VALUE1 = 300 and VALUE2 = 100, then output OUT\_VAL = 300/100 = 3.

Input (IN1): VALUE1 (INT) = 300 (16#012C)	0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0
	/ (DIV)
(IN2): VALUE2 (INT) = 100 (16#0064)	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
Output (OUT): OUT_VAL (INT) = 3 (16#3)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

## 8. Basic Function/Function Block Library

### DIV\_TIME

Time division
---------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: Time to divide IN2: The value to divide</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: divided result time</p>

#### ■ Function

It divides IN1 (time) by IN2 (number) and produces output OUT (divided time).

#### ■ Error

If a divisor (IN2) is 0, \_ERR and \_LER flags will be set.

#### ■ Program Example

This is the program that calculates the time required to produce one product in some product line if the working time of day is 12hr 24min 24sec and product quantity of a day is 12 in a product line.

LD	IL
	<pre> LD          %I0.1.0 JMPN       SS LD          TOTAL_TIME DIV_TIME   IN1:= CURRENT_RESULT            IN2:= PRODUCT_COUNT ST         TIME_PER_PRO SS : </pre>

(1) If the transition condition (%I0.1.0) is on, DIV\_TIME function will be executed.

(2) If it divides TOTAL\_TIME (T#12H24M24S) by PRODUCT\_COUNT (12), the time required to produce one product TIME\_PER\_PRO (T#1H2M2S) will be an output. That is, it takes 1hr 2min 2sec to produce one product.

Input (IN1): TOTAL\_TIME (TIME) = T#12H24M24S

/ (DIV\_TIME)

(IN2): PRODUCT\_COUNT (INT) = 12



Output (OUT): TIME\_PER\_PRO (TIME) = T#1H2M2S

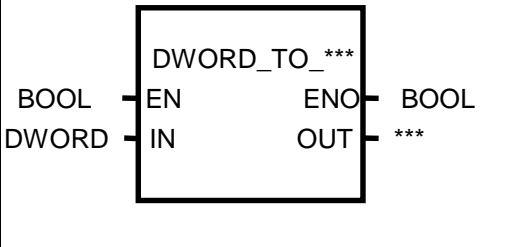


## 8. Basic Function/Function Block Library

### DWORD\_TO\_\*\*\*

DWORD type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

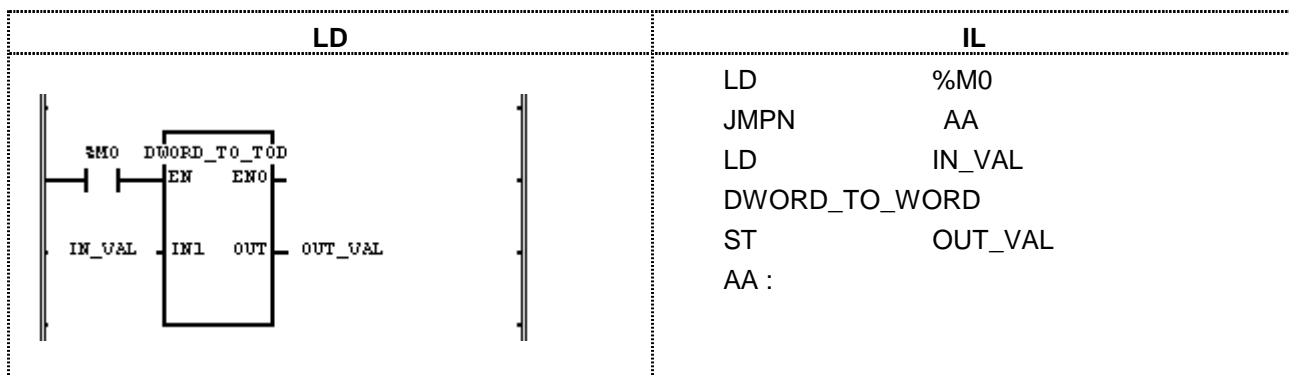
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: bit string to convert (32bit)</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

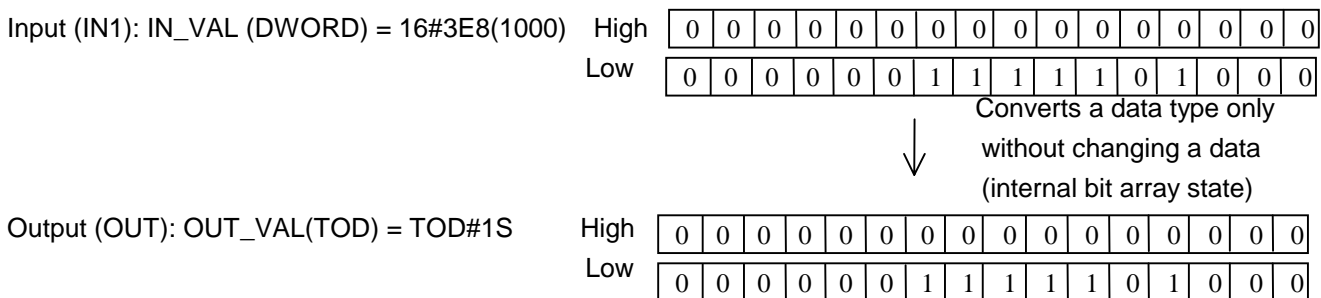
It converts Input IN type and produces output OUT.

Function	Output type	Description
DWORD_TO_SINT	SINT	Takes the lower 8 bits and converts into SINT type.
DWORD_TO_INT	INT	Takes the lower 16 bits and converts into INT type.
DWORD_TO_DINT	DINT	Converts into DINT type without changing the internal bit array.
DWORD_TO_LINT	LINT	Converts into LINT type filling the upper bits with 0
DWORD_TO_USINT	USINT	Takes the lower 8 bits and converts into USINT type.
DWORD_TO_UINT	UINT	Takes the lower 16 bits and converts into UINT type.
DWORD_TO_UDINT	UDINT	Converts into UDINT type without changing the internal bit array.
DWORD_TO_ULINT	ULINT	Converts into ULINT type filling the upper bits with 0.
DWORD_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
DWORD_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
DWORD_TO_WORD	WORD	Takes the lower 16 bits and converts into WORD type.
DWORD_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
DWORD_TO_REAL	REAL	Converts into REAL type without changing the internal bit array.
DWORD_TO_TIME	TIME	Converts into TIME type without changing the internal bit array.
DWORD_TO_TOD	TOD	Converts into TOD type without changing the internal bit array.
DWORD_TO_STRING	STRING	Changes input value into decimal and converts into STRING type.

■ Program Example



- (1) If the transition condition (%M0) is on, DWORD\_TO\_TOD function will be executed.
- (2) If output IN\_VAL (DWORD) = 16#3E8 (1000), output OUT\_VAL (TOD) = TOD#1S.



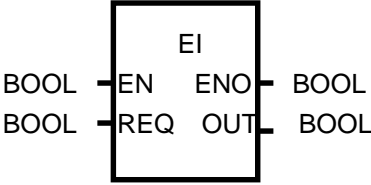
Calculates TIME, TOD by converting decimal into MS unit. That is, 1000 is 1000ms = 1s.  
Refer to 3.2.4. Data Type Structure.

## 8. Basic Function/Function Block Library

### EI

Permits running for task program

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 REQ: requires to permit running for task program</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: If EI is executed, an output will be 1.</p>

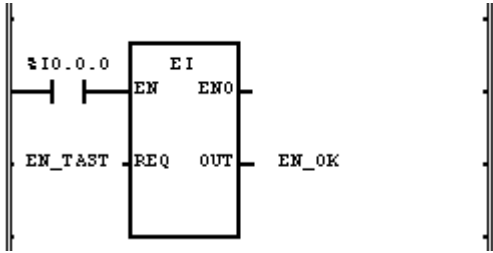
#### ■ Function

If EN is 1 and REQ input is 1, task program blocked by 'DI' function starts normally.

Once 'EI' command is executed, task program starts normally even if REQ input is 0.

Task programs created when they are not permitted to operate will be executed after 'EI' function or the current-running task program execution.

#### ■ Program Example (refer to DI)

LD	IL
	<pre>LD      %I0.0.0 JMPN   LSB LD      EN_TASK EI ST      EN_OK LSB :</pre>

If EN\_TASK is 1, a task program starts normally.

If EI function permits running for a task program, output EN\_OK will be 1.

**EQ**

'Equal to' comparison

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: the value to be compared            IN2: The value to compare            Input variable number can be extended up to 8.            IN1, IN2, ... should be the same type.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: comparison result value</p>

■ **Function**

If IN1 = IN2 = IN3 ... = INn (n : input number), output OUT will be 1.  
 In other cases, OUT will be 0.

■ **Program Example**

LD	IL
	<pre> LD      %I0.1.0 JMPN   AA LD      VALUE1 EQ     IN1:= CURRENT RESULT        IN2:= VALUE2        IN3:= VALUE3 ST      %Q0.0.1 AA :           </pre>

- (1) If the transition condition (%I0.1.0) is on, EQ function will be executed.
- (2) If VALUE1 = 300, VALUE2 = 300, VALUE3 = 300 (comparison result VALUE1 = VALUE2 = VALUE3), output %Q0.0.1 = 1.

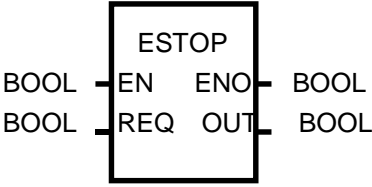
Input (IN1): VALUE1 (INT) = 300 (16#012C)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;">= (EQ)</p>	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0		
(IN2): VALUE2 (INT) = 300 (16#012C)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;">= (EQ)</p>	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0		
(IN3): VALUE1 (INT) = 300 (16#012C)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td> </tr> </table>	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0		
Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)	<div style="text-align: right; margin-right: 20px;">↓</div> <table border="1" style="width: 20px; height: 20px; margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td> </tr> </table>	1																
1																		

## 8. Basic Function/Function Block Library

### ESTOP

Emergency running stop by program

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 REQ: requires the emergency running stop</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: If ESTOP is executed, an output will be 1.</p>

#### ■ Function

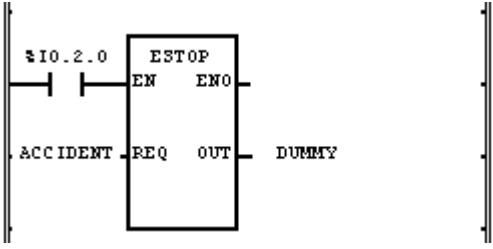
If transition condition EN is 1 and the signal to require the emergency running stop by program REQ is 1, program operation stops immediately and returns to STOP mode.

In case that a program stops by 'ESTOP' function, it does not start despite of power re-supply.

If operation mode moves from STOP to RUN, it restarts.

If 'ESTOP' function is executed, the running program stops during operation; if it is not a cold restart mode, an error may occur when restarts.

#### ■ Program Example

LD	IL
	<pre>LD      %I0.2.0 JMPN   SSS LD      ACCIDENT ESTOP (ST    DUMMY) SSS :</pre>

(1) If the transition condition (%I0.2.0) is on, ESTOP function will be executed.

(2) If ACCIDENT = 1, the running program stops immediately and returns to STOP mode.

In case of emergency, it is available to use it as a double safety device with mechanical interrupt.



**EXP**

Natural exponential operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of exponent operation</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: result value IN, OUT should be the same data type.</p>

■ **Function**

It calculates the natural exponent with exponent IN and produces output OUT.

$$OUT = e^{IN}$$

■ **Program Example**

LD	IL
	<pre>LD      %M5 JMPN   JJ LD      INPUT EXP ST      RESULT JJ :</pre>

(1) If the transition condition (%M5) is on, EXP function will be executed.

(2) If INPUT is 2.0, RESULT will be 7.3890...

$$e^{2.0} = 7.3890.....$$

Input (IN1): INPUT (REAL) = 2.0

High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Low	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0

(16#40000000)

(EXP)

Output (OUT): RESULT (REAL) = 7.38905621E+00

High	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Low	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0	0

(16#40EC7326)

## 8. Basic Function/Function Block Library

### EXPT

Exponential operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: real number IN2: exponent</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: result value</p> <p>IN1 and OUT should be the same data type.</p>

#### ■ Function

It calculates IN1 with exponent IN2 and produces output OUT.

$$OUT = IN1^{IN2}$$

#### ■ Error

If an output is out of range of related data type, \_ERR and \_LER flags will be set.

#### ■ Program Example

LD	IL
	<pre> LD      %I0.1.0 JMPN   LSB LD      IN_VAL EXPT   IN1:= CURRENT RESULT         IN2:= VALUE ST      OUT_VAL LSB : </pre>

(1) If the transition condition (%I0.1.0) is on, 'EXPT' exponential function will be executed.

(2) If input IN\_VAL = 1.5, VALUE = 3, output OUT\_VAL =  $1.5^3 = 1.5 \times 1.5 \times 1.5 = 3.375$ .

Input (IN1): IN\_VAL (REAL) = 1.5

(IN2): VALUE (INT) = 3



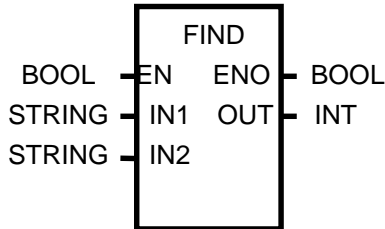
(EXPT)

Output (OUT): OUT\_VAL (REAL) = 3.37500000E+00

## FIND

Finds a character string

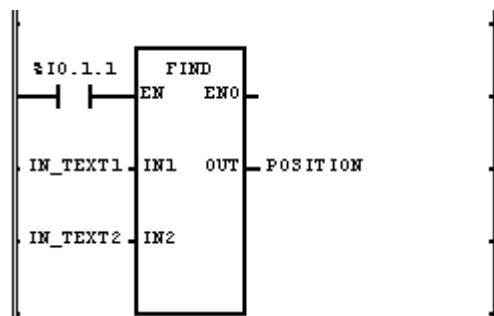
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: input character string IN2: character string to find</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: location of character string to be found</p>

### ■ Function

It finds the location of character string IN2 from input character string IN1. If the location is found, it shows a position of a first character of character string IN2 from character string IN1. Otherwise, output will be 0.

### ■ Program Example

LD	IL
	<pre>LD      %I0.1.1 JMPM   XYZ LD      IN_TEXT1 FIND   IN1:= CURRENT RESULT         IN2:= IN_TEXT2 ST      POSITION XYZ :</pre>

- (1) If the transition condition (%I0.1.1) is on, FIND function will be executed.
- (2) If input character string IN\_TEXT1='ABCEF' and IN\_TEXT2='BC', then output variable POSITION = 2.
- (3) The first location of IN\_TEXT2 ('BC') from input character string IN\_TEXT1 ('ABCEF') is 2<sup>nd</sup>.

Input (IN1): IN\_TEXT1 (STRING) = 'ABCEF'

(FIND)

(IN2): IN\_TEXT2 (STRING) = 'BC'



Output (OUT): POSITION (INT) = 2

## 8. Basic Function/Function Block Library

### GE

'Greater than or equal to' comparison

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: the value to be compared            IN2: the value to compare            Input variable number can be extended up to 8.            IN1, IN2, ... should be the same data type.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: comparison result value</p>

#### ■ Function

If  $IN1 \geq IN2 \geq IN3 \dots \geq INn$  (n: input number), an output will be 1.  
 Otherwise it will be 0.

#### ■ Program Example

LD	IL
	<pre>LD      %M77 JMPN   YY LD      VALUE1 GE     IN1=  CURRENT RESULT        IN2=  VALUE2        IN3=  VALUE3 ST      %Q0.0.1 YY:</pre>

- (1) If the transition condition (%M77) is on, GE function will be executed.
- (2) If input variable VALUE1 = 300, VALUE3 = 200, comparison result will be  $VALUE1 \geq VALUE2 \geq VALUE3$ . The output %Q0.01 = 1.

Input (IN1): VALUE1 (INT) = 300 (16#012C)	0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0
	(GE)
(IN2): VALUE2 (INT) = 200 (16#00C8)	0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
	(GE)
(IN3): VALUE3 (INT) = 100 (16#0064)	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
	↓
Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)	1

# GT

'Greater than' comparison

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: the value to be compared            IN2: the value to compare            Input variable number can be extended up to 8.            IN1, IN2, ... should be the same data type.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: comparison result value</p>

## Function

If  $IN1 > IN2 > IN3 \dots > INn$  (n: input number), an output will be 1.  
 Otherwise it will be 0.

## Program Example

LD	IL
	<pre> LD      %M0 JMPN   AAA LD      VALUE1 GT      IN1:= CURRENT RESULT         IN2:= VALUE2         IN3:= VALUE3 ST      %Q0.0.1 AAA :           </pre>

- (1) If the transition condition (%M0) is on, GT function will be executed.
- (2) If input variable VALUE1 = 300, VALUE2 = 200, and VALUE3 = 100, comparison result will be VALUE1 > VALUE2 > VALUE3. The output %Q0.0.1 = 1.

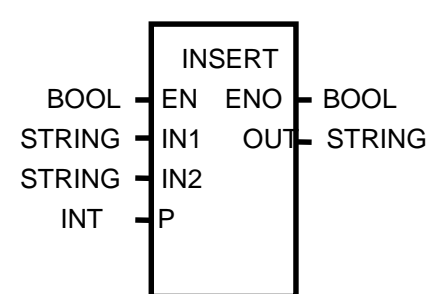
Input (IN1): VALUE1 (INT) = 300 (16#012C)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;">&gt; (GT)</p>	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0		
(IN2): VALUE2 (INT) = 200 (16#00C8)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td> </tr> </table> <p style="text-align: center;">&gt; (GT)</p>	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0		
(IN3): VALUE3 (INT) = 100 (16#0064)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td> </tr> </table>	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0		
Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)	<div style="text-align: right; margin-right: 20px;">↓</div> <table border="1" style="width: 50px; height: 20px; margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">1</td> </tr> </table>	1																
1																		

## 8. Basic Function/Function Block Library

### INSERT

Inserts a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1  IN1: character string to be inserted  IN2: character string to insert  P: position to insert a character string</p> <p><b>Output</b> ENO: without an error, it will be 1.  OUT: output character string</p>

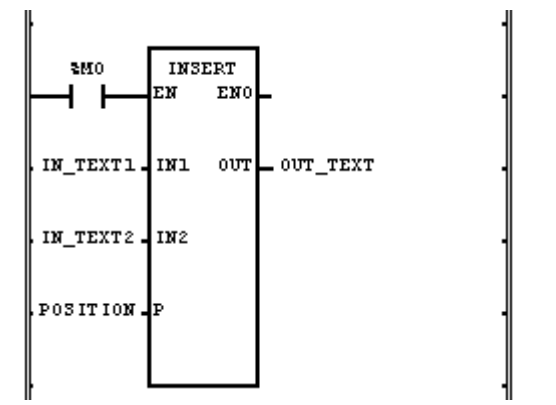
#### ■ Function

It inserts character string IN2 after the P character of IN1 and produces output OUT.

#### ■ Error

If  $P \leq 0$ , 'character number of variable IN1'  $< P$ , or if the character number of result exceeds 30 (just 30 characters are produced), then `_ERR`, `_LER` flags will be set.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AA LD      IN_TEXT1         INSERT IN1:= CURRENT RESULT         IN2:= IN_TEXT2         P:= POSITION ST      OUT_TEXT AA:</pre>

- (1) If the transition condition (%M0) is on, INSERT function will be executed.
- (2) If input variable IN\_TEXT1 = 'ABCD', IN\_TEXT2 = 'XY', and POSITON = 2, output variable OUT\_TEXT = 'ABXYCD'.

Input (IN1): IN\_TEXT1 (STRING) = 'ABCD'  
(IN2): IN\_TEXT2 (STRING) = 'XY'  
(P): POSITION (INT) = 2  
↓ (FIND)  
Output (OUT): OUT\_TEXT = 'ABXYCD'

## INT\_TO\_\*\*\*

INT type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: integer value to convert</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

### ■ Function

It converts input IN type and produces output OUT.

Function	Output type	Description
INT_TO_SINT	SINT	If input is -128 ~ 127, normal conversion. Except this, an error occurs.
INT_TO_DINT	DINT	Converts into DINT type normally.
INT_TO_LINT	LINT	Converts into LINT type normally.
INT_TO_USINT	USINT	If input is 0 ~ 255, normal conversion. Except this, an error occurs.
INT_TO_UINT	UINT	If input is 0 ~ 32767, normal conversion. Except this, an error occurs.
INT_TO_UDINT	UDINT	If input is 0 ~ 32767, normal conversion. Except this, an error occurs.
INT_TO_ULINT	ULINT	If input is 0 ~ 32767, normal conversion. Except this, an error occurs.
INT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
INT_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
INT_TO_WORD	WORD	Converts into WORD type without changing the internal bit array.
INT_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
INT_TO_LWORD	LWORD	Converts into LWORD type filling the high bit with 0.
INT_TO_BCD	WORD	If input is 0~9,999, normal conversion. Except this, an error occurs.
INT_TO_REAL	REAL	Converts INT into REAL type normally.
INT_TO_LREAL	LREAL	Converts INT into LREAL type normally.

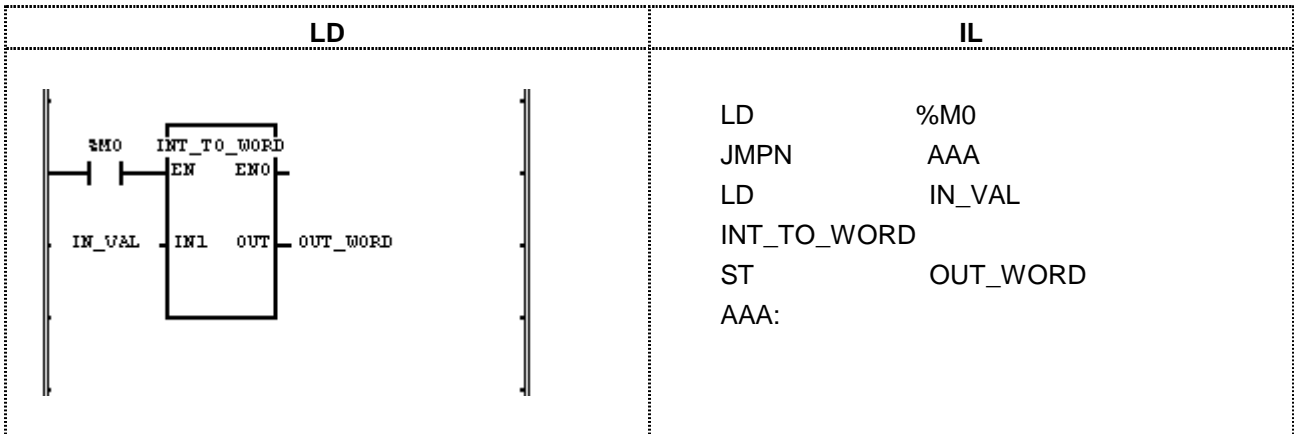
### ■ Error

If a conversion error occurs, \_ERR\_LER flags will be set.

If an error occurs, take as many lower bits as the bit number of the output type and produces an output without changing the internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example



- (1) If the input condition (%M0) is on, INT\_TO\_WORD function will be executed.
- (2) If input variable IN\_VAL (INT) = 512 (16#200), output variable OUT\_WORD (WORD) = 16#200.

Input (IN1): IN\_VAL (INT) = 512 (16#200)

0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↓ (INT\_TO\_WORD)

Output (OUT): OUT\_WORD (WORD) = 16#200

0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



**LE**

'Less than or equal to' comparison

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1                      IN1: the value to be compared                      IN2: the value to compare                      Input variable number can be extended up to 8.                      IN1, IN2, ...should be the same data type.</p> <p><b>Output</b> ENO: without an error, it will be 1.                      OUT: comparison result value</p>

■ **Function**

If  $IN1 \leq IN2 \leq IN3 \dots \leq INn$  (n: input number), output OUT will be 1.  
 Otherwise it will be 0.

■ **Program Example**

LD	IL
	<pre> LD      %M0 JMPN   BBB LD      VALUE1 LE     IN1:= CURRENT RESULT         IN2:= VALUE2         IN3:= VALUE3 ST      %Q0.0.1 BBB:     </pre>

- (1) If the transition condition (%M0) is on, LE function will be executed.
- (2) If input variable VALUE1 = 150, VALUE2 = 200, and VALUE3 = 250, output %Q0.0.1 = 1 (VALUE1 ≤ VALUE2 ≤ VALUE3).

Input (IN1): VALUE1 (INT) = 150 (16#0096)	0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0
	≤ (LE)
(IN2): VALUE2 (INT) = 200 (16#00C8)	0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
	≤ (LE)
(IN3): VALUE1 (INT) = 250 (16#0064)	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0
Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)	↓ 1

## 8. Basic Function/Function Block Library

### LEFT

Takes the left side of a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            IN: input character string            L: length of character string</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1.            OUT: output character string</p>

#### ■ Function

It takes a left character string (L) of IN and produces output OUT.

#### ■ Error

If  $L < 0$ , `_ERR` and `_LER` flags will be set.

#### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   FF  LD      IN_TEXT LEFT   IN:= CURRENT RESULT         L:=  LENGTH ST      OUT_TEXT FF: </pre>

(1) If the transition condition (%M0) is on, function LEFT function will be executed.

(2) If input variable IN\_TEXT = 'ABCDEFGH' and LENGTH = 3, output character string OUT\_TEXT = 'ABC'.

Input (IN1): IN\_TEXT (STRING) = 'ABCDEFGH'  
 (IN2): LENGTH (INT) = 3

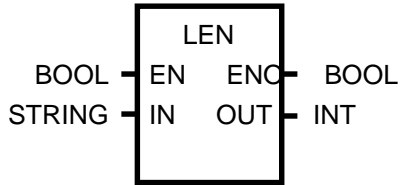
↓ (LEFT)

Output (OUT): OUT\_TEXT (STRING) = 'ABC'

**LEN**

Finds a length of a character string

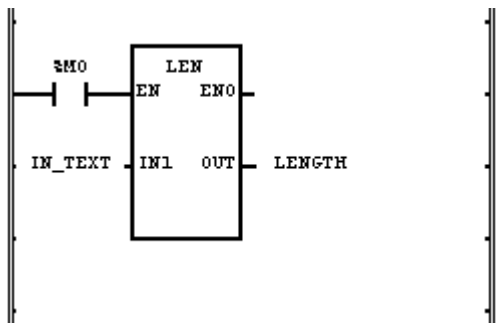
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input character string</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the length of a character string</p>

■ **Function**

It produces a length (character number) of the input character string (IN).

■ **Program Example**

LD	IL
	<pre> LD      %M0 JMPN   I1 LD      IN_TEXT LEN    IN:= CURRENT RESULT ST      LENGTH I1:         </pre>

- (1) If the transition condition (%M0) is on, LEN function will be executed.
- (2) If input variable IN\_TEXT = 'ABCD', output variable LENGTH = 4.

Input (IN1): IN\_TEXT (STRING) = 'ABCD'  
 ↓ (LEN)  
 Output (OUT): LENGTH (INT) = 4

## 8. Basic Function/Function Block Library

### LIMIT

Limits upper and lower boundary

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1  MN: minimum value  IN: the value to be limited  MX: maximum value</p> <p><b>Output</b> ENO: without an error, it will be 1.  OUT: value in the range</p> <p>MN, IN, MX, OUT should be the same data type.</p>

#### ■ Function

If input IN value is between MN and MX, the IN will be an output.

That is, if  $MN \leq IN \leq MX$ ,  $OUT = IN$

If input IN value is less than MN, MN will be an output. That is, if  $IN < MN$ ,  $OUT = MN$ .

If input IN value is greater than MX, MX will be an output. That is, if  $IN > MX$ ,  $OUT = MX$

#### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   MM LD      LIMIT_LOW LIMIT  MN:= CURRENT RESULT         IN := IN_VALUE         MX:= LIMIT_HIGH ST      OUT_VAL MM: </pre>

(1) If the transition condition (%M0) is on, LIMIT function will be executed.

(2) Output variable OUT\_VAL for lower limit input LIMIT\_LOW, upper limit input (LIMIT\_HIGH) and limited value input IN\_VALUE will be as follows:

LIMIT_LOW	IN_VALUE	LIMIT_HIGH	OUT_VAL
1000	2000	3000	2000
1000	500	3000	1000
1000	4000	3000	3000

Input (MN): LIMIT\_LOW (INT) = 1000

(IN): IN\_VALUE (INT) = 4000

(MX): IN\_VALUE (INT) = 3000

↓ (LIMIT)

Output (OUT): OUT\_VAL (INT) = 3000

**LINT\_TO\_\*\*\***

LINT type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: long integer value to convert</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type converted data</p>

■ **Function**

It converts input IN type and produces output OUT.

Function	Output type	Description
LINT_TO_SINT	SINT	If input is -128 127, normal conversion. Otherwise an error occurs.
LINT_TO_INT	INT	If input is -32,768 32,767, normal conversion. Otherwise an error occurs.
LINT_TO_DINT	DINT	If input is $-2^{31}$ $2^{31}-1$ , normal conversion. Otherwise an error occurs.
LINT_TO_USINT	USINT	If input is 0 255, normal conversion. Otherwise an error occurs.
LINT_TO_UINT	UINT	If input is 0 65,535, normal conversion. Otherwise an error occurs.
LINT_TO_UDINT	UDINT	If input is 0 $2^{32}-1$ , normal conversion. Otherwise an error occurs.
LINT_TO_ULINT	ULINT	If input is 0 $2^{63}-1$ , normal conversion. Otherwise an error occurs.
LINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
LINT_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
LINT_TO_WORD	WORD	Takes the lower 16 bits and converts into WORD type.
LINT_TO_DWORD	DWORD	Takes the lower 32 bits and converts into DWORD type.
LINT_TO_LWORD	LWORD	Converts into LWORD type without changing the internal bit array.
LINT_TO_BCD	LWORD	If input is 0~9,999,999,999,999,999, normal conversion. Otherwise an error occurs.
LINT_TO_REAL	REAL	Converts LINT into REAL type. During the conversion, an error caused by the precision may occur.
LINT_TO_LREAL	LREAL	Converts LINT into LREAL type. During the conversion, an error caused by the precision may occur.

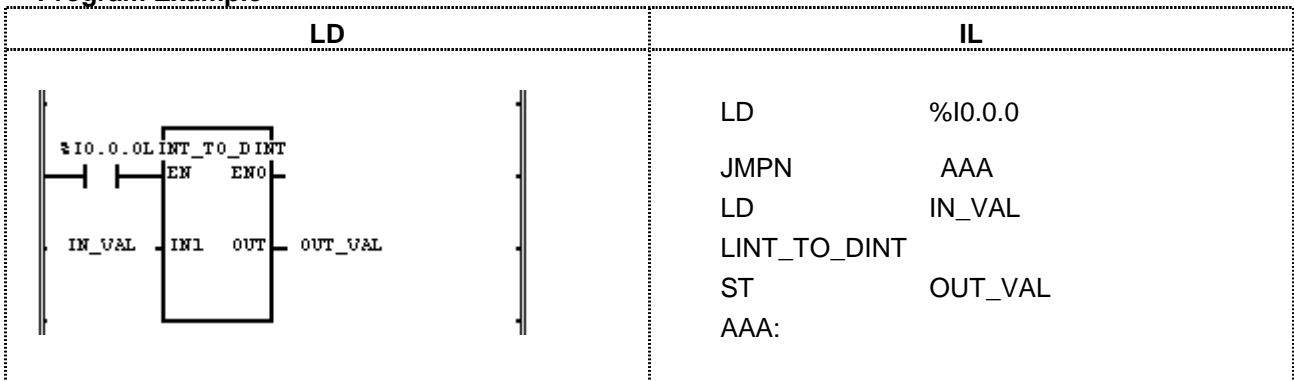
■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set.

If an error occurs, take as many lower bits as the bit number of the output type and produces an output without changing the Internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example



- (1) If the input condition (%I0.0.0) is on, LINT\_TO\_DINT function will be executed.
- (2) If input variable IN\_VAL (LINT) = 123\_456\_789, output variable OUT\_VAL (DINT) = 123\_456\_789.

Input (IN1): IN\_VAL (LINT) = 123,456,789  
(16#75BCD15)

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

↓ (LINT\_TO\_DINT)

Output (OUT): OUT\_VAL (DINT) = 123,456,789  
(16#75BCD15)

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

## LN

Natural logarithm operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of natural logarithm operation</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: natural logarithm value IN, OUT should be the same data type.</p>

### ■ Function

It finds a natural logarithm value of IN and produces output OUT.

$$OUT = \ln IN$$

### ■ Error

If an input is 0 or a negative number, \_ERR and \_LER flags will be set.

### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AE LD      INPUT LN ST      RESULT AE:</pre>

(1) If the transition condition (%M0) is on, LN function will be executed.

(2) If input variable INPUT is 2.0, output variable RESULT will be 0.6931 ....

$$\ln(2.0) = 0.6931\dots$$

Input (IN1): INPUT (REAL) = 2.0

↓ (LN)

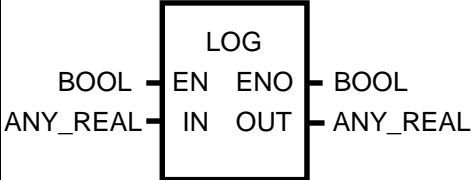
Output (OUT): RESULT (REAL) = 6.93147182E-01

## 8. Basic Function/Function Block Library

### LOG

Base 10 Logarithm operation

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of common logarithm operation</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the value of common logarithm operation IN, OUT should be the same data type.</p>

#### ■ Function

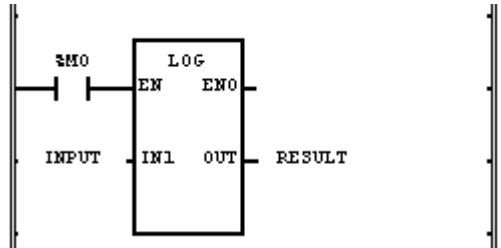
It finds the value of Base 10 Logarithm of IN and produces output OUT.

$$OUT = \log_{10} IN = \log IN$$

#### ■ Error

If input value IN is 0 or a negative number, \_ERR and \_LER flags will be set.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   BB LD      INPUT LOG ST      RESULT BB:</pre>

(1) If the transition condition (%M0) is on, LOG function will be executed.

(2) If input variable INPUT is 2.0, output variable RESULT will be 0.3010 .....

$$\log_{10} (2.0) = 0.3010\dots$$

Input (IN1): INPUT (REAL) = 2.0

↓ (LOG)

Output (OUT): RESULT (REAL) = 3.01030010E-01



### LREAL\_TO\_\*\*\*

LREAL type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: LREAL value to convert</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type converted data</p>

#### ■ Function

It converts input IN type and produces output OUT.

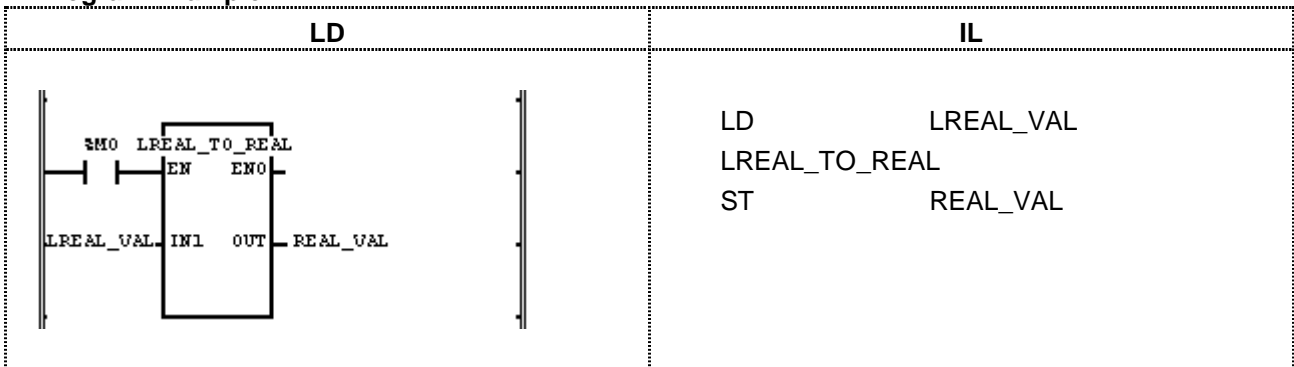
Function	Output type	Description
LREAL_TO_SINT	SINT	If integer number of input is -128 127, normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_INT	INT	If integer number of input is -32768 32767, normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_DINT	DINT	If integer number of input is $-2^{31}$ $2^{31}-1$ , normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_LINT	LINT	If integer number of input is $-2^{63}$ $2^{63}-1$ , normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_USINT	USINT	If integer number of input is 0 255, normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_UINT	UINT	If integer number of input is 0 65,535, normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_UDINT	UDINT	If integer number of input is 0 $2^{32}-1$ , normal conversion. Otherwise an error occurs (decimal round off).
LREAL_TO_ULINT	ULINT	If integer number of input is 0 $2^{64}-1$ , normal conversion. Otherwise an error occurs (decimal round-off).
LREAL_TO_LWORD	LWORD	Converts into LWORD type without changing the internal bit array.
LREAL_TO_REAL	REAL	Converts LREAL into REAL type normally. During the conversion, an error caused by the precision may occur.

#### ■ Error

If an overflow occurs because an input value is greater than the value available for the output type, \_ERR and \_LER flags will be set. If an error occurs, an output will be 0.

## 8. Basic Function/Function Block Library

### ■ Program Example



- (1) If the input condition (%M0) is on, LREAL\_TO\_REAL function will be executed.
- (2) If input variable LREAL\_VAL (LREAL) = -1.34E-12, output variable REAL\_VAL (REAL) = -1.34E-12.

Input (IN1): LREAL\_VAL (LREAL) = -1.34E-12

↓ (LREAL\_TO\_REAL)

Output (OUT): REAL\_VAL (REAL) = -1.34E-12

**LT**

'Less than' comparison

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1                      IN1: the value to be compared                      IN2: the value to compare                      Input variable number can be extended up to 8.                      IN1, IN2, ...should be the same data type.</p> <p><b>Output</b> ENO: without an error, it will be 1.                      OUT: comparison result value</p>

■ **Function**

If  $IN1 < IN2 < IN3 \dots < INn$  (n: input number), output value OUT will be 1.  
 Otherwise output OUT will be 0.

■ **Program Example**

LD	IL
	<pre> LD      %M0 JMPN   AA LD      VALUE1 LT      IN1:= CURRENT RESULT         IN2:= VALUE2         IN3:= VALUE3 ST      %Q0.0.1 AA:     </pre>

- (1) If the transition condition (%M0) is on, LT function will be executed.
- (2) If input variable VALUE1 = 100, VALUE2 = 200, and VALUE3 = 300, output %Q0.0.1 = 1.

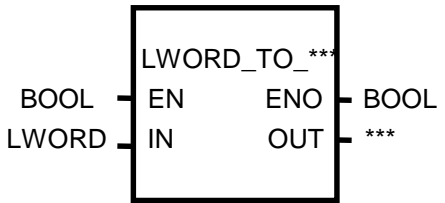
Input (IN1): VALUE1 (INT) = 100 (16#0064)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td> </tr> <tr> <td colspan="16" style="text-align: center;">&lt; (LT)</td> </tr> </table>	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	< (LT)															
0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0																		
< (LT)																																		
(IN2): VALUE2 (INT) = 200 (16#00C8)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td> </tr> <tr> <td colspan="16" style="text-align: center;">&lt; (LT)</td> </tr> </table>	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	< (LT)															
0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0																		
< (LT)																																		
(IN3): VALUE3 (INT) = 300 (16#012C)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td> </tr> </table>	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0																
0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0																		
Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)	<div style="text-align: right; margin-right: 20px;">↓</div> <table border="1" style="width: 50px; height: 20px; margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="text-align: center;">1</td> </tr> </table>	1																																
1																																		

## 8. Basic Function/Function Block Library

### LWORD\_TO\_\*\*\*

LWORD type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: bit string to convert (64bit)</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

It converts input IN type and produces output OUT.

Function	Output type	Description
LWORD_TO_SINT	SINT	Takes the lower 8 bits and converts into SINT type.
LWORD_TO_INT	INT	Takes the lower 16bits and converts into INT type.
LWORD_TO_DINT	DINT	Takes the lower 32bits and converts into DINT type.
LWORD_TO_LINT	LINT	Converts into LINT type without changing the internal bit array.
LWORD_TO_USINT	USINT	Takes the lower 8 bits and converts into USINT type.
LWORD_TO_UINT	UINT	Takes the lower 16 bits and converts into UINT type.
LWORD_TO_UDINT	UDINT	Takes the lower 32bits and converts into UDINT type.
LWORD_TO_ULINT	ULINT	Converts into ULINT type without changing the internal bit array.
LWORD_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
LWORD_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
LWORD_TO_WORD	WORD	Takes the lower 16 bits and converts into WORD type.
LWORD_TO_DWORD	DWORD	Takes the lower 32 bits and converts into DWORD type.
LWORD_TO_LREAL	LREAL	Converts LWORD into LREAL type.
LWORD_TO_DT	DT	Converts into DT type without changing the internal bit array.
LWORD_TO_STRING	STRING	Converts input value into STRING type.



## 8. Basic Function/Function Block Library

### MAX

Maximum value
---------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: the value to be compared            IN2: the value to compare            Input variable number can be extended up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: maximum value among input</p> <p>IN1, IN2,..., OUT should be the same data type.</p>

#### ■ Function

It produces the maximum value among input IN1, IN2,..., INn (n: input number).

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   GG LD      VALUE1 MAX     IN1:= CURRENT RESULT         IN2:= VALUE2 ST      OUT_VALUE GG:</pre>

(1) If the transition condition (%M0) is on, MAX function will be executed.

(2) As the result of comparing input variable (VALUE1 = 100 and VALUE2 = 200), maximum value is 200.  
 Output OUT\_VAL will be 200.

Input (IN1): VALUE1 (INT) = 100 (16#0064)

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
(MAX)																

(IN2): VALUE2 (INT) = 200 (16#00C8)

0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↓

Output (OUT): OUT\_VAL (INT) = 200 (16#00C8)

0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

**MID**

Takes the middle part of a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            IN: input character string            L: the length of character string to output            P: starting location of character string to output</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1.            OUT: output character string</p>

■ **Function**

It produces a character string (L) of IN from the P character.

■ **Error**

If (character number of variable IN) < P, P <= 0 or L < 0, then \_ERR and \_LER flags will be set.

■ **Program Example**

LD	IL
	<pre> LD      %I0.0.0 JMPN   MM LD      IN_TEXT MID    IN:= CURRENT RESULT         L:= LENGTH         P:= POSITION ST      OUT_TEXT MM: </pre>

- (1) If the transition condition (%I0.0.0) is on, MID function will be executed.
- (2) If input character string IN\_TEXT = 'ABCDEFGH', the length of character string LENGTH = 3, and starting location of character starting POSITION = 2, output variable OUT\_TEXT = 'BCD'.

Input (IN): IN\_TEXT1 (STRING) = 'ABCDEFGH'  
 (L): LENGTH (INT) = 3  
 (P): POSITION (INT) = 2  
 ↓ (MID)  
 Output (OUT): OUT\_TEXT = 'BCD'

## 8. Basic Function/Function Block Library

### MIN

Minimum value	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1  IN1: value to be compared  IN2: value to compare  Input variable number can be extended up to 8</p> <p><b>Output</b> ENO: without an error, it will be 1  OUT: minimum value among input values</p> <p>IN1, IN2, ..., OUT should be all the same data type.</p>

#### ■ Function

Produces the minimum value among input IN1, IN2, ... , INn (n: input number).

#### ■ Program Example

LD	IL
	<pre> LD          %M100 JMPN       BBB LD          VALUE1 MIN        IN1:= CURRENT RESULT            IN2:= VALUE2 ST          OUT_VALUE BBB: </pre>

(1) If the transition condition (%M100) is ON, MIN function is executed.

(2) The output is OUT\_VALUE = 100 because its minimum value is 100 as the result of comparing VALUE1 = 100 to VALUE2 = 200.

Input (IN1): VALUE1 (INT) = 100 (16#0064)

0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
(MIN)																

(IN2): VALUE2 (INT) = 200 (16#00C8)

0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↓

Output (OUT): OUT\_VAL (INT) = 100 (16#0064)

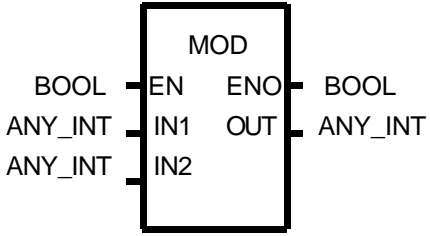
0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



# MOD

Dividing result (remainder)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

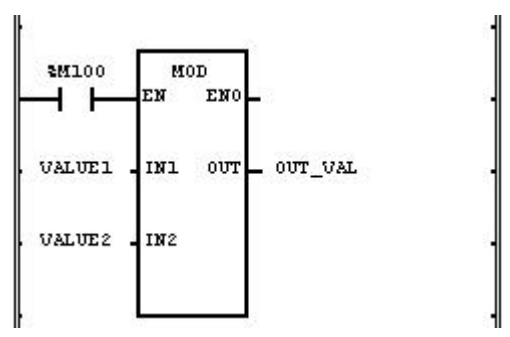
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: dividend IN2: divisor</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: dividing result (remainder)</p> <p>IN1, IN2, ..., OUT should be all the same data type.</p>

### Function

Divides IN1 by IN2 and outputs its remainder as OUT.  
 $OUT = IN1 - (IN1/IN2) \times IN2$  (if  $IN2 = 0$ ,  $OUT = 0$ )

IN1	IN2	OUT
7	2	1
7	-2	1
-7	2	-1
-7	-2	-1
7	0	0

### Program Example

LD	IL
	<pre> LD      %M100 JMPN   BB LD      VALUE1 MOD    IN1:= CURRENT RESULT        IN2:= VALUE2 ST      OUT_VAL BB:         </pre>

- (1) If the transition condition (%M100) is ON, MOD function is executed.
- (2) If the dividend VALUE1 = 37 and the divisor VALUE2 = 10, the remainder value OUT\_VAL is 7 as a result of dividing 37 by 10.

Input (IN1): VALUE1 (INT) = 37 (16#0025)

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

(IN2): VALUE2 (INT) = 10 (16#000A)

(MOD)

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

Output (OUT): OUT\_VAL (INT) = 7 (16#0007)

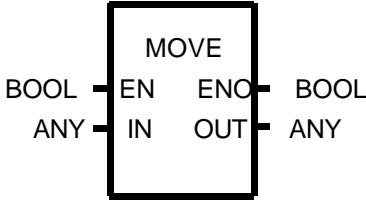
↓

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

## 8. Basic Function/Function Block Library

### MOVE

Data movement (Copy data)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

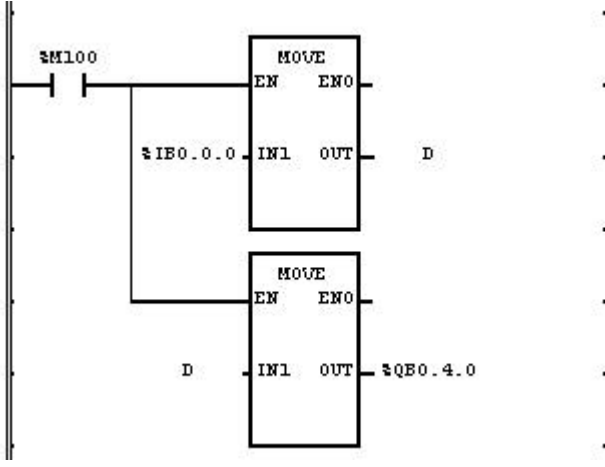
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: value to be moved</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: moved value</p> <p>Variables connected to IN and OUT are the same type.</p>

#### ■ Function

Moves an IN value to OUT.

#### ■ Program Example

This is a program that transfers the 8-contact inputs %I0.0.0 %I0.0.7 to the variable D and then moves them to output %Q0.4.0 %Q0.4.7.

LD	IL
	<pre> LD      %M100 JMPN   AAA LD      %IB0.0.0 MOVE ST      D LD      D MOVE ST      %QB0.4.0 AAA: </pre>

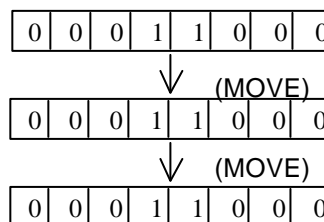
(1) If the transition condition (%M100) is ON, MOVE function is executed.

(2) It moves 8-contact input module data to the variable D by the first MOVE function and moves them to %Q0.4.0 %Q0.4.7.

Input (IN1): %IB0.0.0 (BYTE) = 16#18

D (BYTE) = 16#18

Output (OUT): %QB0.4.0 (BYTE) = 16#18



# MUL

Multiplication
----------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: multiplicand            IN2: multiplier            Input is available to extend up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1            OUT: multiplied value</p> <p>Variables connected to IN1, IN2, ..., OUT are all the same data type.</p>

■ **Function**

Multiplies an IN1, IN2,..., INn (n: input number) and outputs the result as OUT.

$$OUT = IN1 \times IN2 \times \dots \times INn$$

■ **Error**

If an output value is out of its data-type range, \_ERR and \_LER flags are set.

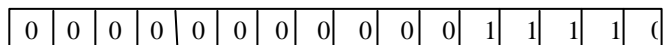
■ **Program Example**

LD	IL
	<pre>LD      %M0 JMPN   ABC LD      VALUE1 MUL    IN1:= CURRENT RESULT         IN2:= VALUE2         IN3:= VALUE3 ST      OUT_VAL ABC:</pre>

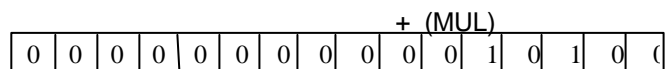
(1) If the transition condition (%M0) is ON, MUL function is executed.

(2) If input variables of MUL function, VALUE1 = 30, VALUE2 = 20, VALUE3 = 10, then the output variable OUT\_VAL = 30 × 20 × 10 = 6000.

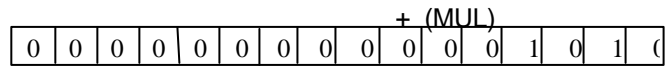
Input (IN1): VALUE1 (INT) = 30 (16#001E)



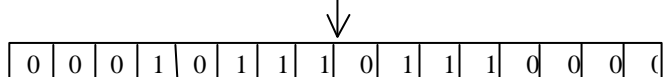
(IN2): VALUE2 (INT) = 20 (16#0014)



(IN3): VALUE3 (INT) = 10 (16#000A)



Output (OUT): OUT\_VAL (INT) = 6000 (16#1770)



## 8. Basic Function/Function Block Library

### MUL\_TIME

Time multiplication	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: time to be multiplied IN2: multiplying value</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: multiplied result</p>

#### ■ Function

Multiplies the IN1 (time) by IN2 (number) and outputs the result time as OUT.

#### ■ Error

If an output value is out of its TIME-data range, \_ERR and \_LER flags are set.

#### ■ Program Example

This is the program that sets the required working time: the average estimated time per unit product is 20min 2sec and the number of product to produce a day is 20 in one product line.

LD	IL
	<pre>LD      %M0 JMPN   ABC LD      UNIT_TIME MUL_TIME IN1:= CURRENT RESULT         IN2:= PRODUCT_COUNT ST      TOTAL_TIME ABC:</pre>

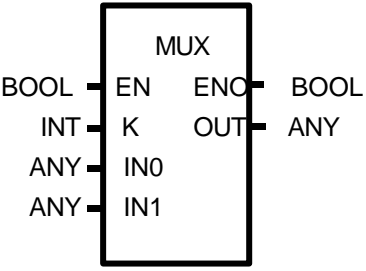
- (1) Write input variable (IN1: the estimated time per unit product) UNIT\_TIME: T#20M2S.
- (2) Write input variable (IN2: quantity of production) PRODUCT\_COUNT: 20.
- (3) Write TOTAL\_TIME to the output variable (OUT: total required working time).
- (4) If the transition condition (%M0) is on, T#6H40M40S will be produced in output TOTAL\_TIME.

```
Input (IN1): UNIT_TIME (TIME) = T#20MS2S
              (MUL_TIME)
(IN2): PRODUCT_COUNT (INT) = 16#18
              ↓
Output (OUT): TOTAL_TIME (TIME) = T#6H40M40S
```

## MUX

Selection from multiple inputs

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
 <p>The diagram shows a rectangular block labeled 'MUX'. On the left side, there are four input terminals: 'EN' (labeled 'BOOL'), 'K' (labeled 'INT'), 'IN0' (labeled 'ANY'), and 'IN1' (labeled 'ANY'). On the right side, there are two output terminals: 'ENO' (labeled 'BOOL') and 'OUT' (labeled 'ANY').</p>	<p><b>Input</b> EN: executes the function in case of 1            K: selection            IN0: the value to be selected            IN1: the value to be selected            Input variable number can be extended up to 8</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: the selected value</p> <p>IN0, IN1, ..., OUT should be the same time.</p>

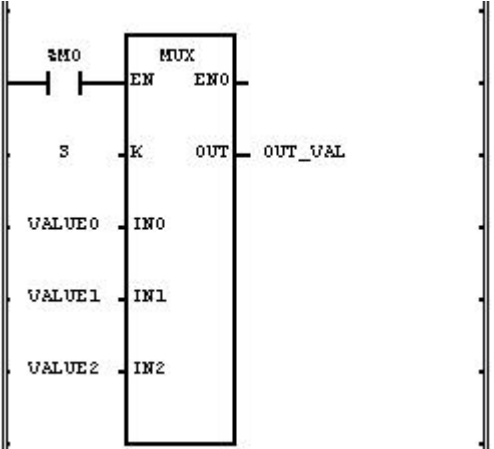
### ■ Function

Selects one among several inputs (IN0, IN1, ..., INn) with K value and produces it.  
 If K = 0, IN0 will be an output; if K = 1, IN1 will be an output; if K = n, INn will be an output.

### ■ Error

If K is greater than or equal to the number of input variable INn, then IN0 will be an output and \_ERR, \_LER flags will be set.

### ■ Program Example

LD	IL
 <p>The diagram shows a ladder logic network. The first rung consists of a normally open contact labeled '%M0' in series with a coil labeled 'MUX'. The second rung consists of a normally open contact labeled 'S' in series with a coil labeled 'OUT_VAL'. The 'MUX' block has inputs 'EN', 'K', 'IN0', and 'IN1', and outputs 'ENO' and 'OUT'. 'EN' is connected to '%M0', 'K' is connected to 'S', 'IN0' is connected to 'VALUE0', 'IN1' is connected to 'VALUE1', and 'IN2' is connected to 'VALUE2'. 'ENO' is connected to 'OUT_VAL'.</p>	<pre> LD      %M0 JMPN   ABC LD      S MUX    K:=  CURRENT RESULT         IN0:=  VALUE0         IN1:=  VALUE1         IN2:=  VALUE2  ST      OUT_VAL ABC: </pre>

- (1) If the transition condition (%M0) is on, MUX function will be executed.
- (2) Input variable is selected by selection variable S and is moved to OUT.

Input (K): S (INT) = 2

(IN0): VALUE0 (WORD) = 16#11

(IN1): VALUE1 (WORD) = 16#22

(IN2): VALUE2 (WORD) = 16#33

↓ (MUX)

Output (OUT): OUT\_VAL (WORD) = 16#33

## 8. Basic Function/Function Block Library

### NE

' Not equal to' comparison	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: The value to be compared            IN2: The value to be compared            IN1, IN2 should be the same data type.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: the compared result value</p>

#### ■ Function

If IN1 is not equal to IN2, output OUT will be 1.

If IN1 is equal to IN2, output OUT will be 0.

#### ■ Program Example

LD	IL
	<pre> LD      %I0.0.0 JMPN   PP LD      VALUE1 NE     IN1:= CURRENT RESULT        IN2:= VALUE2 ST      %Q0.0.1 PP:           </pre>

(1) If the transition condition (%I0.0.0) is on, NE function will be executed.

(2) If input variable VALUE1 = 300, VALUE2 = 200 (the compared result VALUE1 and VALUE2 are different), output result value will be %Q0.0.1 = 1.

Input (IN1): VALUE1 (INT) = 300 (16#012C)

0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

(IN2): VALUE2 (INT) = 200 (16#0C8)

(NE)																
0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0

Output (OUT): %Q0.0.1 (BOOL) = 1 (16#1)

↓
1

# NOT

Reverse Logic (Logic inversion)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: the value to be logically inverted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: the inversed (NOT) value</p> <p>IN, OUT should be the same data type.</p>

### ■ Function

It inverts the IN (by bit) and produces output OUT.

IN 1100 ..... 1010  
OUT 0011 ..... 0101

### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      %MB10 NOT     IN:= CURRENT RESULT ST      %QB0.0.0 AAA:</pre>

(1) If the transition condition (%M0) is on, NOT function will be executed.

(2) If NOT function is executed, input data value of %MB10 will be inversed and will be written in %QB0.0.0.

Input (IN1): %MB10 (BYTE) = 16#CC      

1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

↓ (NOT)

Output (OUT): %QB0.0.0 (BYTE) = 16#33      

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

## 8. Basic Function/Function Block Library

### NUM\_TO\_STRING

Converts number to a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input data to be converted to STRING</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: converted data (character)</p>

#### ■ Function

It converts the numeric data of IN to the character data and produces output OUT.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      IN_VALUE NUM_TO_STRING ST      OUT_STRING</pre>

(1) If the transition condition (%M0) is ON, function NUM\_TO\_STRING will be executed.

(2) If IN\_VALUE (INT) = 123, OUT\_STRING will be '123'; if IN\_VALUE (REAL) = 123.0, OUT\_STRING will be '1.23E2'.

Input (IN1): IN\_VALUE (INT) = 123

↓ (NUM\_TO\_STRING)

Output (OUT): OUT\_STRING (STRING) = '123'



# OR

Logical OR
------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: input 1            IN2: input 2            Input variables can be extended up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: OR result</p> <p>IN1, IN2, OUT should be all the same data type.</p>

## ■ Function

It performs a logical OR on the input variables by bit and produces output OUT.

```

IN1 1111 ..... 0000
OR
IN2 1010 ..... 1010
OUT 1111 ..... 1010
    
```

## ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   AAA LD      %MB10 OR     IN1:= CURRENT RESULT        IN2:= ABC ST      %QB0.0.0     </pre>

- (1) If the transition condition (%M0) is on, function OR will be executed.
- (2) The result of a logic sum (OR) for %MB10 = 11001100 and ABC = 11110000 will be produced in %QB0.0.0 = 11111100.

Input (IN1): %MB10 (BYTE) = 16#CC

1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

Logical OR operation

(IN2): ABC (BYTE) = 16#F0

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---



Output (OUT): %QB0.0.0 (BYTE) = 16#FC

1	1	1	1	1	1	0	0
---	---	---	---	---	---	---	---

## 8. Basic Function/Function Block Library

### REAL\_TO\_\*\*\*

REAL type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: the REAL value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

#### ■ Function

It converts the IN type and outputs it as OUT.

Function	Output type	Description
REAL_TO_SINT	SINT	If integer part of input is -128 127, normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_INT	INT	If integer part of input is -32768 32767, normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_DINT	DINT	If integer part of input is $-2^{31}$ $2^{31}-1$ , normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_LINT	LINT	If integer part of input is $-2^{63}$ $2^{63}-1$ , normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_USINT	USINT	If integer part of input is 0 255, normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_UINT	UINT	If integer part of input is 0 65,535, normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_UDINT	UDINT	If integer part of input is 0 $2^{32}-1$ , normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_ULINT	ULINT	If integer part of input is 0 $2^{64}-1$ , normal conversion. Otherwise an error occurs. (Decimals round-off)
REAL_TO_DWORD	DWORD	Converts into DWORD type without changing the internal bit array.
REAL_TO_LREAL	LREAL	Converts REAL into LREAL type normally.

#### ■ Error

If overflow occurs (an input value is greater than the value to be stored in output type), \_ERR, \_LER flags will be set. If an error occurs, the output will be 0.

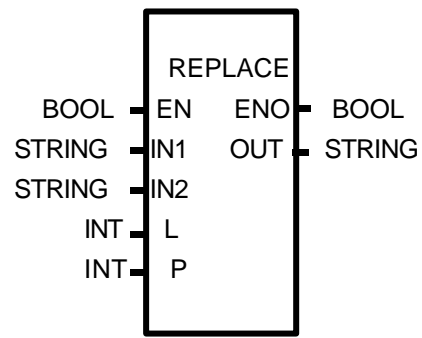


## 8. Basic Function/Function Block Library

### REPLACE

Replace a string (Character string replacement)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            IN1: character string to be replaced            IN2: character string to replace            L: the length of character string to be replaced            P: position of character string to be replaced</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1.            OUT: output character string</p>

#### ■ Function

Its function is to remove the L-length character from IN1 (starting from P) and put IN2 in the removed position as output OUT.

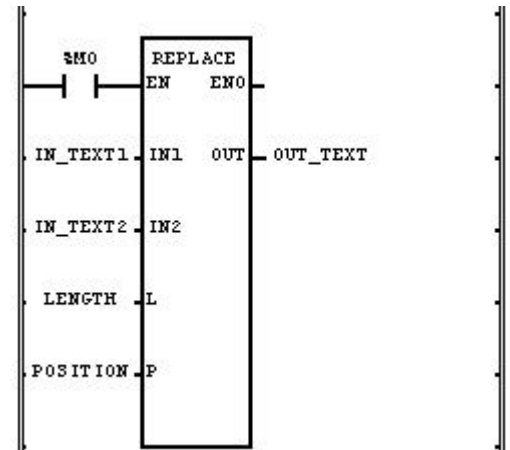
#### ■ Error

\_ERR, \_LER flags will be set if:

$P \leq 0$  or  $L < 0$

$P > (\text{input character number of IN1})$   
 character number of result  $> 30$

#### ■ Program Example

LD	IL
	<pre> LD          %M0 JMPN       MBC LD          IN_TEXT1 REPLACE    IN1:= CURRENT RESULT            IN2:= IN_TEXT2            L:= LENGTH            P:= POSITION ST          OUT_TEXT ABC: </pre>

## 8. Basic Function/Function Block Library

---

- (1) If the transition condition (%M0) is ON, function REPLACE (character string replacement) will be executed.
- (2) If input variable of character string to be replaced IN\_TEXT1 = `ABCDEF`, input variable of character string to replace N\_TEXT2 = `X`, input variable of character string length to be replaced LENGTH = 3 and input variable of character string position designation to be replaced POSITION = 2, then `BCD` of IN\_TEXT will be replaced with `X` of IN\_TEXT2 and output variable OUT\_TEXT will be `AXET` .

Input (IN1): IN\_TEXT1 (STRING) = `ABCDEF`  
(IN2): IN\_TEXT2 (STRING) = `X`  
(L): LENGTH (INT) = 3  
(P): POSITION (INT) = 2



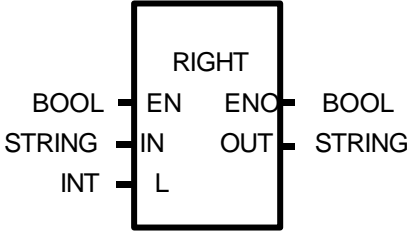
Output (OUT): OUT\_TEXT (STRING) = `AXET`

## 8. Basic Function/Function Block Library

### RIGHT

To take the right of character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: If EN is 1, function executes. IN: input character string L: length of character string</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: output character string</p>

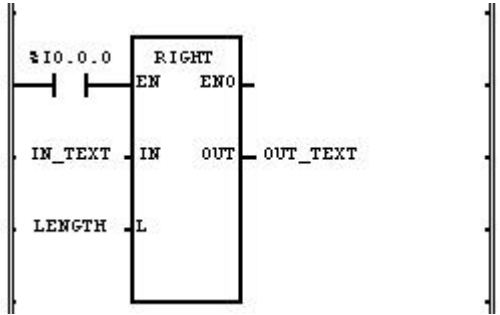
#### ■ Function

It takes a right L-length character string of IN and produces output OUT.

#### ■ Error

If  $L < 0$ , `_ERR` and `_LER` flags will be set.

#### ■ Program Example

	IL
	<pre> LD      %I0.0.0 JMPN   AAA LD      IN_TEXT RIGHT  IN:= CURRENT RESULT         L: = LENGTH ST      OUT_TEXT AAA: </pre>

- (1) If the transition condition (`%I0.0.0`) is on, function RIGHT (to take the right of character string) will be executed.
- (2) If character string declared as input variable `IN_TEXT = `ABCDEFG`` and the length of character string to output `LENGTH = 3`, output character string variable `OUT_TEXT = `EFG``.

Input (IN1): `IN_TEXT (STRRING) = `ABCDEFG``

(L): `LENGTH (INT) = 3`

↓ (RIGHT)

Output (OUT): `OUT_TEXT (STRRING) = `EFG``

## ROL

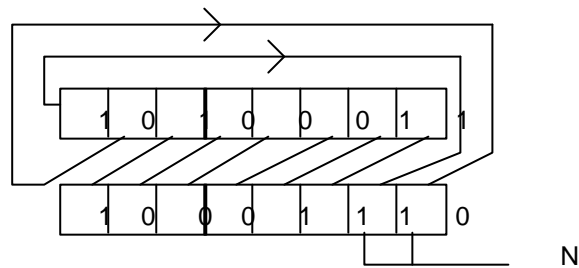
Rotate to left

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: the value to be rotated N: bit number to rotate</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: the rotated value</p>

### ■ Function

It rotates input IN to the left as many as N bit number.



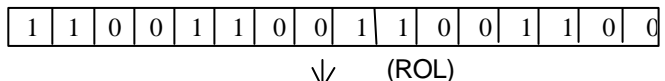
### ■ Program Example

This is the program that rotates the value of input data (1100\_1100\_1100\_1100:16#CCCC) to the left by 3 bits if input %I0.0.0 is on.

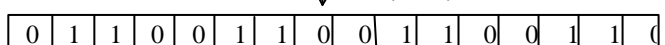
LD	IL
	<pre>LD      %I0.0.0 JMPN   PPP LD      IN_VALUE ROL     IN:= CURRENT RESULT         N:= 3 ST      OUT_VALUE PPP:</pre>

- (1) Set input variable IN\_VALUE to rotate.
- (2) Set the value to be rotated (3).
- (3) Set output variable to output the rotated data value as OUT\_VALUE.
- (4) If the transition condition (%I0.0.0) is ON, function ROL will be executed and a data bit set as input variable will be rotated to the left by 3 bits and produces output OUT\_VALUE.

Input (IN1): IN\_VALUE (WORD) = 16#CCCC  
(N): 3



Output (OUT): OUT\_VALUE (WORD) = 16#6666



## 8. Basic Function/Function Block Library

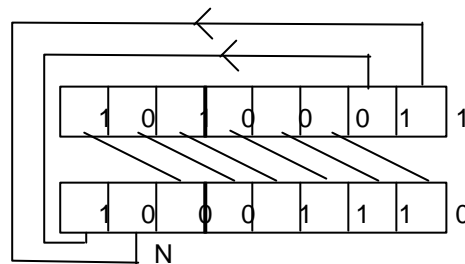
### ROR

Rotate to right	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
<p>The diagram shows a rectangular function block labeled 'ROR'. On the left side, there are three input terminals: 'EN' (labeled 'BOOL'), 'IN' (labeled 'ANY_BIT'), and 'INT' (labeled 'INT'). On the right side, there are two output terminals: 'ENO' (labeled 'BOOL') and 'OUT' (labeled 'ANY_BIT').</p>	<p><b>Input</b> EN: executes the function in case of 1 IN: the value to be rotated N: bit number to rotate</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the rotated value</p>

#### ■ Function

It rotates input IN to the right as many as N bit number.



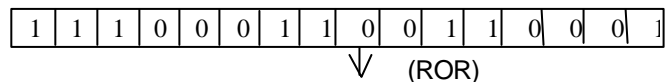
#### ■ Program Example

This is the program that rotates input data value (1110001100110001: 16#E331) to the right by 3 bits if input %I0.0.0 is ON.

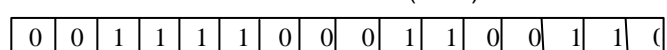
LD	IL
<p>The diagram shows a single normally open contact labeled '%I0.0.0' connected to the 'EN' terminal of the 'ROR' function block. The 'IN' terminal is connected to 'IN_VALUE1'. The 'INT' terminal is connected to the constant value '3'. The 'ENO' terminal is connected to 'ENO'. The 'OUT' terminal is connected to 'OUT_VALUE'.</p>	<pre> LD      %I0.0.0 JMPN   PO LD      IN_VALUE1 ROR     IN1:= CURRENT RESULT         N:= 3 ST      OUT_VALUE PO </pre>

- (1) Set input variable of a data value to rotate as IN\_VALUE1.
- (2) Insert bit number 3 into bit number input N.
- (4) If the transition condition (%I0.0.0) is ON, function ROR (rotate Right) will be executed and data bit set as input variable will be rotated to the right by 3 bits and produces output OUT\_VALUE.

Input (IN1): IN\_VALUE1 (WORD) = 16#E331  
(N): 3



Output (OUT): OUT\_VALUE(WORD) = 16#3C66





## SEL

Selection from two inputs

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
<p>The diagram shows a rectangular block labeled 'SEL'. On the left side, there are four input terminals: 'EN' (labeled 'BOOL'), 'G' (labeled 'BOOL'), 'IN0' (labeled 'ANY'), and 'IN1' (labeled 'ANY'). On the right side, there are three output terminals: 'ENO' (labeled 'BOOL'), 'OUT' (labeled 'ANY'), and 'ANY' (labeled 'ANY').</p>	<p><b>Input</b> EN: executes the function in case of 1  G: selection  IN0: the value to be selected  IN1: the value to be selected</p> <p><b>Output</b> ENO: without an error, it will be 1  OUT: the selected value</p> <p>IN1, IN2, OUT should be all the same type.</p>

### ■ Function

If G is 0, IN0 will be an output and if G is 1, IN1 will be an output.

### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   PPP LD      S SEL     G:= CURRENT RESULT         IN1:= VALUE1         IN2:= VALUE2 ST      %QW0.0.0 PPP: </pre>

(1) If the transition condition (%M0) is ON, function SEL will be executed.

(2) If S = 1 and VALUE1 = 16#1110, VALUE2 = 16#FF00, then output variable %QW0.0.0 = 16#FF00.

Input (G): S = 1

(IN0): VALUE1 (WORD) = 16#1110

(IN1): VALUE2(WORD) = 16#FF00

↓ (SEL)

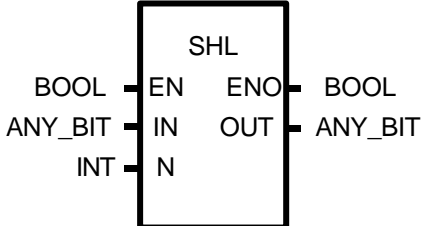
Output (OUT): %QW0.0.0 (WORD) = 16#FF00

## 8. Basic Function/Function Block Library

### SHL

Shift Left
------------

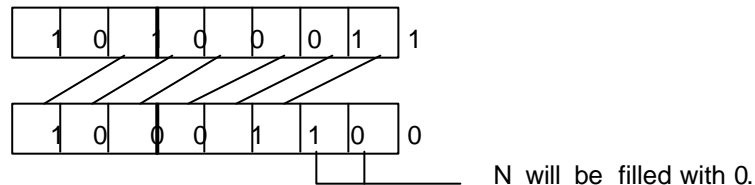
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
 <p>The diagram shows a rectangular function block labeled 'SHL'. On the left side, there are three input terminals: 'EN' (labeled 'BOOL'), 'IN' (labeled 'ANY_BIT'), and 'N' (labeled 'INT'). On the right side, there are two output terminals: 'ENO' (labeled 'BOOL') and 'OUT' (labeled 'ANY_BIT').</p>	<p><b>Input</b> EN: If EN is 1, function is executed. IN: bit string to be shifted N: bit number to be shifted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: the shifted value</p>

#### ■ Function

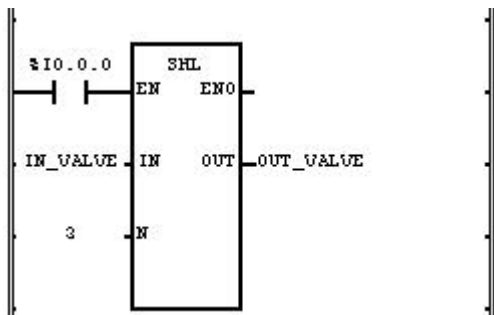
It shifts input IN to the left as many as N bit number.

N number bit on the rightmost of input IN will be filled with 0.



#### ■ Program Example

This is the program that shifts input data value (1100\_1100\_1100\_1100:16#CCCC) to the left by 3 bits if input %I0.0.0 is ON.

LD	IL
 <p>The ladder logic diagram shows a normally open contact labeled '%I0.0.0' connected to the 'EN' terminal of the 'SHL' function block. The 'IN' terminal is connected to a variable 'IN_VALUE'. The 'N' terminal is connected to the constant value '3'. The 'OUT' terminal is connected to a variable 'OUT_VALUE'. The 'ENO' terminal is connected to the power rail.</p>	<pre>LD      %I0.0.0 JMPN   ABC LD      IN_VALUE SHL    IN:= CURRENT RESULT         N:= 3 ST      OUT_VALUE ABC:</pre>

(1) Set the input variable IN\_VALUE (11001110:16#CE).

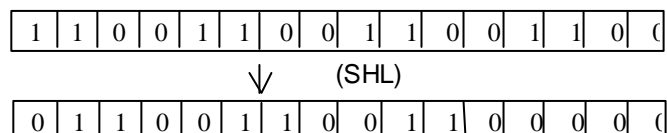
(2) Insert bit number 3 into N.

(3) If the transition condition (%I0.0.0) is ON, function SHL (shift Left) will be executed and data bit set as input variable shifts to the left by 3 bits and produces output OUT\_VALUE.

Input (IN1): IN\_VALUE (WORD) = 16#CCCC

(N): 3

Output (OUT): OUT\_VALUE (WORD) = 16#6660



**SHR**

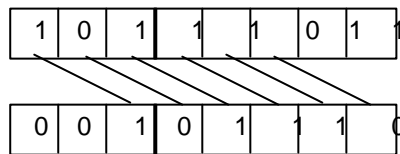
Shift Right

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
<p>The diagram shows a rectangular function block labeled 'SHR'. On the left side, there are three input terminals: 'EN' (labeled 'BOOL'), 'IN' (labeled 'ANY_BIT'), and 'INT' (labeled 'INT'). On the right side, there are three output terminals: 'ENO' (labeled 'BOOL'), 'OUT' (labeled 'ANY_BIT'), and 'N' (labeled 'INT').</p>	<p><b>Input</b> EN: executes the function in case of 1 IN: bit string to be shifted N: bit number to be shifted</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the shifted value</p>

■ **Function**

It shifts input IN to the right as many as N bit number.  
N number bit on the leftmost of input IN will be filled with 0.



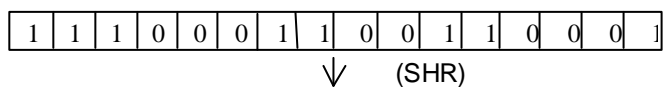
N will be filled with 0.

■ **Program Example**

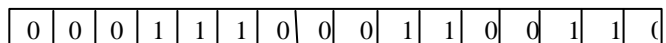
LD	IL
	<pre>LD      %M0 JMPN   AAA LD      IN_VALUE SHR    IN:= CURRENT RESULT         N:= SHIFT_NUM ST      OUT_VALUE</pre>

- (1) If the transition condition (%M0) is on, function SHL (Shift Left) will be executed.
- (2) Data bit set as input variable shift to the right by 3 bits and produces outputs OUT\_VALUE.

Input (IN1): IN\_VALUE (WORD) = 16#E331  
(N): 3



Output (OUT): OUT\_VALUE (WORD) = 16#1C66



## 8. Basic Function/Function Block Library

### SIN

Sine operation	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of Sine operation (radian)</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: Sine operation result value</p> <p>IN, OUT should be the same data type.</p>

#### ■ Function

Finds the Sine operation value of IN and produces output OUT.

OUT = SIN (IN)

#### ■ Program Example

LD	IL
	<pre>LD      %I0.0.0 JMPN   PPP LD      INPUT SIN ST      RESULT PPP:</pre>

(1) If the transition condition (%I0.0.0) is ON, function SIN (Sine operation) will be executed.

(2) If the value of input variable INPUT is 1.0471 .... ( $\pi/3$  rad =  $60^\circ$ ), RESULT declared as output variable will be 0.8660 .... ( $\sqrt{3}/2$ ).

$$\text{SIN } (\pi/3) = \sqrt{3}/2 = 0.8660$$

Input (IN1): INPUT (REAL) = 1.0471

↓ (SIN)

Output (OUT): RESULT (REAL) = 8.65976572E-01

**SINT\_TO\_\*\*\***

SINT type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: short Integer value</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
SINT_TO_INT	INT	Converts into INT type normally.
SINT_TO_DINT	DINT	Converts into DINT type normally.
SINT_TO_LINT	LINT	Converts into LINT type normally.
SINT_TO_USINT	USINT	If input is 0 ~ 127, normal conversion. Otherwise an error occurs.
SINT_TO_UINT	UINT	If input is 0 ~ 127, normal conversion. Otherwise an error occurs.
SINT_TO_UDINT	UDINT	If input is 0 ~ 127, normal conversion. Otherwise an error occurs.
SINT_TO_ULINT	ULINT	If input is 0 ~ 127, normal conversion. Otherwise an error occurs.
SINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
SINT_TO_BYTE	BYTE	Converts into BYTE type without changing the internal bit array.
SINT_TO_WORD	WORD	Converts into WORD type filling the upper bits with 0.
SINT_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
SINT_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
SINT_TO_BCD	BYTE	If input is 0 ~ 99, normal conversion. Otherwise an error occurs.
SINT_TO_REAL	REAL	Converts SINT into REAL type normally.
SINT_TO_LREAL	LREAL	Converts SINT into LREAL type normally.

■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set. If an error occurs, take the lower bits as many as bit number of output type and output it without changing the internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   AAA LD      IN_VAL SINT_TO_BCD ST      BCD_VAL AAA: </pre>

(1) If the input condition (% M0) is ON, function SINT\_TO\_BCD will be executed.

(2) If input variable IN\_VAL (SINT) = 64 (2#0100\_0000), output variable OUT\_VAL (BCD type) = 16#64 (2#0110\_0100).

Input (IN1): IN\_VAL(SINT) = 64(16#40)

0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---

↓ (SINT\_TO\_BCD)

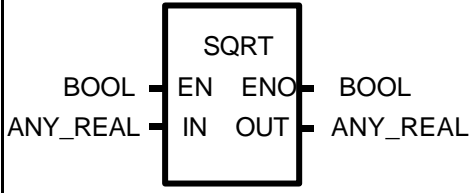
Output (OUT): OUT\_VAL(BCD) = 16#64(16#64)

0	1	1	0	0	1	0	0
---	---	---	---	---	---	---	---

### SQRT

Calculate SQRT (Square root operation)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: input value of square root operation</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: square root value IN, OUT should be the same data type.</p>

■ **Function**

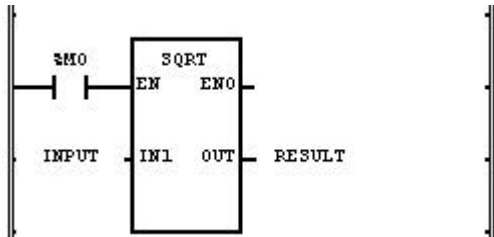
It finds the square root value of IN and output it as OUT.

$$OUT = \sqrt{IN}$$

■ **Error**

If the value of IN is a negative number, \_ERR and \_LER flag will be set.

■ **Program Example**

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      INPUT SQRT ST      RESULT AAA:</pre>

(1) If the transition condition (%M0) is ON, function SQRT (square root operation) will be executed.

(2) If the value of input variable declared as INPUT is 9.0, RESULT declared as output variable will be 3.0.

$$\sqrt{9.0} = 3.0$$

Input (IN1): INPUT (REAL) = 9.0

↓ (SQRT)

Output (OUT): RESULT (REAL) = 3.0

## 8. Basic Function/Function Block Library

### STOP

Stop running by program	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 RE: requires the operation stop by program</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: If STOP function is executes, it will be 1.</p>

#### ■ Function

If EN and REQ are 1, stop running and return to STOP mode.

If function 'STOP' is executed, the program stops after completing scan program in executing.

Program restarts in case of power re-supply or the change of operation mode from STOP to RUN.

#### ■ Program Example

LD	IL
	<pre>LD      %I0.0.0 JMPN   PT LD      LOG_OUT STOP ST      SHUT_OFF PT:</pre>

(1) If the transition condition (%I0.0.0) and LOG\_OUT is 1, it becomes to STOP mode after completing the scan program in executing.

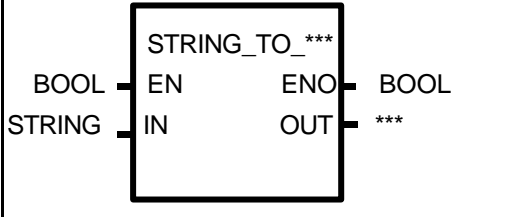
(2) It is recommended to turn off the power of PLC in the stable state after executing 'STOP' function declared as input variable.



**STRING\_TO\_\*\*\***

STRING type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: If EN is 1, function converts. IN: character string</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: type-converted data</p>

■ **Function**

Converts the IN type and outputs it as OUT.

Function	Output type	Description
STRING_TO_SINT	SINT	Converts STRING into SINT type.
STRING_TO_INT	INT	Converts STRING into INT type.
STRING_TO_DINT	DINT	Converts STRING into DINT type.
STRING_TO_LINT	LINT	Converts STRING into LINT type.
STRING_TO_USINT	USINT	Converts STRING into USINT type.
STRING_TO_UINT	UINT	Converts STRING into UINT type.
STRING_TO_UDINT	UDINT	Converts STRING into UDINT type.
STRING_TO_ULINT	ULINT	Converts STRING into ULINT type.
STRING_TO_BOOL	BOOL	Converts STRING into BOOL type.
STRING_TO_BYTE	BYTE	Converts STRING into BYTE type.
STRING_TO_WORD	WORD	Converts STRING into WORD type.
STRING_TO_DWORD	DWORD	Converts STRING into DWORD type.
STRING_TO_LWORD	LWORD	Converts STRING into LWORD type.
STRING_TO_REAL	REAL	Converts STRING into REAL type.
STRING_TO_LREAL	LREAL	Converts STRING into LREAL type.
STRING_TO_DT	DT	Converts STRING into DT type.
STRING_TO_DATE	DATE	Converts STRING into DATE type.
STRING_TO_TOD	TOD	Converts STRING into TOD type.
STRING_TO_TIME	TIME	Converts STRING into TIME type.

■ **Error**

If input character type does not match with output data type, \_ERR and \_LER flags will be set.

## 8. Basic Function/Function Block Library

### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   ZZ LD      IN_VAL STRING_TO_REAL ST      OUT_VAL ZZ:</pre>

(1) If the input condition (%M0) is ON, function STRING\_TO\_REAL will be executed.

(2) If input variable IN\_VAL (STRING) = '-1.34E12', output variable OUT\_VAL (REAL) = -1.34E12.

Input (IN1): IN\_VAL (STRING) = '-1.34E12'

↓ (STRING\_TO\_REAL)

Output (OUT): OUT\_VAL (REAL) = -1.34E12

## 8. Basic Function/Function Block Library

### STRING\_TO\_ARY

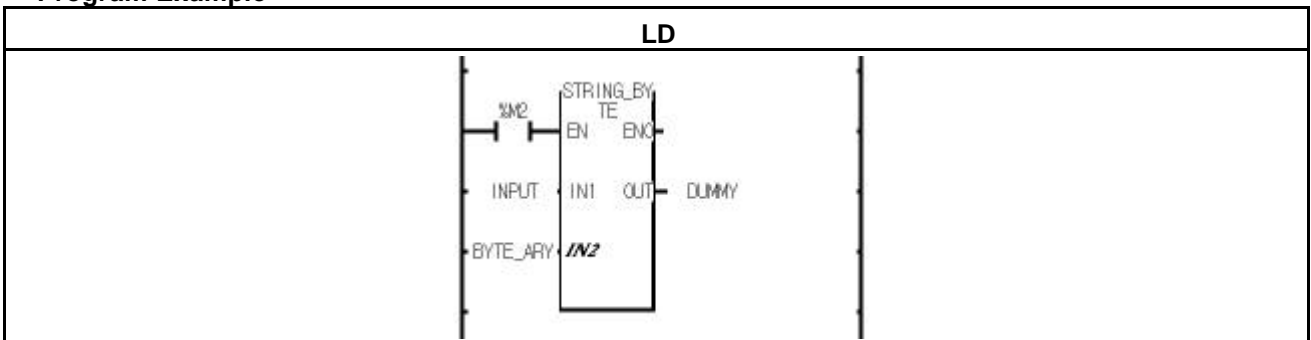
Convert a string into a byte array	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: If EN is 1, function converts. IN: string input</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: dummy output</p> <p><b>In/Out</b> IN2: converted byte array output</p>

#### ■ Function

It converts a string into 30 byte arrays.

#### ■ Program Example



(1) If the transition condition (%M2) is on, STRING\_BYTE function is executed.

(2) If input variable INPUT is "GM4-CPUA", In/Out variable BYTE\_ARY is as follows:

16#{22("), 47(G), 4D(M), 34(4), 2D(-), 43(C), 50(P), 55(U), 41(A), 22(")}.

## 8. Basic Function/Function Block Library

### SUB

Subtraction	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: the value to be subtracted IN2: the value to subtract</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the subtracted result value</p> <p>The variables connected to IN1, IN2 and OUT should be all the same data type.</p>

#### ■ Function

It subtracts IN2 from IN1 and outputs it as OUT.  
 $OUT = IN1 - IN2$

#### ■ Error

If output value is out of range of related data type, `_ERR` and `_LER` flags will be set.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      VALUE1 SUB    IN1:= CURRENT RESULT         IN2:= VALUE2 ST      OUT_VAL AAA:</pre>

(1) If the transition condition (%M0) is ON, function SUB will be executed.

(2) If input variables VALUE1 = 300, VALUE2 = 200, OUT\_VAL will be 100 after operation.

Input (IN1): VALUE1 (INT) = 300 (16#012C)

0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0
- (SUB)																

(IN2): VALUE2 (INT) = 200 (16#00C8)

0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output (OUT): OUT\_VAL (INT) = 100 (16#0064)

↓																
0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0

## SUB\_DATE

Date subtraction	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
<p>The diagram shows a rectangular function block labeled 'SUB_DATE'. On the left side, there are three input terminals: 'EN' (connected to a 'BOOL' label), 'IN1' (connected to a 'DATE' label), and 'IN2' (connected to a 'DATE' label). On the right side, there are two output terminals: 'ENO' (connected to a 'BOOL' label) and 'OUT' (connected to a 'TIME' label).</p>	<p><b>Input</b> EN: executes the function in case of 1  IN1: standard date  IN2: the date to subtract</p> <p><b>Output</b> ENO: without an error, it will be 1.  OUT: produces the difference between two dates as time data.</p>

### ■ Function

It subtracts IN2 (specific date) from IN1(standard date) and outputs the difference between two dates as OUT.

### ■ Error

If output value is out of range (TIME data type), \_ERR and \_LER flags will be set.

An error occurs: 1) when date difference exceeds the range of TIME data type (T#49D17H2M47S295MS); 2) the result of date operation is a negative number.

### ■ Program Example

LD	IL
<p>The diagram shows a single normally open contact labeled '%I0.0.0' connected to the EN input of a SUB_DATE function block. The IN1 input is connected to 'CURRENT_DATE' and the IN2 input is connected to 'START_DATE'. The ENO output is connected to 'WORK_DAY'.</p>	<pre> LD          %I0.0.0 JMPN       PPP LD          CURRENT_DATE SUB_DATE   IN1:= CURRENT_DATE            IN2:= START_DATE ST          WORK_DAY PPP: </pre>

(1) If the transition condition (%I0.0.0) is ON, function SUB\_DATE will be executed.

(2) If input variable CURRENT\_DATE is D#1995-12-15 and START\_DATE is D#1995-11-1, the working days declared as output variable WORK\_DAY will be T#44D.

Input (IN1): CURRENT\_DATE (DATE) = D#1995-12-15  
(SUB\_DATE)  
(IN2): START\_DATE (DATE) = D#1995-11-1

Output (OUT): WORK\_DAY (TIME) = T#44D    ↓

## 8. Basic Function/Function Block Library

### SUB\_DT

Date and Time subtraction	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: standard date and time of day IN2: date and time of day to subtract</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the subtracted result time</p>

#### ■ Function

It subtracts IN2 (specific date and time of day) from IN1 (standard date and time of day) and outputs the time difference as OUT.

#### ■ Error

If output value is out of range of TIME data type, \_ERR and \_LER flags will be set.

If the result of date and time of day subtraction operation is a negative number, an error occurs.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   PPP LD      CURRENT_DT SUB_DT  IN1:= CURRENT_RESULT         IN2:= START_DT ST      WORK_TIME PPP:</pre>

(1) If the transition condition (%M0) is ON, function SUB\_DT (Time and Date subtraction) will be executed.

(2) If the current date and time of day CURRENT\_DT is DT#1995-12-15-14:30:00 and the starting date and the time of day to work START\_DT is DT#1995-12-13-12:00:00, the continuous working time declared as output variable WORK\_TIME will be T#2D2H30M.

Input (IN1): CURRENT\_DT (DT) = DT#1995-12-15-14:30:00  
(SUB\_DATE)

(IN2): START\_DT (DT) = DT#1995-12-13-12:00:00



Output (OUT): WORK\_TIME (TIME) = T#2D2H30M

## SUB\_TIME

Time subtraction	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN1: standard time of day IN2: the time to subtract</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the subtracted result time or time of day OUT data type is the same as the input IN1 type. That is, if IN1 type is TIME, OUT type should be TIME.</p>

### Function

If IN1 is TIME, it subtracts the time from the standard time and produces OUT (time difference).

If IN1 is TIME\_OF\_DAY, it subtracts the time from the standard time of day and outputs the time of a day as OUT.

If IN1 is DATE\_AND\_TIME, it subtracts the time from the standard date and the time of day and produces the date and the time of day as OUT.

### Error

If the output value is out of range of related data type, \_ERR and \_LER flags will be set.

If the result subtracting the time from the standard time is a negative number or the result subtracting the time from the time of day is a negative number, an error occurs.

### Program Example

LD	IL
	<pre> LD          %I0.0.0 JMPN       AAA LD          TARGET_TIME SUB_TIME   IN1:= CURRENT RESULT            IN2:= ELAPSED_TIME  ST          TIME_TO_GO AAA: </pre>

(1) If the transition condition (%I0.0.0) is ON, function SUB\_TIME (time subtraction) will be executed.

(2) If total working time declared as input variable TARGET\_TIME is T#2H30M, the elapsed time ELAPSED\_TIME is T#1H10M30S300MS, the remaining working time declared as output variable TIME\_TO\_GO will be T#1H19M29S700MS.

Input (IN1): TARGET\_TIME (TIME) = T#2H30M

(SUB\_TIME)

(IN2): ELAPSED\_TIME (TIME) = T#1H10M30S300MS



Output (OUT): TIME\_TO\_GO (TIME) = T#1H19M29S700MS

## 8. Basic Function/Function Block Library

### SUB\_TOD

TOD Subtraction	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: standard time of day            IN2: the time of day to subtract</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: the subtracted result time</p>

#### ■ Function

It subtracts the IN2 (specific time of day) from IN1 (standard time of day) and outputs the time difference as OUT.

#### ■ Error

If the result subtracting the time of day from the time of day is a negative number, an error occurs.

#### ■ Program Example

LD	IL
	<pre>LD      %I0.0.0 JMPN   AAA LD      END_TIME SUB_TOD IN1:= CURRENT RESULT         IN2:= START_TIME ST      WORK_TIME AAA:</pre>

- (1) If the transition condition (%I0.0.0) is ON, function SUB\_TOD (time of day subtraction) will be executed.
- (2) If END\_TIME declared as input variable is TOD#14:20:30.5 and the starting time to work START\_TIME is TOD#12:00:00, the required time to work WORK\_TIME declared as output variable will be T#2H20M30S500MS.

Input (IN1): END\_TIME (TOD) = TOD#14:20:30.5  
 (SUB\_TOD)  
 (IN2): START\_TIME (TOD) = TOD#12:00:00  
 ↓  
 Output (OUT): WORK\_TIME (TIME) = T#2H20M30S500MS



## TAN

Tangent Operation	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

\* Applied only in GM4-CPUC among GM4 series

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: tangent input value (radian)</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: the result value of Tangent operation IN, OUT should be the same data type.</p>

### ■ Function

It performs Tangent operation of IN and produces output OUT.

$$OUT = TAN(IN)$$

### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   BBB LD      INPUT TAN ST      RESULT BBB:</pre>

(1) If the transition condition (%M0) is ON, function TAN (Tangent operation) will be executed.

(2) If the value of input variable declared as INPUT is 0.7853... ( $\pi/4$  rad =  $45^\circ$ ), RESULT declared as output variable will be 1.0000.

$$TAN(\pi/4) = 1$$

Input (IN1): INPUT (REAL) = 0.7853

↓ (TAN)

Output (IN2): RESULT (REAL) = 9.99803722E-01

## 8. Basic Function/Function Block Library

### TIME\_TO\_\*\*\*

TIME type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: time data to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

#### ■ Function

It converts the IN type and produces OUT.

Function	Output type	Description
TIME_TO_UDINT	UDINT	Converts TIME into UDINT type. It converts only data type without changing the data (internal bit array state).
TIME_TO_DWORD	DWORD	Converts TIME into DWORD type. It converts only data type without changing the data (internal bit array state).
TIME_TO_STRING	STRING	Converts TIME into STRING type.

#### ■ Program Example

LD	IL
	<pre>LD      %M0 JMPN   AA LD      IN_VAL TIME_TO_UDINT ST      OUT_VAL AA:</pre>

(1) If the transition condition (%M0) is ON, function TIME\_TO\_UDINT will be executed.

(2) If input variable IN\_VAL (TIME) = T#120MS, output variable OUT\_VAL (UDINT) = 120.

Input (IN1): IN\_VAL (TIME) = T#120MS (16#78)

0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↓ (TIME\_TO\_UDINT)

Output (OUT): OUT\_VAL (UDINT) = 120 (16#78)

0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

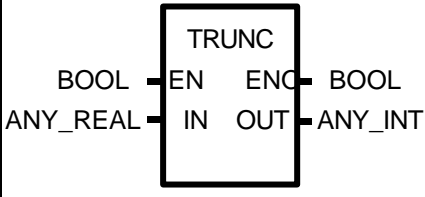


## 8. Basic Function/Function Block Library

### TRUNC

Set TRUNC (Round off the decimal fraction of IN and converts into integer number)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
 <p>The diagram shows a rectangular block labeled 'TRUNC'. On the left side, there are two input ports: 'EN' (with 'BOOL' above it) and 'IN' (with 'ANY_REAL' above it). On the right side, there are two output ports: 'ENO' (with 'BOOL' above it) and 'OUT' (with 'ANY_INT' above it).</p>	<p><b>Input</b> EN: executes the function in case of 1 IN: REAL value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: the Integer converted value</p>

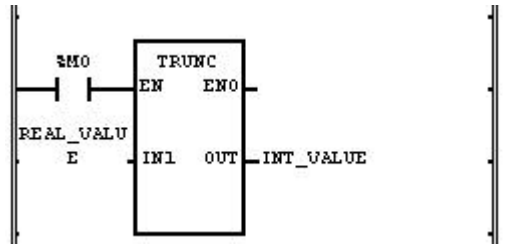
#### ■ Function

Function	Input type	Output type	Description
TRUNC	REAL LREAL	DINT LINT	Round off the decimal fraction of input IN and outputs the Integer value as OUT.

#### ■ Error

\_ERR, \_LER flags will be set: 1) if the converted value is greater than maximum value of data type connected to OUT; 2) if the variable connected to OUT is Unsigned Integer and the converted output value is a negative number, the output is 0.

#### ■ Program Example

LD	IL
 <p>The diagram shows a ladder logic network. The first rung consists of a normally open contact labeled '%M0'. This is followed by a coil labeled 'TRUNC'. The second rung consists of a coil labeled 'INT_VALUE'. The TRUNC block has inputs 'EN' and 'IN1', and outputs 'ENO' and 'OUT'. 'EN' is connected to '%M0', 'IN1' is connected to 'REAL_VALUE', 'ENO' is connected to 'E', and 'OUT' is connected to 'INT_VALUE'.</p>	<pre>LD      REAL_VALUE TRUNC ST      INT_VALUE</pre>

(1) If the transition condition (%M0) is ON, function TRUNC will be executed.

(2) If input variable REAL\_VALUE (REAL) = 1.6, output variable INT\_VALUE (INT) = 1.

If REAL\_VALUE (REAL) = -1.6, INT\_VALUE (INT) = -1.

Input (IN1): REAL\_VALUE (REAL) = 1.6

↓ (TRUNC)

Output (OUT): INT\_VALUE (INT) = 1

**UDINT\_TO\_\*\*\***

UDINT type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
<pre> graph LR     subgraph UDINT_TO_***         EN[EN]         IN[IN]         ENO[ENO]         OUT[OUT]     end     EN --&gt; ENO     IN --&gt; OUT         </pre>	<p><b>Input</b> EN: executes the function in case of 1 IN: Unsigned Double Integer value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
UDINT_TO_SINT	SINT	If input is 0~127, normal conversion. Otherwise an error occurs.
UDINT_TO_INT	INT	If input is 0~32767, normal conversion. Otherwise an error occurs.
UDINT_TO_DINT	DINT	If input is 0~2,147,483,648, normal conversion. Otherwise an error occurs.
UDINT_TO_LINT	LINT	Converts UDINT into LINT type normally.
UDINT_TO_USINT	USINT	If input is 0~255, normal conversion. Otherwise an error occurs.
UDINT_TO_UINT	UINT	If input is 0~65535, normal conversion. Otherwise an error occurs.
UDINT_TO_ULINT	ULINT	Converts UDINT into ULINT type normally.
UDINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
UDINT_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
UDINT_TO_WORD	WORD	Takes the lower 16 bits and converts into WORD type.
UDINT_TO_DWORD	DWORD	Converts into DWORD type without changing the internal bit array.
UDINT_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
UDINT_TO_BCD	DWORD	If input is 0 ~ 99,999,999, normal conversion. Otherwise an error occurs.
UDINT_TO_REAL	REAL	Converts UDINT into REAL type. During the conversion, an error caused by the precision may occur.
UDINT_TO_LREAL	LREAL	Converts UDINT into LREAL type. During the conversion, an error caused by the precision may occur.
UDINT_TO_TOD	TOD	Converts into TOD type without changing the internal bit array.
UDINT_TO_TIME	TIME	Converts into TIME type without changing the internal bit array.

■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set. If an error occurs, take the lower bits as many as a bit number of an output data type and produces the output without changing the internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   ZZ LD      IN_VAL UDINT_TO_TIME ST      OUT_VAL ZZ: </pre>

(1) If the input condition (%M0) is ON, function UDINT\_TO\_TIME will be executed.

(2) If input variable IN\_VAL (UDINT) = 123, output variable OUT\_VAL (TIME) = T#123MS.

Input (IN1): IN\_VAL (UDINT) = 123  
↓

Output (OUT): OUT\_VAL (TIME) = T#123MS

**UINT\_TO\_\*\*\***

UINT type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: Unsigned Integer value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
UINT_TO_SINT	SINT	If input is 0~127, normal conversion. Otherwise an error occurs.
UINT_TO_INT	INT	If input is 0~32,767, normal conversion. Otherwise an error occurs.
UINT_TO_DINT	DINT	Converts UINT into UDINT type normally.
UINT_TO_LINT	LINT	Converts UINT into ULINT type normally.
UINT_TO_USINT	USINT	If input is 0~255, normal conversion. Otherwise an error occurs.
UINT_TO_UDINT	UDINT	Converts UINT into UDINT type normally.
UINT_TO_ULINT	ULINT	Converts UINT into ULINT type.
UINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
UINT_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
UINT_TO_WORD	WORD	Converts into WORD type without changing the internal bit array.
UINT_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
UINT_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
UINT_TO_BCD	BCD	If input is 0~99,999,999, normal conversion. Otherwise an error occurs.
UINT_TO_REAL	REAL	Converts UINT into REAL type.
UINT_TO_LREAL	LREAL	Converts UINT into LREAL type.
UNIT_TO_DATE	DATE	Converts into DATE type without changing the internal bit array.

■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set. If error occurs, it takes as many lower bits as a bit number of output type and produces an output without changing its internal bit array.

## 8. Basic Function/Function Block Library

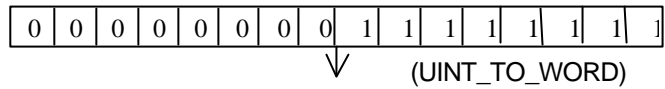
### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   PO LD      IN_VAL UINT_TO_WORD ST      OUT_VAL PO: </pre>

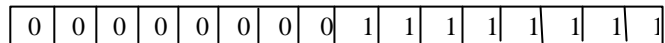
(1) If the input condition (%M0) is ON, function UINT\_TO\_WORD will be executed.

(2) If input variable IN\_VAL (UINT) = 255 (2#0000\_0000\_1111\_1111),  
output variable OUT\_VAL (WORD) = 2#0000\_0000\_1111\_1111.

Input (IN1): IN\_VAL (UINT) = 255



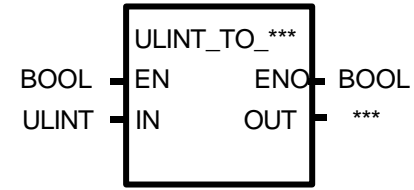
Output (OUT): OUT\_VAL (WORD) = 16#FF





**ULINT\_TO\_\*\*\***

ULINT type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: Unsigned Long Integer value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
ULINT_TO_SINT	SINT	If input is 0~127, normal conversion. Otherwise an error occurs.
ULINT_TO_INT	INT	If input is 0~32,767, normal conversion. Otherwise an error occurs.
ULINT_TO_DINT	DINT	If input is 0~2 <sup>31</sup> -1, normal conversion. Otherwise an error occurs.
ULINT_TO_LINT	LINT	If input is 0~2 <sup>63</sup> -1, normal conversion. Otherwise an error occurs.
ULINT_TO_USINT	USINT	If input is 0~255, normal conversion. Otherwise an error occurs.
ULINT_TO_UINT	UINT	If input is 0~65,535, normal conversion. Otherwise an error occurs.
ULINT_TO_UDINT	UDINT	If input is 0~2 <sup>32</sup> -1, normal conversion. Otherwise an error occurs.
ULINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
ULINT_TO_BYTE	BYTE	Takes the lower 8 bits and converts into BYTE type.
ULINT_TO_WORD	WORD	Takes the lower 16 bits and converts into WORD type.
ULINT_TO_DWORD	DWORD	Takes the lower 32 bits and converts into DWORD type.
ULINT_TO_LWORD	LWORD	Converts into LWORD type without changing the internal bit array.
ULINT_TO_BCD	BCD	If input is 0~9,999,999,999,999,999, normal conversion. Otherwise an error occurs.
ULINT_TO_REAL	REAL	Converts ULINT into REAL type. During the conversion, an error caused by the precision may occur.
ULINT_TO_LREAL	LREAL	Converts ULINT into LREAL type. During the conversion, an error caused by the precision may occur.

■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set. If error occurs, it takes as many lower bits as a bit number of output type and produces an output without changing its internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   PP LD      IN_VAL ULINT_TO_LINT ST      OUT_VAL PP:         </pre>

(1) If the input condition (%M0) is ON, function ULINT\_TO\_LINT will be executed.

(2) If input variable IN\_VAL (ULINT) = 123,567,899, then output variable OUT\_VAL (LINT) = 123,567,899.

Input (IN1): IN\_VAL (ULINT) = 123,567,899



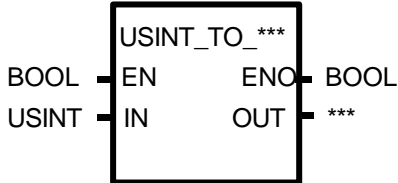
(ULINT\_TO\_LINT)

Output (OUT): OUT\_VAL (LINT) = 123,567,899

**USINT\_TO\_\*\*\***

USINT type conversion

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: Unsigned Short Integer value to be converted</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
USINT_TO_SINT	SINT	If input is 0~127, normal conversion. Otherwise an error occurs.
USINT_TO_INT	INT	Converts USINT into INT type normally.
USINT_TO_DINT	DINT	Converts USINT into DINT type normally.
USINT_TO_LINT	LINT	Converts USINT into LINT type normally.
USINT_TO_UINT	UINT	Converts USINT into UINT type normally.
USINT_TO_UDINT	UDINT	Converts USINT into UDINT type normally.
USINT_TO_ULINT	ULINT	Converts USINT into ULINT type normally.
USINT_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
USINT_TO_BYTE	BYTE	Converts into BYTE type without changing the internal bit array.
USINT_TO_WORD	WORD	Converts into WORD type filling the upper bits with 0.
USINT_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
USINT_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
USINT_TO_BCD	BCD	If input is 0 ~ 99, normal conversion. Otherwise an error occurs.
USINT_TO_REAL	REAL	Converts USINT into REAL type.
USINT_TO_LREAL	LREAL	Converts USINT into LREAL type.

■ **Error**

If a conversion error occurs, \_ERR and \_LER flags will be set. If error occurs, it takes as many lower bits as a bit number of output type and produces an output without changing its internal bit array.

## 8. Basic Function/Function Block Library

### ■ Program Example

LD	IL
	<pre> LD      %M0 JMPN   LL LD      IN_VAL USINT_TO_SINT ST      OUT_VAL LL: </pre>

- (1) If the input condition (%M0) is ON, function ULINT\_TO\_SINT will be executed.  
(2) If input variable IN\_VAL (USINT) = 123, output variable OUT\_VAL (SINT) = 123.

Input (IN1): IN\_VAL (USINT) = 123 (16#7B)

0	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---

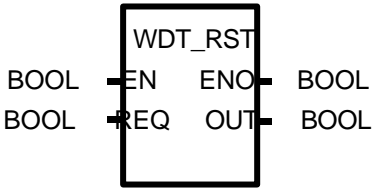
↓ (ULINT TO\_SINT)

Output (OUT): OUT\_VAL (SINT) = 123 (16#7B)

0	1	1	1	1	0	1	0
---	---	---	---	---	---	---	---

## WDT\_RST

Initialize Watch_Dog timer	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
 <p>The diagram shows a rectangular function block labeled 'WDT_RST'. On the left side, there are two input terminals: 'EN' (top) and 'REQ' (bottom), both labeled 'BOOL'. On the right side, there are two output terminals: 'ENO' (top) and 'OUT' (bottom), both labeled 'BOOL'.</p>	<p><b>Input</b> EN: executes the function in case of 1 REQ: requires to initialize watchdog timer</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: After Watch_Dog timer initialization, output will be 1.</p>

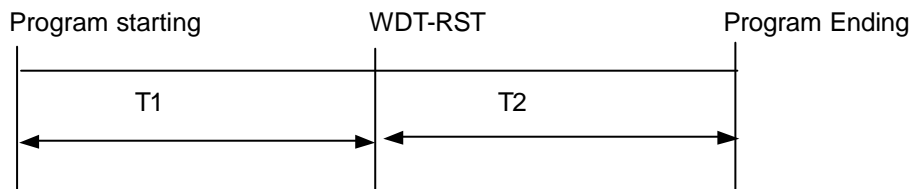
### ■ Function

It resets Watch-Dog Timer among the programs.

Available to use in case that scan time exceeds Watch-Dog Time set by the condition in the program.

If scan time exceeds the scan Watch\_Dog Time, please, change the scan time with the setting value of scan Watch\_Dog Timer in the ' Basic Parameters' of GMWIN.

Care must be taken so that either the time from 0 line of program to WDT\_RST function T1 or the time from WDT\_RST function to the time by the end of program T2 does not exceed the setting value of scan Watch\_Dog Timer.

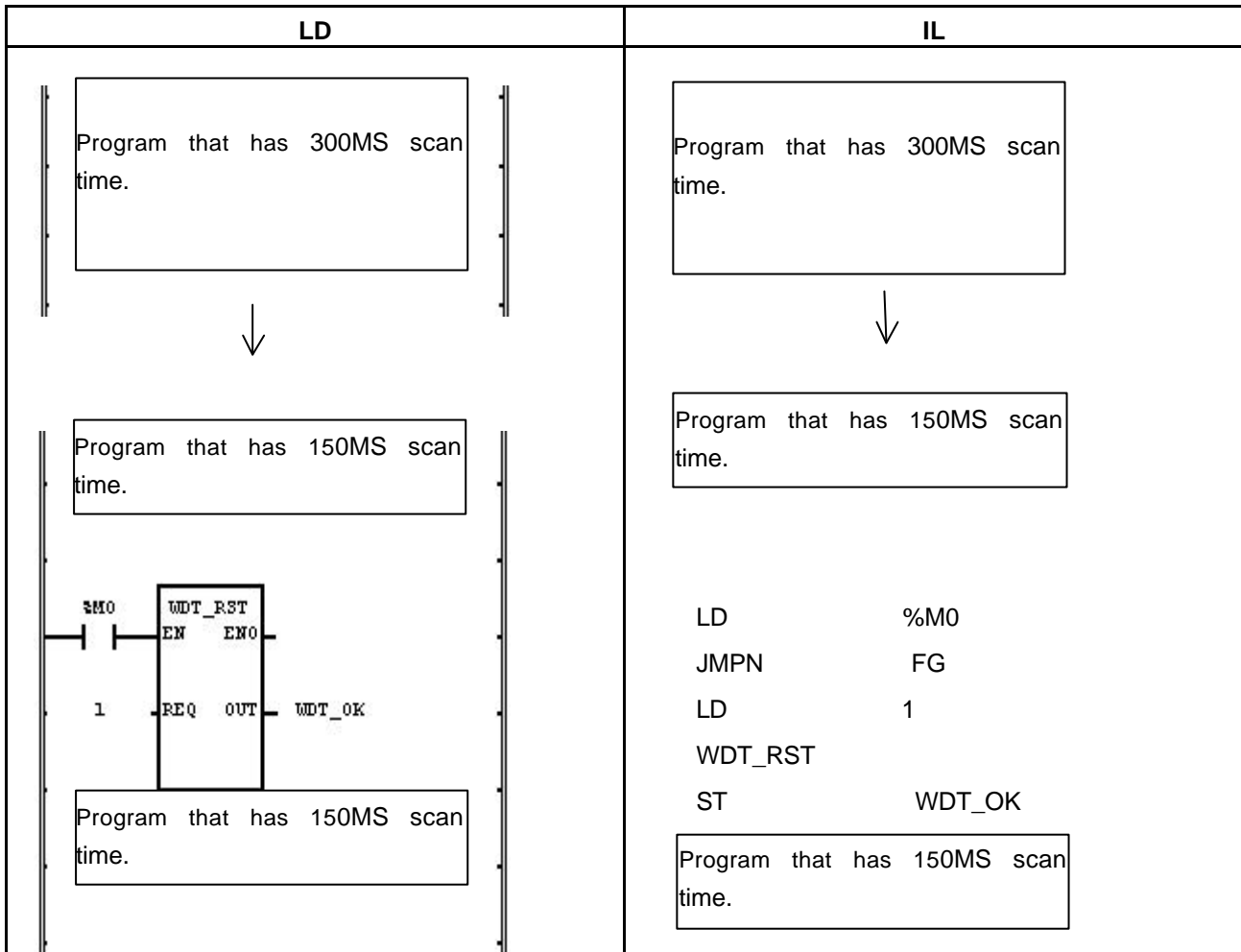


WDT\_RST function is available to use several times during 1 scan.

## 8. Basic Function/Function Block Library

### ■ Program Example

This is the program that the time to execute the program becomes 300ms according to the transition condition in the program of which scan Watch\_Dog timer is set as 200ms.



- (1) If the transition condition (%M0) is ON, function WDT-RST will be executed.
- (2) If WDT-RST function is executed, it is available to set the program that extends the scan time to 300ms according to the transition condition of program within the scan Watch\_Dog Time (200ms).

**WORD\_TO\_\*\*\***

WORD type conversion	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1 IN: Bit string to be converted (16 bit)</p> <p><b>Output</b> ENO: without an error, it will be 1 OUT: type-converted data</p>

■ **Function**

It converts the IN type and outputs it as OUT.

Function	Output type	Description
WORD_TO_SINT	SINT	Takes the lower 8 bits and converts into SINT type.
WORD_TO_INT	INT	Converts into INT type without changing the internal bit array.
WORD_TO_DINT	DINT	Converts into DINT type filling the upper bits with 0.
WORD_TO_LINT	LINT	Converts into LINT type filling the upper bits with 0.
WORD_TO_USINT	USINT	Takes the lower 8 bits and converts into SINT type.
WORD_TO_UINT	UINT	Converts into INT type without changing the internal bit array.
WORD_TO_UDINT	UDINT	Converts into DINT type filling the upper bits with 0.
WORD_TO_ULINT	ULINT	Converts into LINT type filling the upper bits with 0.
WORD_TO_BOOL	BOOL	Takes the lower 1 bit and converts into BOOL type.
WORD_TO_BYTE	BYTE	Takes the lower 8 bits and converts into SINT type.
WORD_TO_DWORD	DWORD	Converts into DWORD type filling the upper bits with 0.
WORD_TO_LWORD	LWORD	Converts into LWORD type filling the upper bits with 0.
WORD_TO_DATE	DATE	Converts into DATE type without changing the internal bit array.
WORD_TO_STRING	STRING	Converts WORD into STRING type.

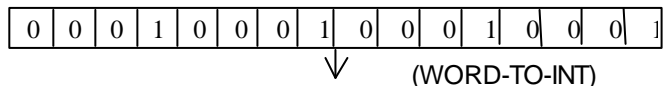
■ **Program Example**

LD	IL
	<pre>LD      %M0 JMPN   P0 LD      IN_VAL WORD_TO_INT ST      OUT_VAL PO:</pre>

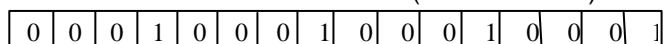
(1) If the input condition (%M0) is ON, function WORD-TO-INT will be executed.

(2) If input variable IN\_VAL (WORD) = 2#0001\_0001\_0001\_0001, output variable OUT\_VAL (INT) = 4096 + 256 + 16 + 1 = 4,369.

Input (IN1): IN\_VAL (WORD) = 16#1111



Output(OUT): OUT\_VAL(INT) = 4,369 (16#1111)



## 8. Basic Function/Function Block Library

### XOR

Exclusive OR	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b> EN: executes the function in case of 1            IN1: the value to be XOR            IN2: the value to be XOR            Input variable number can be extended up to 8.</p> <p><b>Output</b> ENO: without an error, it will be 1.            OUT: the result of XOR operation</p> <p>IN1, IN2, OUT should be all the same data type.</p>

#### ■ Function

Do XOR operation for IN1 and IN2 per bit and produces OUT.

```

IN1  1111 ..... 0000
XOR
IN2  1010 ..... 1010
OUT  0101 ..... 1010
    
```

#### □ Program Example

LD	IL
	<pre> LD      %M0 JMPN   ZZ LD      %MB10 XOR    IN1:= CURRENT RESULT         IN2:=  ABC ST      %QB0.0.0 ZZ:     </pre>

(1) If the transition condition (%M0) is ON, function XOR will be executed.

(2) If input variable %MB10 = 11001100, ABC = 11110000, the result of XOR operation for two inputs will be %QB0.0.0 = 00111100.

Input (IN1): %MB10 (BYTE) = 16#CC

1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

(IN2): ABC (BYTE) = 16#F0

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

(XOR)  
↓

Output (OUT): %QB0.0.0 (BYTE) = 16#3C

0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---



## **8.2 Application Function Library**

This chapter describes application function library (MASTER-K and others).

## 8. Basic Function/Function Block Library

### ARY\_ASC\_TO\_BCD

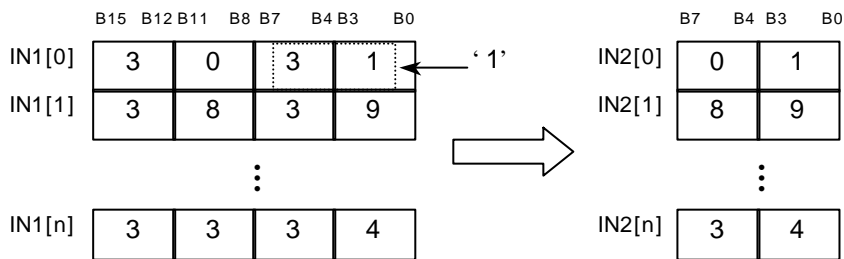
Converts ASCII array into BCD array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            IN1: ASCII Array input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: Dummy output</p> <p><b>In/Out</b>            IN2: BCD Array output</p>

#### ■ Function

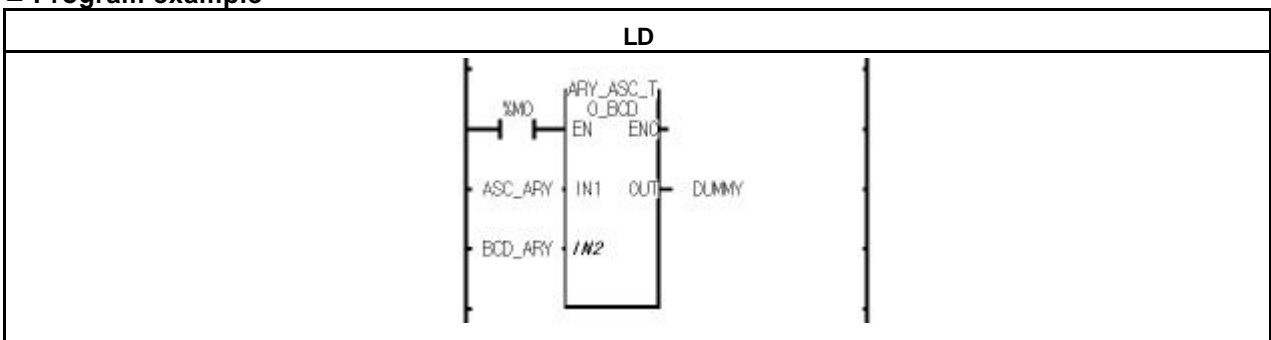
It converts a word array input (ASCII data) to a byte array output (BCD data).



#### ■ Error

If the number of each input array is different, there's no change in IN2 data, and `_ERR` and `_LER` flags are set. If the elements of IN1 array are not between 0 and 9 (hexadecimal), its responding elements of IN2 array are 16#00 (while other elements of IN1 are normally converted), and `_ERR` and `_LER` flags are set.

#### ■ Program example



(1) If the transition condition (%M0) is on, ARY\_ASC\_TO\_BCD function is executed.

(2) If the input ASC\_ARY data is:

ASC_ARY[0]	3031H
ASC_ARY[1]	3839H
ASC_ARY[2]	3334H

In/Out BCD\_ARY data is as follows:

BYTE_ARY[0]	01H
BYTE_ARY[1]	89H
BYTE_ARY[2]	34H

## 8. Basic Function/Function Block Library

### ARY\_ASC\_TO\_BYTE

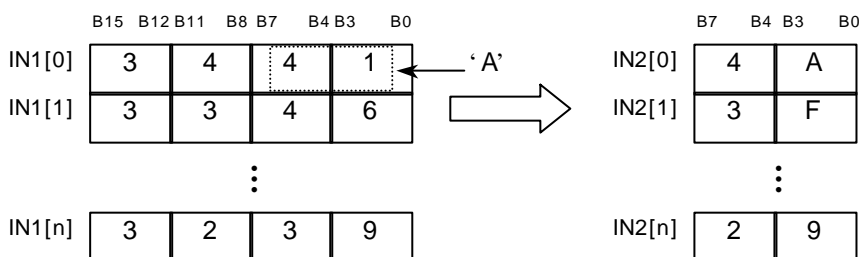
Converts ASCII array into BYTE array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            IN1: ASCII Array input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: Dummy Output</p> <p><b>In/Out</b>            IN2: BYTE Array Output</p>

#### ■ Function

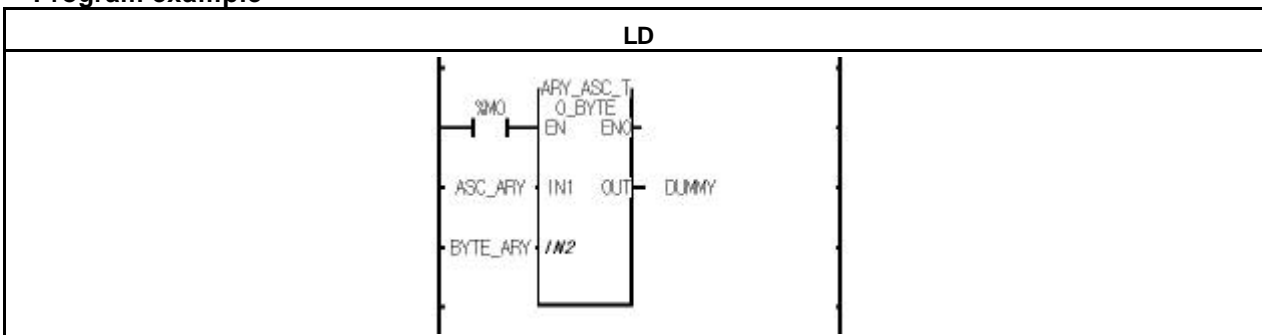
It converts a word array input (ASCII data) to a byte array output (hexadecimal).



#### ■ Error

If the number of each input array is different, there's no change in IN2 data, and `_ERR` and `_LER` flags are set. If the elements of IN1 array are not between 0 and F (hexadecimal), its responding elements of IN2 array are 0 (while other elements of IN1 are normally converted), and `_ERR` and `_LER` flags are set.

#### ■ Program example



(1) If the transition condition is (%M0) is on, ARY\_ASC\_TO\_BYTE function is executed.

(2) If Input ASC\_ARY is as below:

ASC_ARY[0]	3441H
ASC_ARY[1]	3346H
ASC_ARY[2]	3239H

In/Out BYTE\_ARRAY data is as follows:

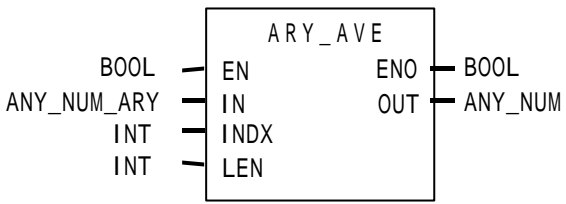
BYTE_ARRAY[0]	4AH
BYTE_ARRAY[1]	3FH
BYTE_ARRAY[2]	29H

## 8. Basic Function/Function Block Library

### ARY\_AVE \*\*\*

Finds an average of an array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            IN: data array for average            INDX: starting point to average in an array            LEN: number of array elements for average</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1            OUT: average of an array</p>

#### ■ Function

ARY\_AVE\_\*\*\* function finds an average for a specified length of an array .

Input and output array is the same type.

If LEN is a minus value, it finds an average between INDX (Array index) and ' INDX - |LEN|' .

Its output is rounded off.

Function	Output type	Description
ARY_AVE_SINT	SINT	Finds an average for SINT value (decimal is rounded off)
ARY_AVE_INT	INT	Finds an average for INT value (decimal is rounded off)
ARY_AVE_DINT	DINT	Finds an average for DINT value (decimal is rounded off)
ARY_AVE_LINT	LINT	Finds an average for LINT value (decimal is rounded off)
ARY_AVE_USINT	USINT	Finds an average for USINT value (decimal is rounded off)
ARY_AVE_UINT	UINT	Finds an average for UINT value (decimal is rounded off)
ARY_AVE_UDINT	UDINT	Finds an average for UDINT value (decimal is rounded off)
ARY_AVE_ULINT	ULINT	Finds an average for ULINT value (decimal is rounded off)
ARY_AVE_REAL	REAL	REAL.
ARY_AVE_LREAL	LREAL	LREAL.

#### ■ Error

If it is designated beyond the array range, \_ERR and \_LER flags are set.

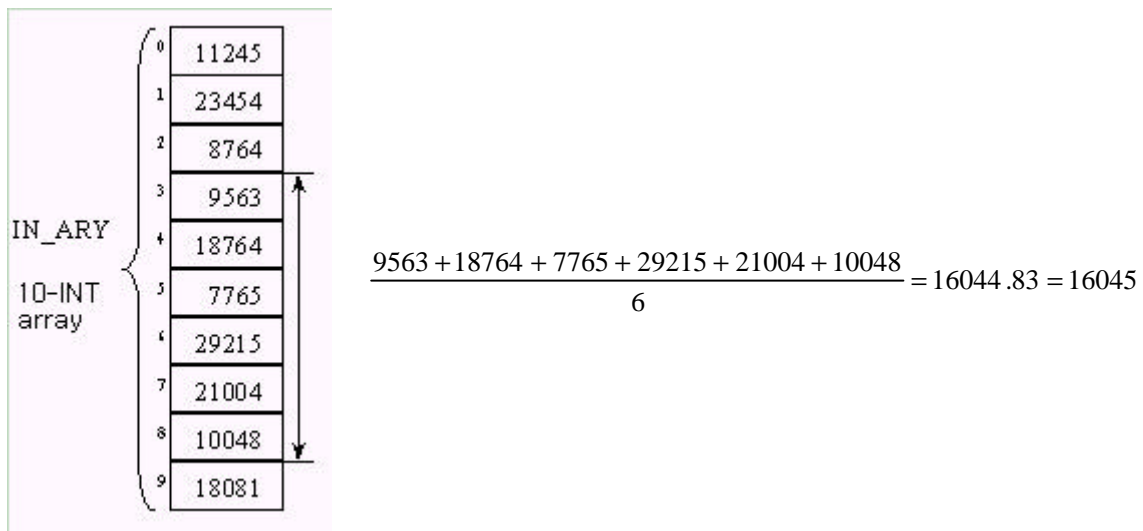
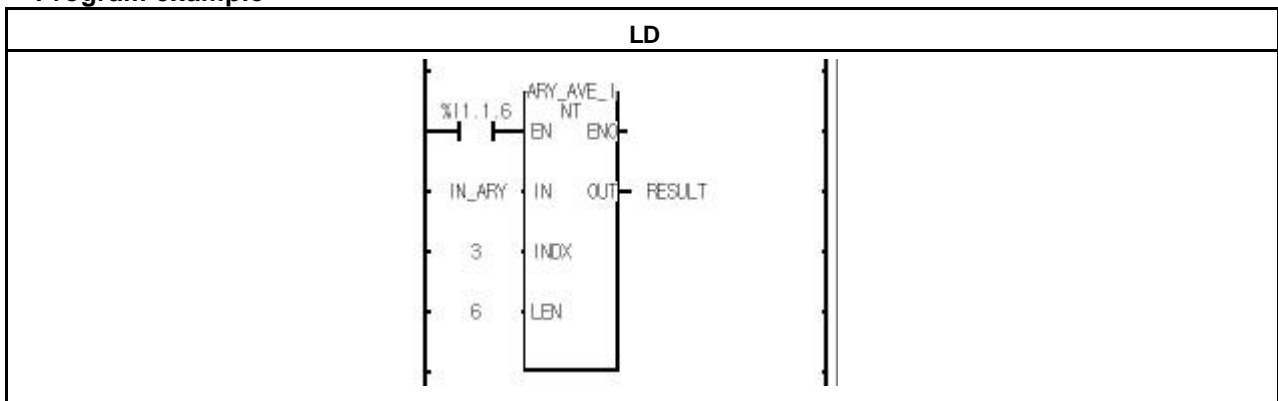
If an error occurs, the output is 0.

An error occurs when:

INDX < 0 or INDX > max. number of IN

**INDX + LEN > max. number of IN**

■ Program example



- (1) If input transition condition (%I1.1.6) is on, ARY\_AVE\_INT function is executed.
- (2) If an array is as the above, it finds an average between INDX 3 and 9.
- (3) The output value is rounded off.

## 8. Basic Function/Function Block Library

### ARY\_BCD\_TO\_ASC

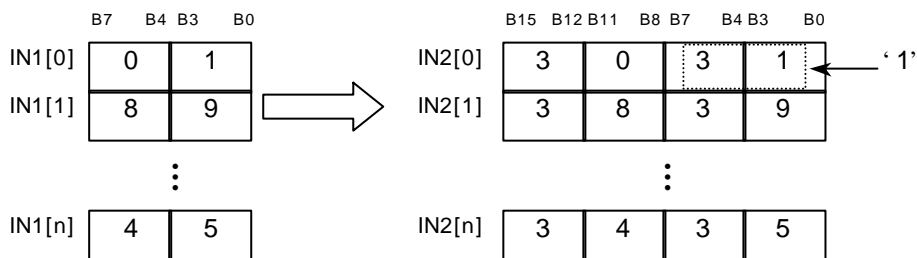
Converts BCD array into ASCII array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            IN1: BCD array input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: dummy output</p> <p><b>In/Out</b>            IN2: ASCII array output</p>

#### ■ Function

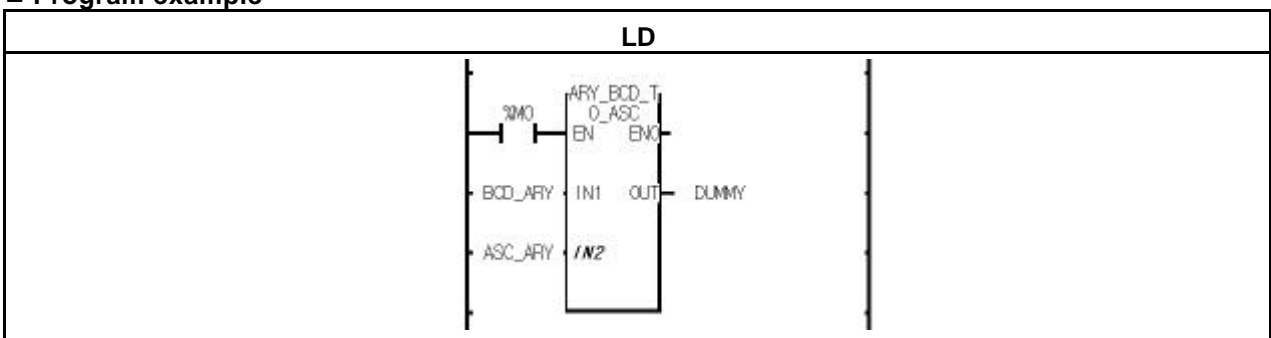
It converts a byte array input (BCD) to a word array (ASCII).



#### ■ Error

If the number of each input array is different, there's no change in IN2 data, and \_ERR and \_LER flags are set. If the elements of IN1 array are not between 0 and 9 (hexadecimal), its responding elements of IN2 array are 16#3030 ("00") (while other elements of IN1 are normally converted), and \_ERR and \_LER flags are set.

#### ■ Program example



(1) If the transition condition (%M0) is on, ARY\_BCD\_TO\_ASC function is executed.

(2) If the input BCD\_ARRAY is as below:

BYTE_ARRAY[0]	01H
BYTE_ARRAY[1]	89H
BYTE_ARRAY[2]	45H



The In/out ASC\_ARY is as follows:

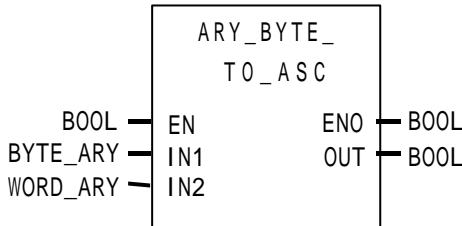
ASC_ARY[0]	3031H
ASC_ARY[1]	3839H
ASC_ARY[2]	3435H

## 8. Basic Function/Function Block Library

### ARY\_BYTE\_TO\_ASC

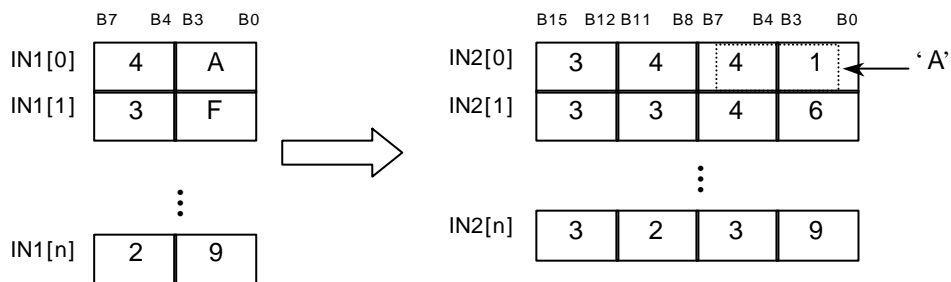
Converts BYTE array into ASCII array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            IN1: BYTE array input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: Dummy output</p> <p><b>In/Out</b>            IN2: ASCII Array Output</p>

#### ■ Function

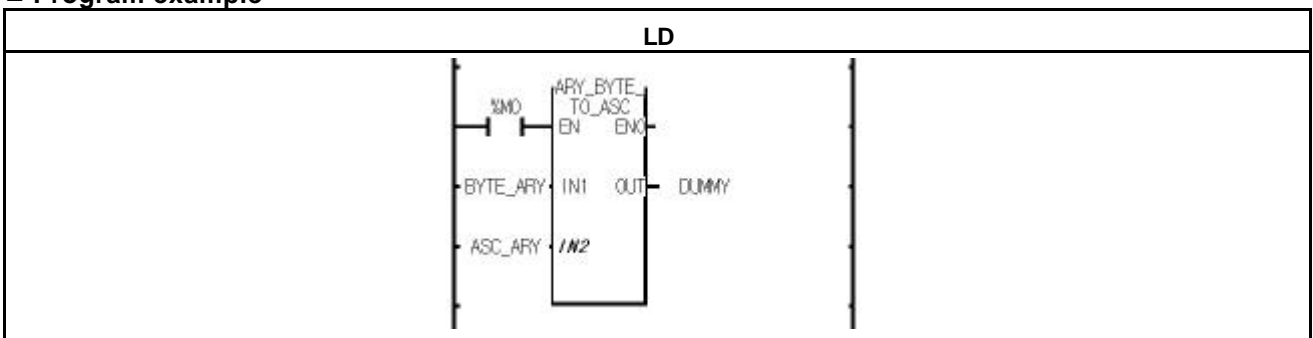
It converts a byte array input (HEX) to a word array (ASCII).



#### ■ Error

If the number of each input array is different, there's no change in IN2 data, and \_ERR and \_LER flags are set.

#### ■ Program example



- (1) If the transition condition (%M0) is on, ARY\_BYTE\_TO\_ASC function is executed.
- (2) If the input BYTE\_ARRAY is as below:

BYTE_ARRAY[0]	4AH
BYTE_ARRAY[1]	3FH
BYTE_ARRAY[2]	29H

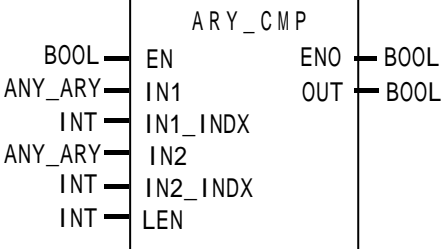
The output ASC\_ARY is as follows:

ASC_ARY[0]	3441H
ASC_ARY[1]	3346H
ASC_ARY[2]	3239H

## 8. Basic Function/Function Block Library

### ARY\_CMP \*\*\*

Array comparison	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1  IN1: first array to compare  IN1_INDX : starting point in 1<sup>st</sup> array for comparison  IN2: second array to compare  IN2_INDX : starting point in 2<sup>nd</sup> array for comparison  LEN: number of elements to compare</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1  OUT: if two arrays are equal, it will be 1</p>

#### ■ Function

It compare two arrays whether they have the same value.

If LEN is minus, it compare two arrays between IN\*\_INDX (Array INDX) and "Array INDX - |LEN|".

Function	Input array type	Description
ARY_CMP_BOOL	BOOL	Compares two BOOL Arrays.
ARY_CMP_BYTE	BYTE	Compares two BYTE Arrays.
ARY_CMP_WORD	WORD	Compares two WORD Arrays.
ARY_CMP_DWORD	DWORD	Compares two DWORD Arrays.
ARY_CMP_LWORD	LWORD	Compares two LWORD Arrays.
ARY_CMP_SINT	SINT	Compares two SINT Arrays.
ARY_CMP_INT	INT	Compares two INT Arrays.
ARY_CMP_DINT	DINT	Compares two DINT Arrays.
ARY_CMP_LINT	LINT	Compares two LINT Arrays.
ARY_CMP_USINT	USINT	Compares two USINT Arrays.
ARY_CMP_UINT	UINT	Compares two UINT Arrays.
ARY_CMP_UDINT	UDINT	Compares two UDINT Arrays.
ARY_CMP_ULINT	ULINT	Compares two ULINT Arrays.
ARY_CMP_REAL	REAL	Compares two REAL Arrays.
ARY_CMP_LREAL	LREAL	Compares two LREAL Arrays.
ARY_CMP_TIME	TIME	Compares two TIME Arrays.
ARY_CMP_DATE	DATE	Compares two DATE Arrays.
ARY_CMP_TOD	TOD	Compares two TOD Arrays.
ARY_CMP_DT	DT	Compares two DT Arrays.

■ Error

If it is designated beyond the array range, \_ERR and \_LER flags are set.

An error occurs when:

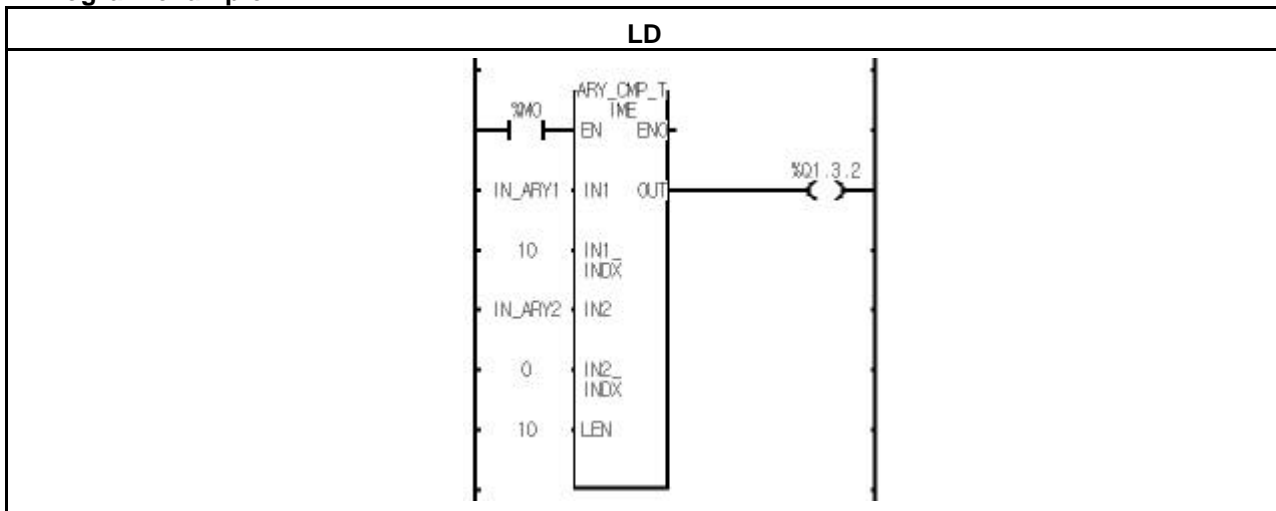
IN1\_INDIX < 0 or IN1\_INDIX > max. number of IN1

IN2\_INDIX < 0 or IN2\_INDIX > max. number of IN2

IN1\_INDIX + LEN ≥ max. number of IN1

IN2\_INDIX + LEN ≥ max. number of IN2

■ Program example



- (1) If the input transition condition (%M0) is on, ARY\_CMP\_TIME function is executed.
- (2) When IN\_ARY1 is a time array with 100 elements and IN\_ARY2 is a time array with 10 elements, if the elements from 11<sup>th</sup> to 20<sup>th</sup> of IN\_ARY1 and the elements of IN\_ARY 2 are equal, the output %Q1.3.2 is on.

## 8. Basic Function/Function Block Library

### ARY\_FLL\_\*\*\*

Filling an array with data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
<pre> graph LR     subgraph ARY_FLL_***         EN[EN]         DATA[DATA]         IN[IN]         INDX[INDX]         LEN[LEN]         ENO[ENO]         OUT[OUT]     end     EN --- ENO     DATA --- OUT     IN --- OUT     INDX --- OUT     LEN --- OUT         </pre>	<p><b>Input</b>  EN: executes the function in case of 1  DATA: the data to fill an array  INDX: starting point of an array to be filled  LEN: number of array elements to be filled</p> <p><b>Output</b>  ENO: without an error, it will be 1  OUT: without an error, it will be 1</p> <p><b>In/Out</b>  IN: an array to be filled</p>

#### ■ Function

It fills an array with the input data.

If LEN is minus, it fills an array from INDX to "INDX - |LEN|".

Function	In/Out Array type	Description
ARY_FLL_BOOL	BOOL	Fills a BOOL Array with the input data.
ARY_FLL_BYTE	BYTE	Fills a BYTE Array with the input data.
ARY_FLL_WORD	WORD	Fills a WORD Array with the input data.
ARY_FLL_DWORD	DWORD	Fills a DWORD Array with the input data.
ARY_FLL_LWORD	LWORD	Fills a LWORD Array with the input data.
ARY_FLL_SINT	SINT	Fills a SINT Array with the input data.
ARY_FLL_INT	INT	Fills a INT Array with the input data.
ARY_FLL_DINT	DINT	Fills a DINT Array with the input data.
ARY_FLL_LINT	LINT	Fills a LINT Array with the input data.
ARY_FLL_USINT	USINT	Fills a USINT Array with the input data.
ARY_FLL_UINT	UINT	Fills a UINT Array with the input data.
ARY_FLL_UDINT	UDINT	Fills a UDINT Array with the input data.
ARY_FLL_ULINT	ULINT	Fills a ULINT Array with the input data.
ARY_FLL_REAL	REAL	Fills a REAL Array with the input data.
ARY_FLL_LREAL	LREAL	Fills a LREAL Array with the input data.
ARY_FLL_TIME	TIME	Fills a TIME Array with the input data.
ARY_FLL_DATE	DATE	Fills a DATE Array with the input data.
ARY_FLL_TOD	TOD	Fills a TOD Array with the input data.
ARY_FLL_DT	DT	Fills a DT Array with the input data.

■ Error

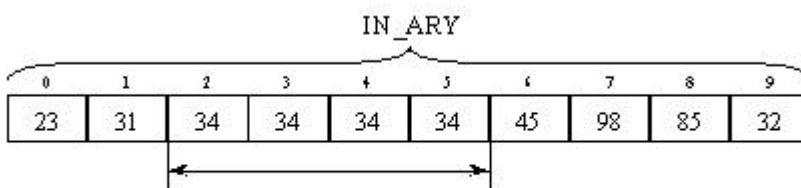
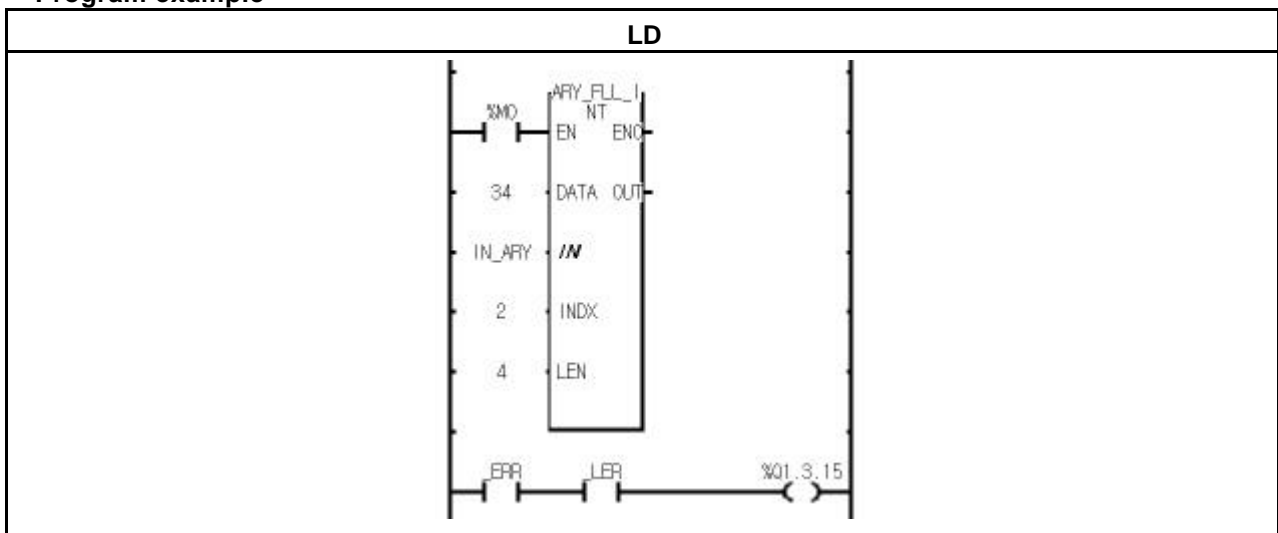
If it is designated beyond the array range, \_ERR and \_LER flags are set.  
 If an error occurs, there's no change in arrays and OUT is off.

An error occurs when:

$INDX < 0$  or  $INDX > \text{max. element number of IN}$

$INDX + LEN \geq \text{max. element number of IN}$

■ Program example



Fills 4 elements starting from INDX.

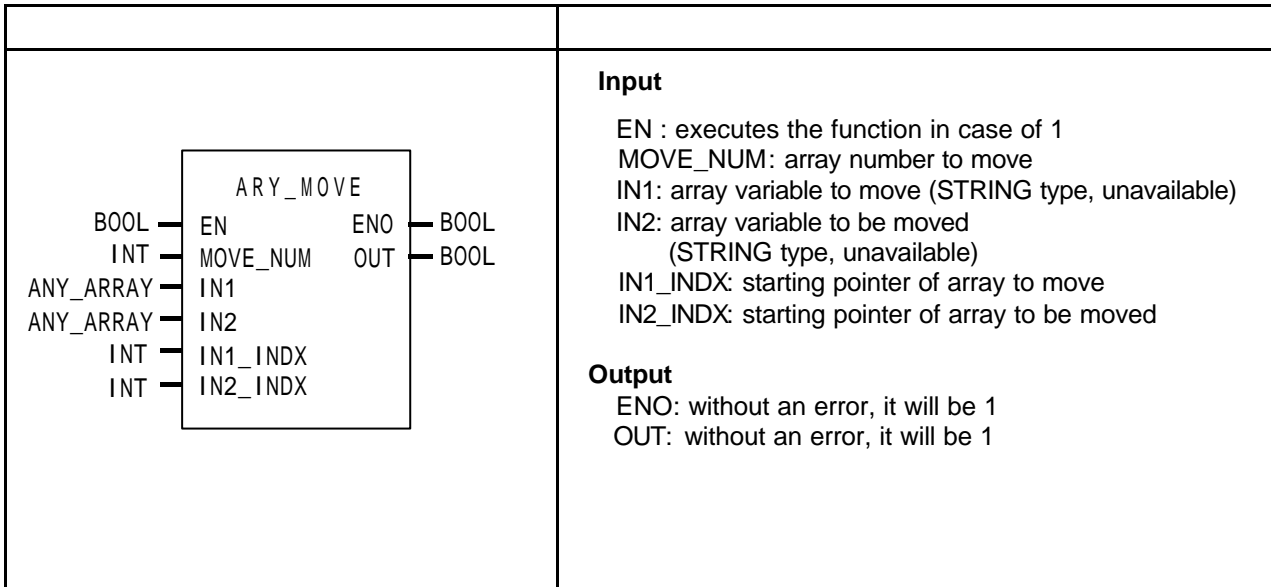
- (1) If input condition (%M0) is on, ARY\_FLL\_INT function is executed.
- (2) It fills 4 elements of IN\_ARRAY starting from INDX with 34.
- (3) If LEN is 9, it is beyond the array range and an error occurs; \_ERR and \_LER flags are set and the output (%Q1.13.15) is on.

## 8. Basic Function/Function Block Library

### ARY\_MOVE

Array move

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							



#### ■ Function

If EN is 1, it moves IN1 data to IN2.

It copies MOVE\_NUM elements of IN1 (from IN1\_INDX) and pastes it in IN2 (from IN2\_INDX).

IN1 and IN2 are the same data type (The number of each array can be different).

The data size is as follows:

Data size	Variable type
1 Bit	BOOL
8 Bit	BYTE, SINT, USINT
16 Bit	WORD / INT / UINT / DATE
32 Bit	DWORD / DINT / UDINT / TIME / TOD
64 Bit	DT

#### ■ Error

An error occurs when IN1 and IN2 data size are different.

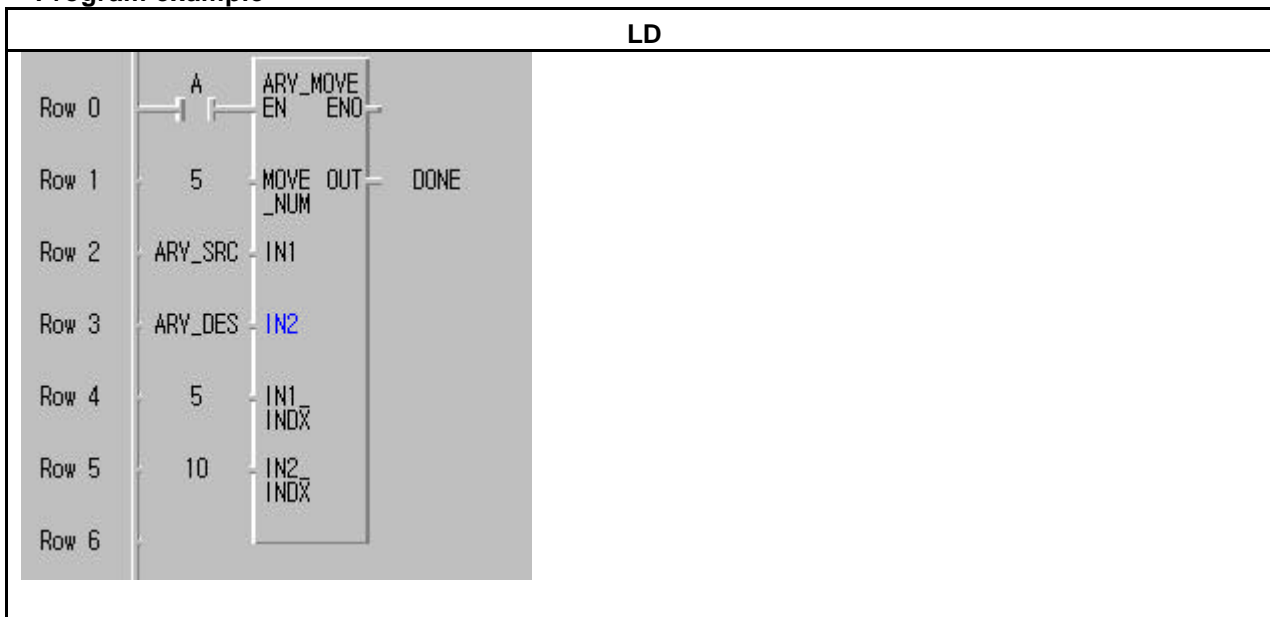
An error occurs when:

- 1) the array number of IN1 Array < (IN1\_INDX + MOVE\_NUM)
- 2) the array number of IN2 Array < (IN2\_INDX + MOVE\_NUM)

Then ARY\_MOVE function is not executed, OUT is 0, ENO is off and \_ERR and \_LER flags are set.



■ Program example



Variable name	Variable type	Array number
ARY_SRC	INT	10
ARY_DES	WORD	15

- (1) If the transition condition (A) is on, ARY\_MOVE function is executed.
- (2) It moves 5 elements from ARY\_SRC[5] to ARY\_DES[10].

Now the data type of ARY\_DES is WORD, it's hexadecimal.

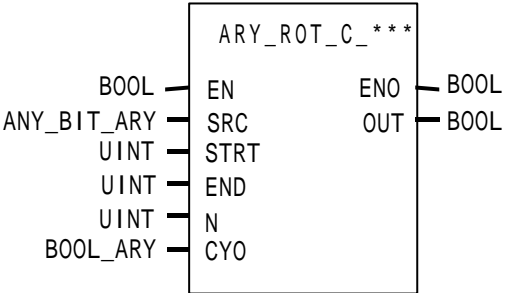
Before				After			
ARY_SRC[0]	0	ARY_DES[0]	16#0	ARY_SRC[0]	0	ARY_DES[0]	16#0
ARY_SRC[1]	11	ARY_DES[1]	16#1	ARY_SRC[1]	11	ARY_DES[1]	16#1
ARY_SRC[2]	22	ARY_DES[2]	16#2	ARY_SRC[2]	22	ARY_DES[2]	16#2
ARY_SRC[3]	33	ARY_DES[3]	16#3	ARY_SRC[3]	33	ARY_DES[3]	16#3
ARY_SRC[4]	44	ARY_DES[4]	16#4	ARY_SRC[4]	44	ARY_DES[4]	16#4
ARY_SRC[5]	55	ARY_DES[5]	16#5	ARY_SRC[5]	55	ARY_DES[5]	16#5
ARY_SRC[6]	66	ARY_DES[6]	16#6	ARY_SRC[6]	66	ARY_DES[6]	16#6
ARY_SRC[7]	77	ARY_DES[7]	16#7	ARY_SRC[7]	77	ARY_DES[7]	16#7
ARY_SRC[8]	88	ARY_DES[8]	16#8	ARY_SRC[8]	88	ARY_DES[8]	16#8
ARY_SRC[9]	99	ARY_DES[9]	16#9	ARY_SRC[9]	99	ARY_DES[9]	16#9
		ARY_DES[10]	16#A			ARY_DES[10]	16#37
		ARY_DES[11]	16#B			ARY_DES[11]	16#42
		ARY_DES[12]	16#C			ARY_DES[12]	16#4D
		ARY_DES[13]	16#D			ARY_DES[13]	16#58
		ARY_DES[14]	16#E			ARY_DES[14]	16#63

## 8. Basic Function/Function Block Library

### ARY\_ROT\_C \*\*\*

Bit rotation of array with carry

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

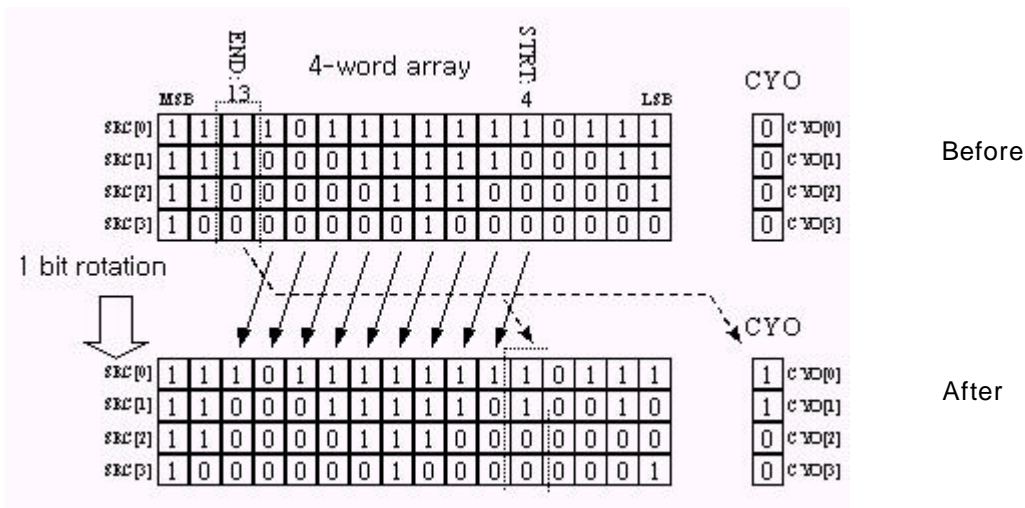
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            STRT: starting bit to rotate            END: ending bit to rotate            N: number to rotate</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: without an error, it will be 1</p> <p><b>In/Out</b>            SRC: Source Array to rotate            CYO: output Carry bit Array</p>

#### ■ Function

It rotates as many bits of array elements as they' re specified.

Setting:

- Scope: it sets a rotation scope with STRT and END.
- Rotation direction and time: it rotates N times from STRT to END.
- Output: the result is stored in ANY\_BIT\_ARRAY and a bit array data from END to STRT is written at CYO.

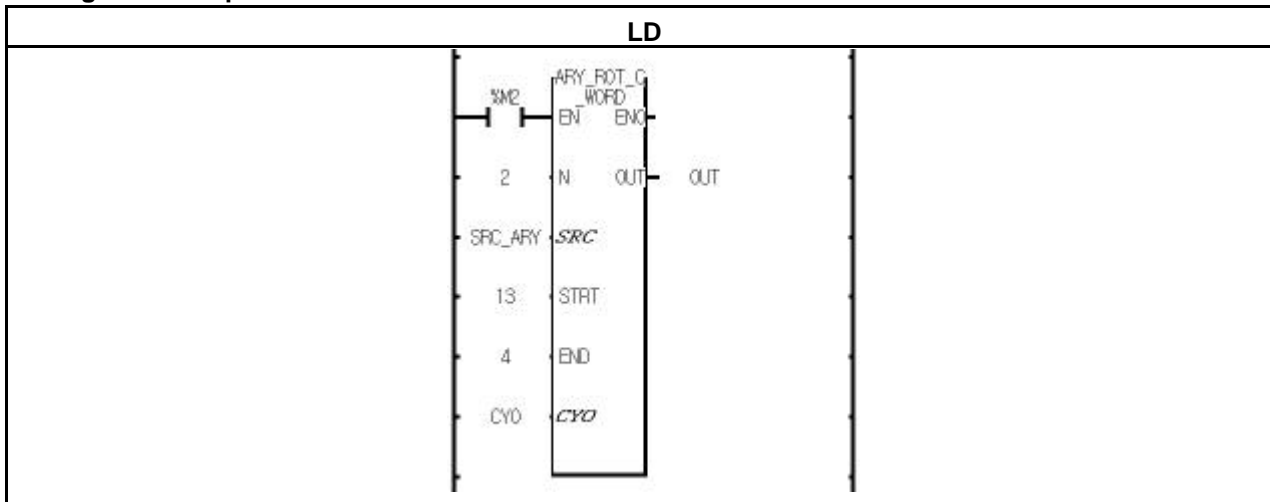


Function	In/out Array type	Description
ARY_ROT_C_BYTE	BYTE	It rotates elements of an array as many bits as they' re specified.
ARY_ROT_C_WORD	WORD	
ARY_ROT_C_DWORD	DWORD	
ARY_ROT_C_LWORD	LWORD	

■ Error

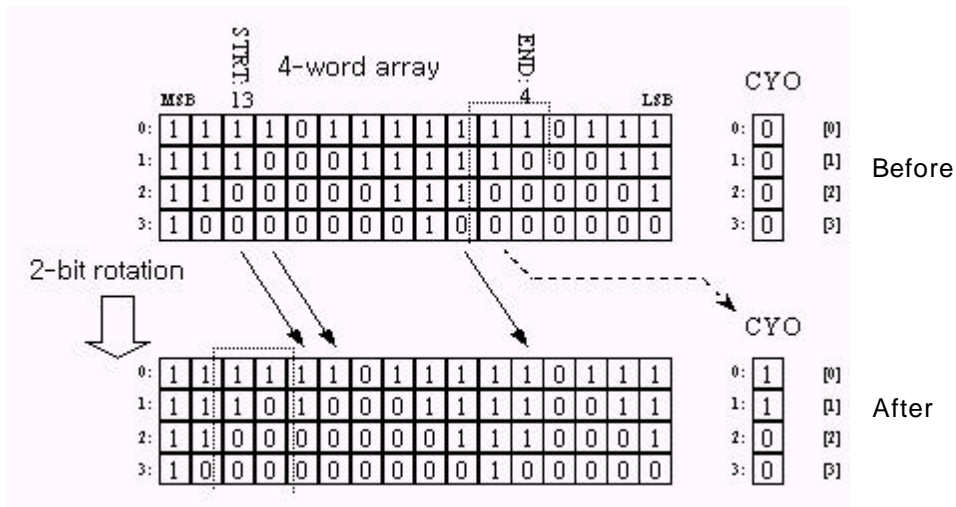
If the number of SRC and CYO Arrays are different, \_ERR and \_LER flags are set.  
 If STRT and END are out of bit range of SRC, an error occurs.  
 When an error occurs, there's no change in SRC and CYO.

■ Program example



- (1) If the input condition (%M2) is on, ARY\_ROT\_C\_WORD function is executed.
- (2) It rotates 2 times the bit (from 4 to 13 bit) arrays of SRC\_ARY from STRT to END.
- (3) The result is stored at SRC\_ARY and the carry bit arrays are written in CYO BOOL Array.

(Before)  
 SRC\_ARY : 16#F7F7  
           16#E3E3  
           16#C1C1  
           16#8080  
 (N) : 2  
 (After)  
 SRC\_ARY : 16#FDF7  
           16#E8F3  
           16#C071  
           16#8020  
 CYO : 2#1100

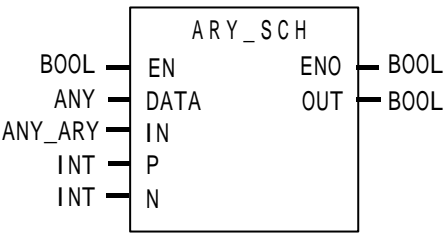


## 8. Basic Function/Function Block Library

### ARY\_SCH \*\*\*

Array search

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

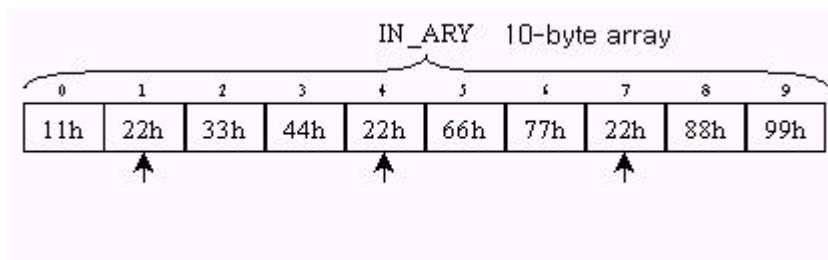
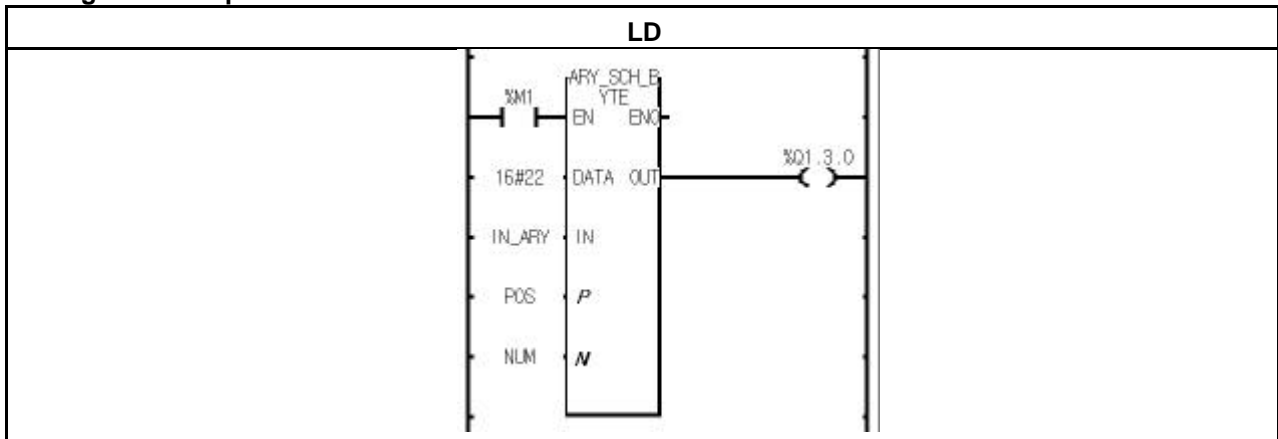
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            DATA: data to search            IN: array to search</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: if it finds, it will be 1</p> <p><b>In/Out</b>            P: first position of an object array            N: total number of array elements equal to an object</p>

#### ■ Function

It finds an equal value of input in arrays and produces its first position and total number. When it finds at least one which is equal to an object in arrays, OUT is 1.

Function	Input Array type	Description
ARY_SCH_BOOL	BOOL	Search in BOOL Array.
ARY_SCH_BYTE	BYTE	Search in BYTE Array.
ARY_SCH_WORD	WORD	Search in WORD Array.
ARY_SCH_DWORD	DWORD	Search in DWORD Array.
ARY_SCH_LWORD	LWORD	Search in LWORD Array.
ARY_SCH_SINT	SINT	Search in SINT Array.
ARY_SCH_INT	INT	Search in INT Array.
ARY_SCH_DINT	DINT	Search in DINT Array.
ARY_SCH_LINT	LINT	Search in LINT Array.
ARY_SCH_USINT	USINT	Search in USINT Array.
ARY_SCH_UINT	UINT	Search in UINT Array.
ARY_SCH_UDINT	UDINT	Search in UDINT Array.
ARY_SCH_ULINT	ULINT	Search in ULINT Array.
ARY_SCH_REAL	REAL	Search in REAL Array.
ARY_SCH_LREAL	LREAL	Search in LREAL Array.
ARY_SCH_TIME	TIME	Search in TIME Array.
ARY_SCH_DATE	DATE	Search in DATE Array.
ARY_SCH_TOD	TOD	Search in TOD Array.
ARY_SCH_DT	DT	Search in DT Array.

■ Program example



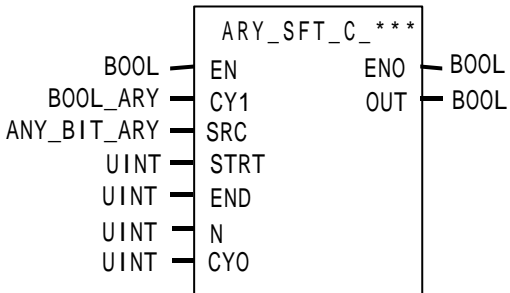
- (1) If the input condition (%M1) is on, ARY\_SCH\_BYTE function is executed.
- (2) When IN\_ARY is a 10-byte array, if you search for "22h" in this array, three bytes are found as the above.
- (3) The result is: 1) 1, the first position of an array, is stored at POS; 2) 3, the total number, is stored at NUM. The total number is 3, so the output %Q1.3.0 is on.

## 8. Basic Function/Function Block Library

### ARY\_SFT\_C \*\*\*

Array bit shift left with carry

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

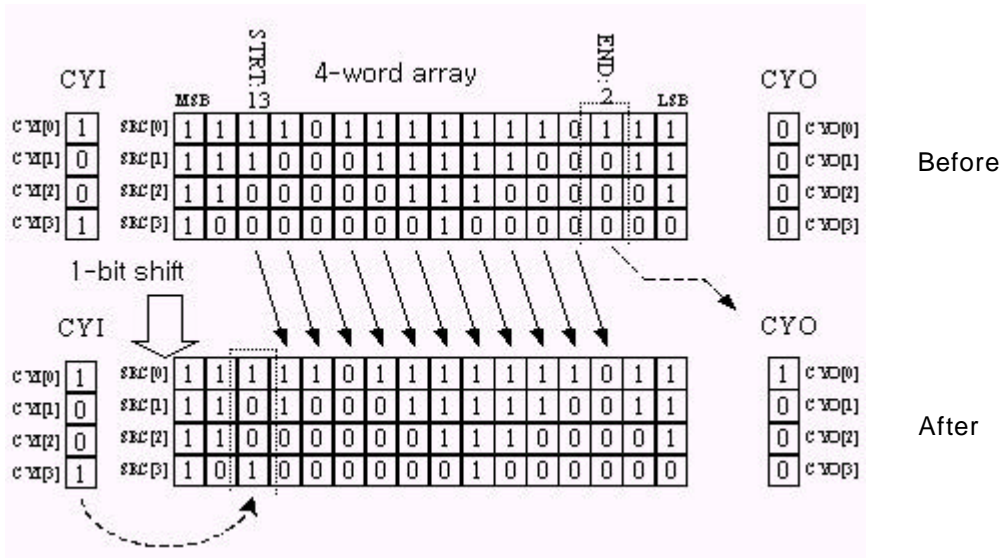
Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1            CYI: Input Carry bit Array            STRT: starting bit to shift            END: ending bit to shift            N: bit number to shift</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1            OUT: without an error, it will be 1</p> <p><b>In/Out</b></p> <p>SRC: Source Array to shift            CYO: Output Carry bit Array after shift</p>

#### ■ Function

It shifts as many bits of array elements as they' re specified.

Setting:

- Scope: it sets a shifting scope with STRT and END.
- Shifting direction and time: it shifts N times from STRT to END.
- Input data: it fills the empty bits with input data (CYI).
- Output: the result is stored in ANY\_BIT\_ARRAY and an overflowing bit array data from END is written at CYO.

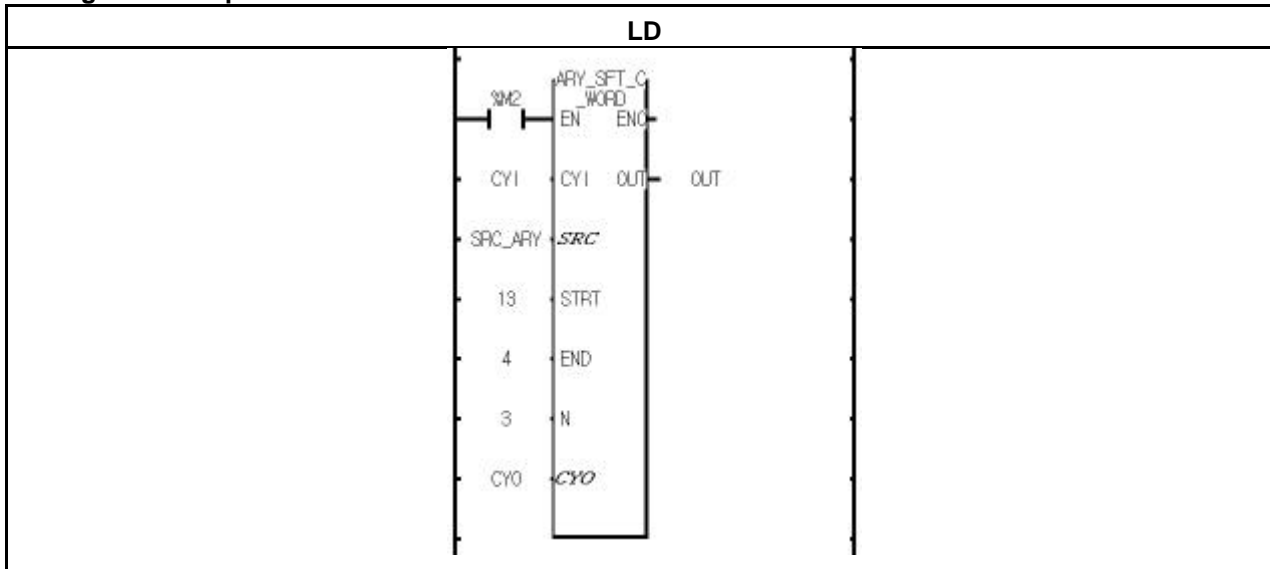


Function	In/Out Array type	Description
ARY_SFT_C_BYTE	BYTE	It shifts as many bits of array elements as they' re specified.
ARY_SFT_C_WORD	WORD	
ARY_SFT_C_DWORD	DWORD	
ARY_SFT_C_LWORD	LWORD	

■ Error

If the number of CYI, SRC and CYO Array are different, \_ERR and \_LER flags are set.  
 An error occurs if STRT and END are out of SRC range.  
 When an error occurs, there' s no change in SRC and CYO.

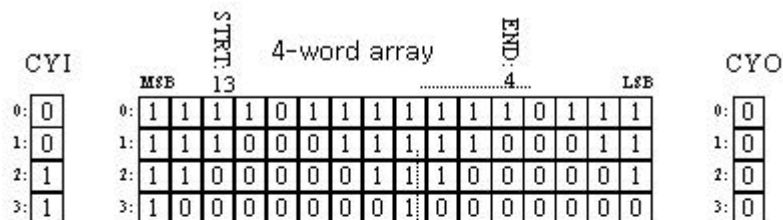
■ Program example



- (1) If input condition (%M2) is on, ARY\_SFT\_C\_WORD function is executed.
- (2) It shifts a bit array (from 4 to 13 bit) of SRC 3 times from STRT to END.
- (3) The bit array after shifting is filled with CYI (2#0011).
- (4) It produces its shifting result at SRC\_ARY and a carry bit array is written at CYO.

(Before)

CYI: 2#0011  
 SRC\_ARY: 16#F7F7  
 16#E3E3  
 16#C1C1  
 16#8080

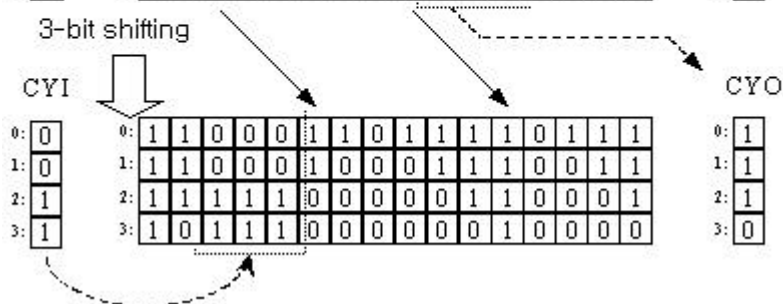


Before

(N): 3

(After)

SRC\_ARY: 16#C6F7  
 16#C473  
 16#F831  
 16#B810  
 CYO: 2#1110



After

## 8. Basic Function/Function Block Library

### ARY\_SWAP \*\*\*

Upper/lower elements swapping of an array

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1            IN1: array input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: Dummy output</p> <p><b>In/Out</b>            IN2: array output after swapping</p>

#### ■ Function

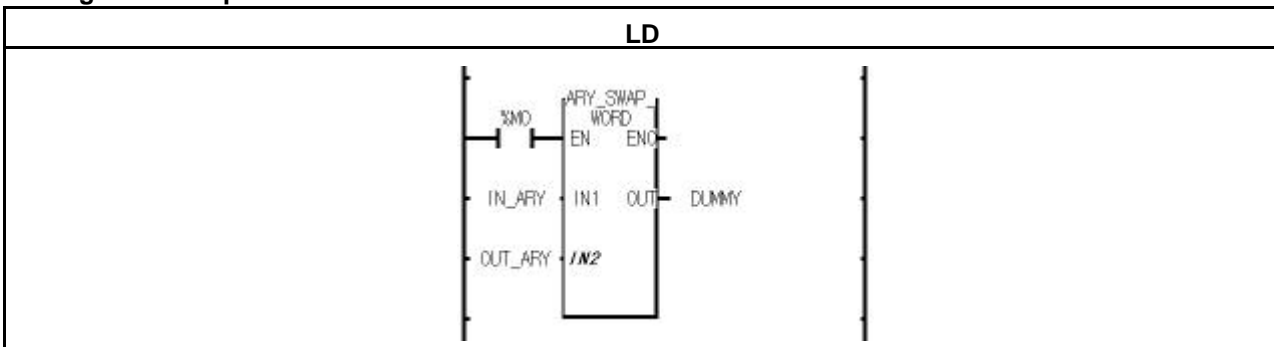
It swaps upper/lower elements after dividing an array.

Function	Input type	Description
ARY_SWAP_BYTE	BYTE	Swaps upper/lower nibble of byte elements.
ARY_SWAP_WORD	WORD	Swaps upper/lower byte of WORD elements.
ARY_SWAP_DWORD	DWORD	Swaps upper/lower WORD of DWORD elements.
ARY_SWAP_LWORD	LWORD	Swaps upper/lower DWORD of LWORD elements.

#### ■ Error

\_ERR and \_LER flags are set if two arrays are different; there's no change in an IN2 array.

#### ■ Program example



(1) If the transition condition (%M0) is on, ARY\_SWAP\_WORD function is executed.

(2) If IN\_ARRAY data is as below:

IN_ARRAY[0]	12ABH
IN_ARRAY[1]	23BCH
IN_ARRAY[2]	34CDH



## 8. Basic Function/Function Block Library

---

OUT\_ARY data is as follows:

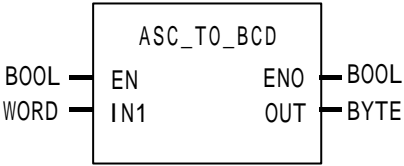
OUT_ARY[0]	AB12H
OUT_ARY[1]	BC23H
OUT_ARY[2]	CD34H

## 8. Basic Function/Function Block Library

### ASC\_TO\_BCD

Converts ASCII to BCD

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: ASCII input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: BCD output</p>

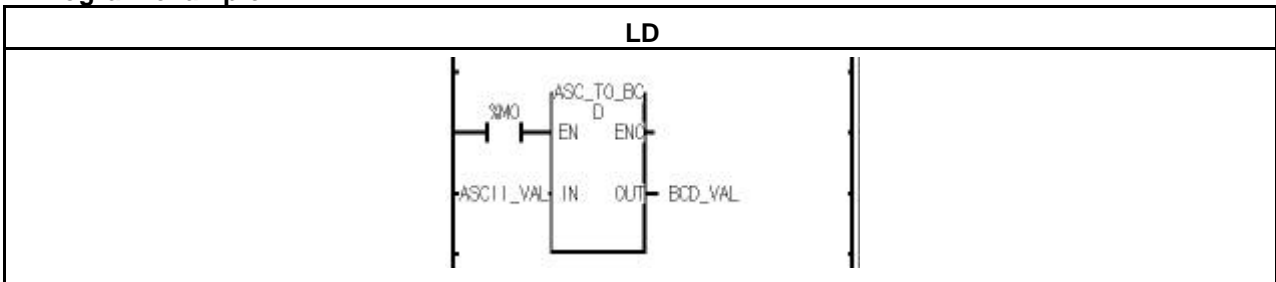
#### ■ Function

It converts two ASCII data into two-digit BCD (Binary Coded Decimal) data.

#### ■ Error

If IN is not hexadecimal number between 0 ~ 9, the output is 16#00 and \_ERR and \_LER flags will be set.

#### ■ Program example



(1) If the transition condition (%M0) is on, ASC\_TO\_BCD function is executed.

(2) If input variable ASCII\_VAL (WORD) = 16#3732 = "72", output variable BCD\_VAL (BYTE) = 16#72.

## ASC\_TO\_BYTE

Converts ASCII to BYTE data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN : executes the function in case of 1. IN : ASCII input</p> <p><b>Output</b></p> <p>ENO : without an error, it will be 1 OUT : BYTE Output</p>

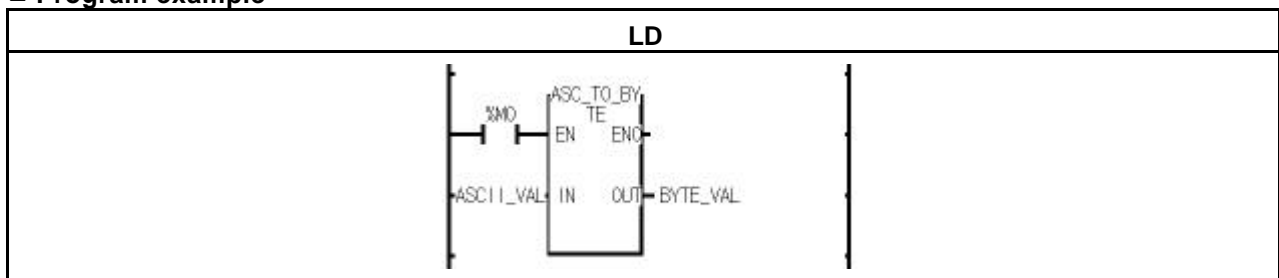
### ■ Function

It converts two ASCII data to 2-digit hexadecimal (HEX).

### ■ Error

If IN is not between ' 0' and ' F' , its output is 0 and \_ERR/\_LER flags are set.

### ■ Program example



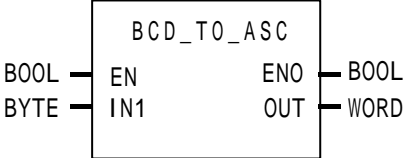
- (1) If the transition condition (%M0) is on, ASC\_TO\_BYTE function is executed.
- (2) If input ASCII\_VAL (WORD) = 16#4339, output BYTE\_VAL (BYTE) = 16#C9.

## 8. Basic Function/Function Block Library

### BCD\_TO\_ASC

Converts BCD to ASCII data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: BCD input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: ASCII Output</p>

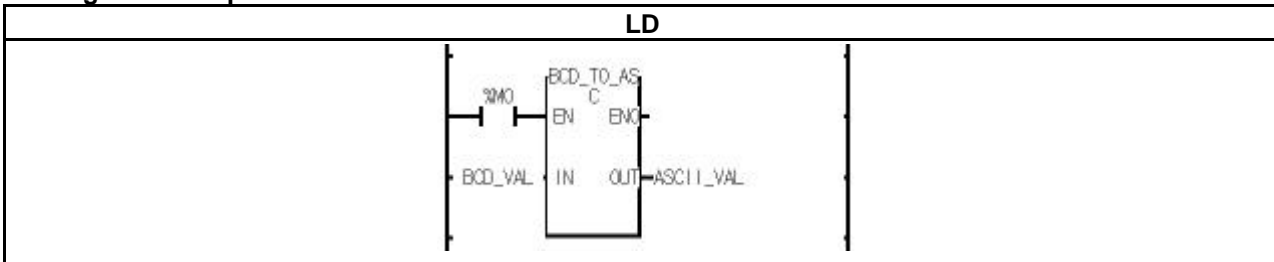
#### ■ Function

It converts two BCD data to two ASCII data.

#### ■ Error

If IN is not between 0 and 9, its output is 16#3030 ("00") and \_ERR/\_LER flags are set.

#### ■ Program example



(1) If the transition condition (%M0) is on, BCD\_TO\_ASC function is executed.

(2) If input BCD\_VAL (BYTE) = 16#85, output ASCII\_VAL (WORD) = 16#3835 = "85".

**BIT\_BYTE**

Combines 8 bits into BYTE

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

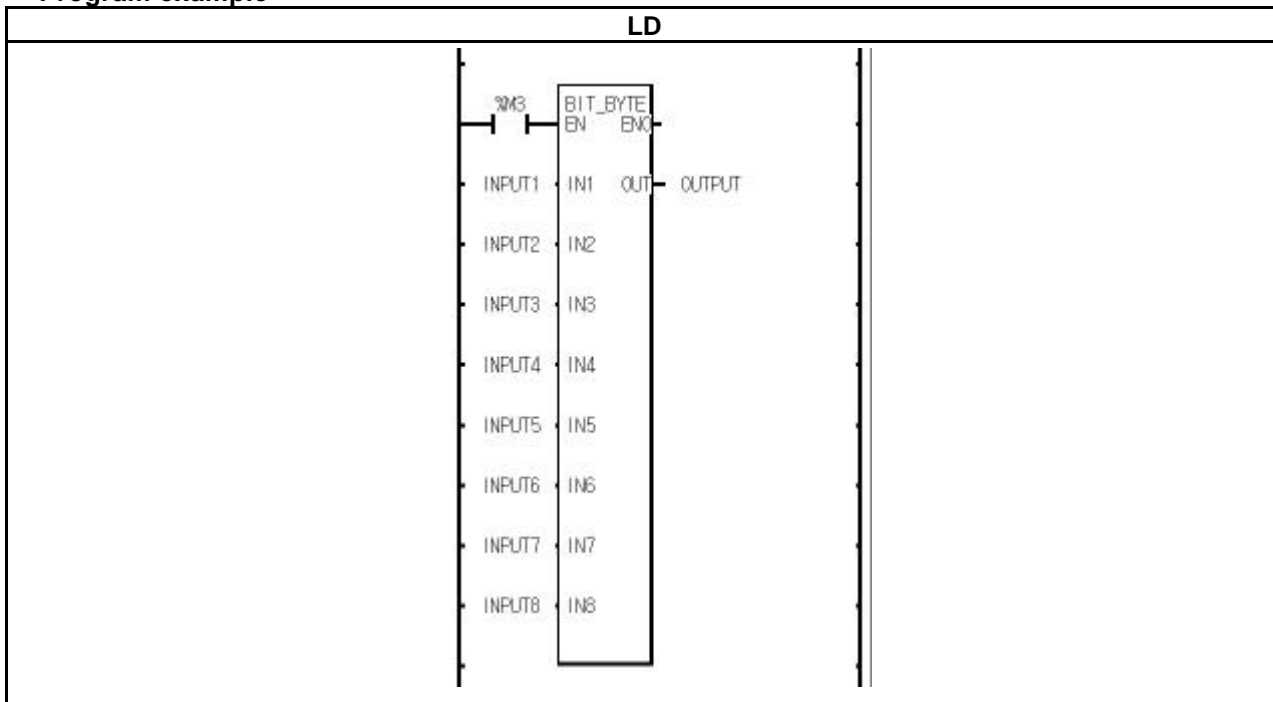
Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      IN1 ~ IN8: Bit input</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: Byte output</p>

■ **Function**

It combines 8 bits into one byte.

IN8: MSB (Most Significant Bit), IN1: LSB (Least Significant Bit)

■ **Program example**



(1) If the transition condition (%M3) is on, BIT\_BYTE function is executed.

(2) If 8 input are (from INPUT1 to INPUT 8) {0,1,1,0,1,1,0,0}, OUTPUT (BYTE) = 2#00110110.

## 8. Basic Function/Function Block Library

### BMOV \*\*\*

Moves part of a bit string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN : executes the function in case of 1.            IN1: String data having bit data to be combined            IN2: String data having bit data to be combined            IN1_P: Start bit position on IN1 set data            IN2_P: Start bit position on IN2 set data            N: Bit number to be combined</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1            OUT: Combined bit string data output</p>

#### ■ Function

If EN is 1, it takes N bits of IN1 starting from the IN1\_P bit and moves it to IN2 starting from IN2\_P bit.

If N1 = 1111 0000 1111 0000, IN2 = 0000 1010 1010 1111, IN1\_P = 4, IN2\_P = 8, N = 4, then output data is 0000 1111 1010 1111. Input data types are B (BYTE), W (WORD), D (DWORD), L (LWORD);

L (LWORD) are available for GM1/2. You can use one of functions ( ' ENCO\_B' , ' ENCO\_W' , ' ENCO\_D' , ' ENCO\_L' ) according to input data.

#### ■ Error

If IN1\_P and IN2\_P exceed the data range or N is negative or N bit of IN1\_P and IN2\_P exceeds the data range, \_ERR and \_LER flags are set.

#### ■ Program example

LD	IL
	<pre> LD          %M0 JMPN       LSB LD          SOURCE BMOV_W    IN1:=  CURRENT RESULT            IN2:=  DESTINE            IN1_P:= 0            IN2_P:= 8            N:=    4  ST          DESTINE LSB : </pre>

---

## 8. Basic Function/Function Block Library

- (1) If the transition condition (%M0) is on, BMOV\_W function is executed.
- (2) If input SOURCE = 2#0101 1111 0000 1010, DESTINE = 2#0000 0000 0000 0000, IN1\_P = 0, IN2\_P = 8, N = 4, then the result DESTINE is 2#0000 1010 0000 0000.

Input (IN1): SOURCE (WORD) = 16#5F0A  
(IN2): DESTINE (WORD) = 16#0000  
(IN1\_P) = 0  
(IN2\_P) = 8  
(N) = 4

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

↓ (BMOV\_W)

Output (OUT): DESTINE (WORD) = 16#0A00

0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

## 8. Basic Function/Function Block Library

### BSUM \*\*\*

Counts on-bit number of input

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: input data to detect ON bit</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1 OUT: Result data (sum of on-bit number)</p>

#### ■ Function

If EN is 1, it counts bit number of 1 among IN bit string and produces output OUT. Input data types are BYTE, WORD, DWORD, LWORD. LWORD is available only for GM1/2.

FUNCTION	IN type	Description
BSUM_BYTE	BYTE	You can select one of these functions according to input data.
BSUM_WORD	WORD	
BSUM_DWORD	DWORD	
BSUM_LWORD	LWORD	

#### □ Program example

LD	IL
	<pre>LD      %I0.0.0 JMPN   AAA LD      SWITCHS BSUM_WORD ST      ON_COUNT AAA:</pre>

(1) If the transition condition (%M0) is on, BSUM\_WORD function is executed.

(2) If input SWITCHS (WORD) = 2#0000 0100 0010 1000, then it counts on-bit number, 3. So the output ON\_COUNT (INT) = 3.



### BYTE\_BIT

Divides byte into 8 bits

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

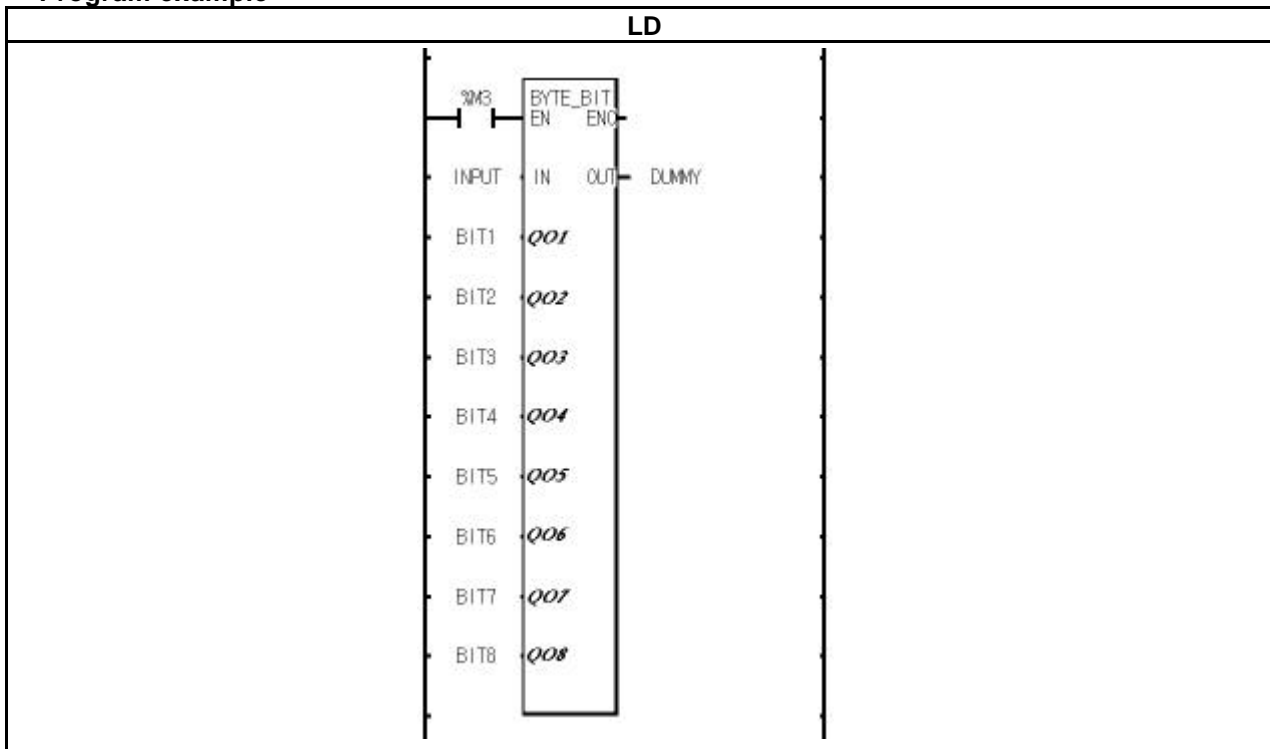
Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      IN: byte input</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: Dummy output</p> <p><b>In/Out</b>                      QO1~8: bit output</p>

■ **Function**

It divides one byte into 8 bits (QO1~QO2).

QO8: MSB (Most Significant Bit), QO1: LSB (Least Significant Bit)

■ **Program example**



(1) If the transition condition (%M0) is on, BYTE\_BIT function is executed.

(2) If INPUT = 16#AC = 2#10101100, it distributes INPUT from Q01 to Q08 in order.

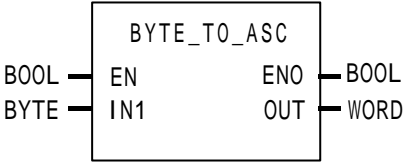
The order is 2#{0, 0, 1, 1, 0, 1, 0, 1}.

## 8. Basic Function/Function Block Library

### BYTE\_TO\_ASC

Converts byte into ASCII

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: BYTE input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: ASCII output</p>

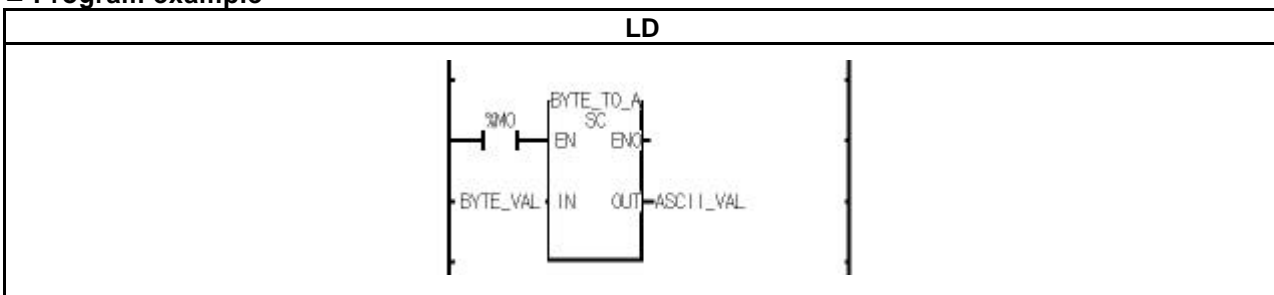
#### ■ Function

It converts 2-digit hexadecimal into two ASCII data.

Ex) 16#12 -> 3132

In case of 16#A~F, it produces ASCII data for character.

#### ■ Program example



(1) If the transition condition (%M0) is on, BYTE\_TO\_ASC function is executed.

(2) If input BYTE\_VAL (BYTE) = 16#3A, output ASCII\_VAL (WORD) = 16#3341 = '3', 'A'.

### BYTE\_WORD

Combines 2 bytes into WORD

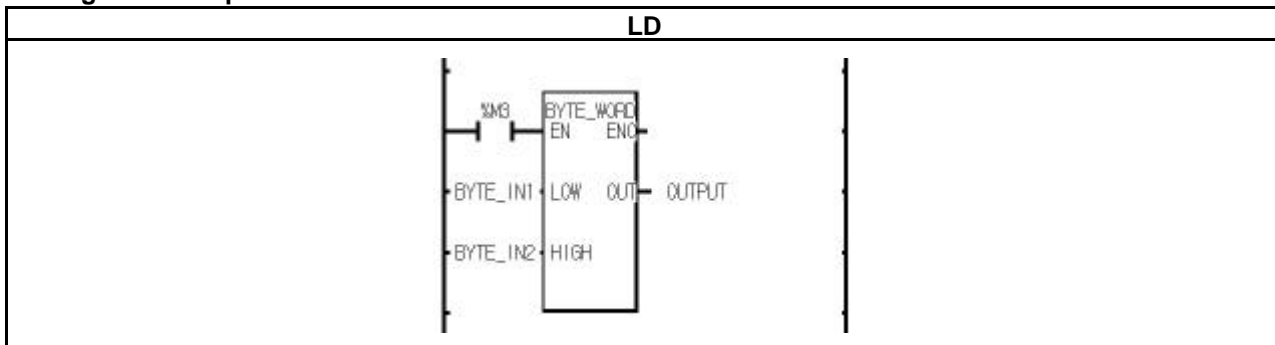
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      LOW: lower BYTE Input                      HIGH: upper BYTE Input</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: WORD output</p>

■ **Function**

It combines two bytes into one word.  
 LOW: lower byte input, HIGH: upper byte input

■ **Program example**



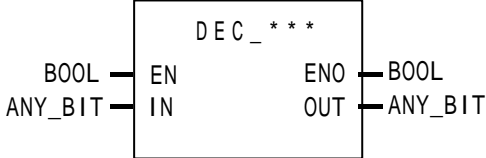
- (1) If the transition condition (%M3) is on, BYTE\_WORD function is executed.
- (2) If input BYTE\_IN1 = 16#56 and BYTE\_IN2 = 16#AD, output variable OUTPUT = 16#AD56.

## 8. Basic Function/Function Block Library

### DEC\_\*\*\*

Decrease IN data by 1 bit

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: input data to decrease</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1 OUT: result data</p>

#### □ Function

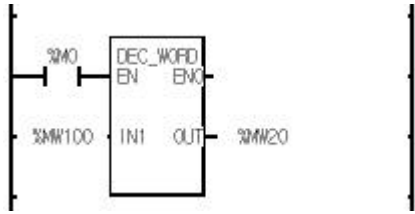
If EN is 1, it produces an output after decreasing bit-string data of IN by 1.

Even though the underflow occurs, an error won't occur and if the result is 16#0000, then the output result data is 16#FFFF.

Input data types are BYTE, WORD, DWORD and LWORD. LWORD is available only for GM1/2.

FUNCTION	IN/OUT type	Description
DEC_BYTE	BYTE	You can select one of these functions according to in/out data type.
DEC_WORD	WORD	
DEC_DWORD	DWORD	
DEC_LWORD	LWORD	

#### □ Program example

LD	IL
	<pre>LD      %M0 JMPN   KKK LD      %MW100 DEC_WORD ST      %MW20 KKK:</pre>

(1) If the transition condition (%M0) is on, DEC\_WORD function is executed.

(2) If input variable %MW100 = 16#0007 (2#0000 0000 0000 0111), output variable %MW20 = 16#0006 (2#0000 0000 0000 0110).

**DECO\_\*\*\***

Decodes the designated bit position

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: input data for decoding</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1 OUT: decoding result data</p>

**Function**

If EN is 1, it turns on 'the designated position bit of output bit-string data' according to the value of IN, and produces an output. Output data types are BYTE, WORD, DWORD and LWORD. LWORD is available only for GM1/2.

FUNCTION	OUT type	Description
DECO_BYTE	BYTE	You can select one of these functions according to output data type.
DECO_WORD	WORD	
DECO_DWORD	DWORD	
DECO_LWORD	LWORD	

**Error**

If input data is a negative number or bit position data is out of output-type range, (in case of DECO\_WORD, it's more than 16), then OUT is 0 and \_ERR/\_LER flags are set.

**Program example**

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      ON_POSITION DECO_WORD ST      RELAYS AAA:</pre>

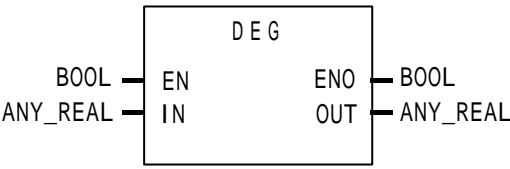
- (1) If the transition condition (%M0) is on, DECO\_WORD function is executed.
- (2) If ON\_POSITION (INT) = 5, then RELAYS (WORD) = 2#0000 0000 0010 0000.

## 8. Basic Function/Function Block Library

### DEG \*\*\*

Converts radian into degree

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

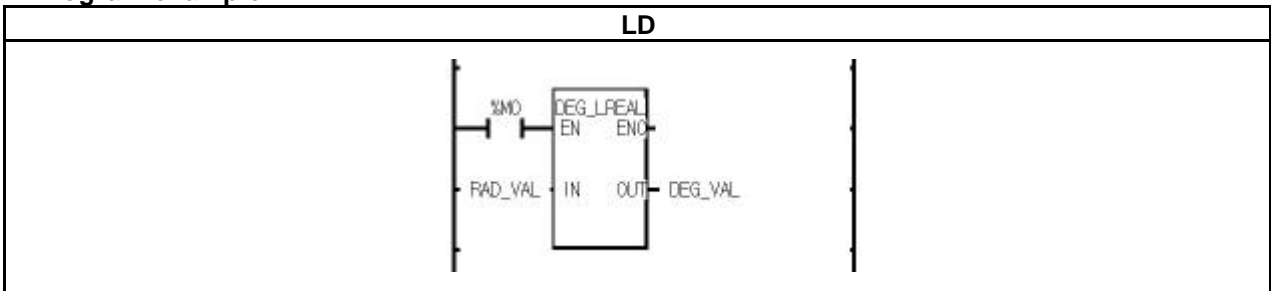
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: radian input</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: degree output</p>

#### ■ Function

It converts radian input into degree output.

Function	Input type	Output type	Description
DEG_REAL	REAL	REAL	It converts input (radian) into output (degree).
DEG_LREAL	LREAL	LREAL	

#### ■ Program example



(1) If the transition condition (%M0) is on, DEG\_LREAL function is executed.

(2) If input variable RAD\_VAL = 1.0, then output variable DEG\_VAL = 5.7295779513078550e+001.

**DIS\_\*\*\***

Data distribution

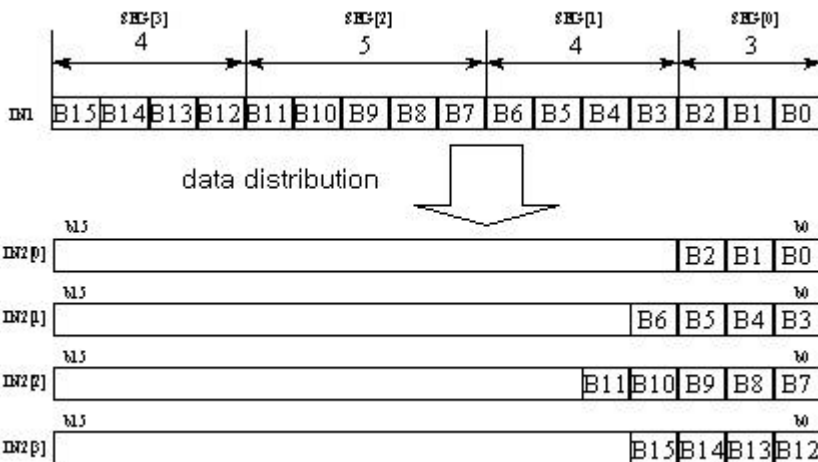
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      IN1: input data                      SEG: designated bit array for data distribution</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: Dummy Output</p> <p><b>In/Out</b>                      IN2: distributed WORD-array Output</p>

■ **Function**

It distributes input data over IN2 after segmenting input data by bit number set by SEG.

Function	Input type	Description
DIS_BYTE	BYTE	It segments IN1 input by bit number set by SEG and produces IN2 array.
DIS_WORD	WORD	
DIS_DWORD	DWORD	
DIS_LWORD	LWORD	



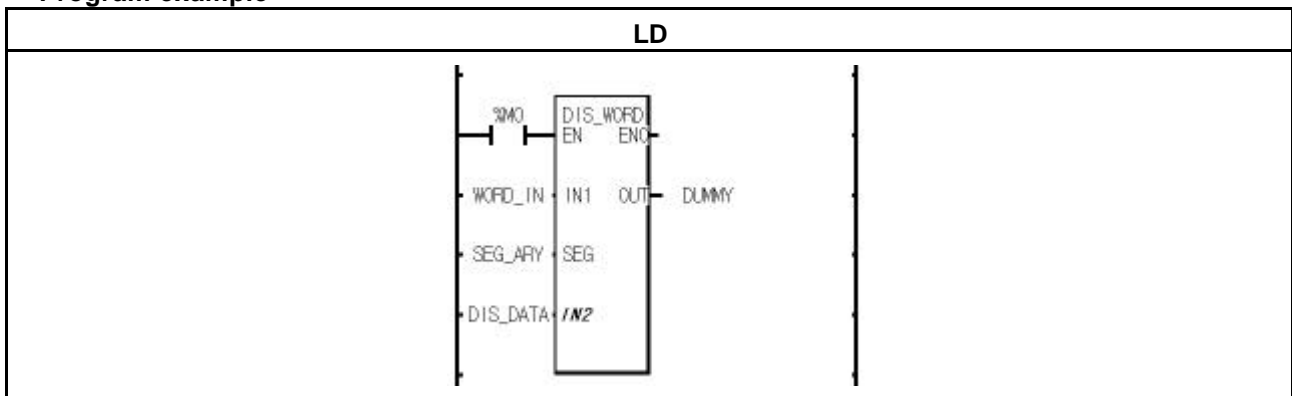
■ **Error**

If the sum of designated number of SEG exceeds input variable bit number, \_ERR/\_LER flags are set.

## 8. Basic Function/Function Block Library

---

### ■ Program example



(1) If the transition condition (%M0) is on, DIS\_WORD function is executed.

(2) If input variable WORD\_IN = 16#3456, SEG\_ARY = {3, 4, 5, 4}, then, output variable DIS\_DATA is:

DIS\_DATA[0] = 16#0006

DIS\_DATA[1] = 16#000A

DIS\_DATA[2] = 16#0008

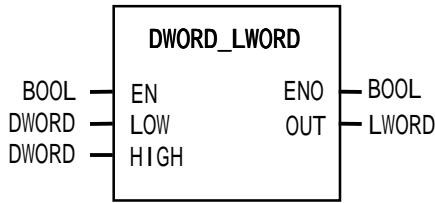
DIS\_DATA[3] = 16#0003



### DWORD\_LWORD

Combines two DWORD data into LWORD

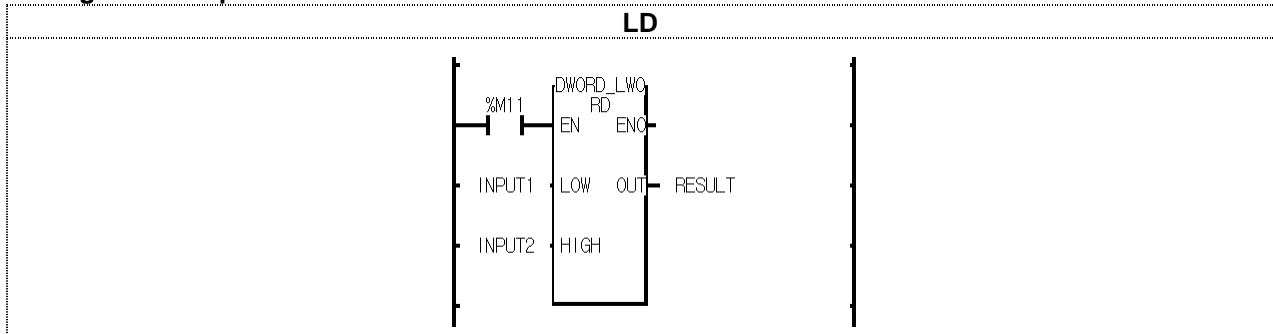
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      LOW: lower DWORD Input                      HIGH: upper DWORD Input</p> <p><b>Output</b>                      ENO: without an error, it will be 1.                      OUT: LWORD Output</p>

■ **Function**

It combines 2 DWORD data into one LWORD data.  
 LOW: lower DWORD Input, HIGH: upper DWORD Input

■ **Program example**



- (1) If the transition condition (%M11) is on, DWORD\_LWORD function is executed.
- (2) If input variable INPUT1 = 16#1A2A3A4A5A6A7A8A and INPUT2 = 16#8C7C6C5C4C3C2C1C, then, output variable RESULT = 16#8C7C6C5C4C3C2C1C1A2A3A4A5A6A7A8A.

## 8. Basic Function/Function Block Library

### DWORD\_WORD

Divides DWORD into 2 WORD data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

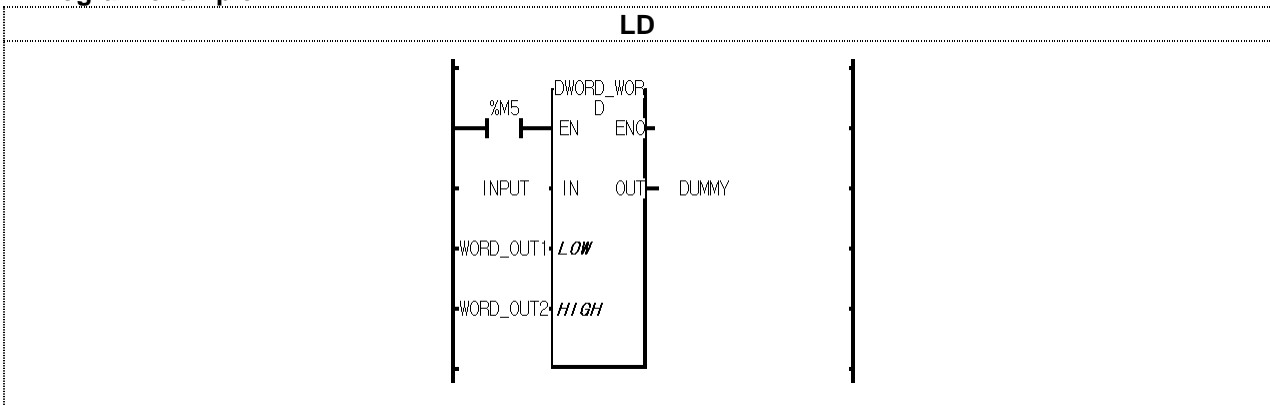
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: DWORD Input</p> <p><b>Output</b>            ENO: without an error, it will be 1.            OUT: Dummy Output</p> <p><b>In/Out</b>            LOW: lower WORD Output            HIGH: upper WORD Output</p>

#### ■ Function

It divides one DWORD into two WORD data.

LOW: lower WORD Output, HIGH: upper WORD Output

#### ■ Program example

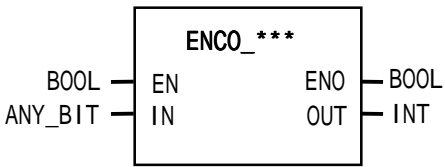


- (1) If the transition condition (%M5) is on, DWORD\_WORD function is executed.
- (2) If input variable INPUT = 16#11223344AABBCCDD, then,  
 WORD\_OUT1 = 16#AABBCCDD and WORD\_OUT2 = 16#11223344.

**ENCO\_\*\*\***

Encodes the on-bit position of IN

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: input data to be encoded</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1 OUT: result data after encoding</p>

■ **Function**

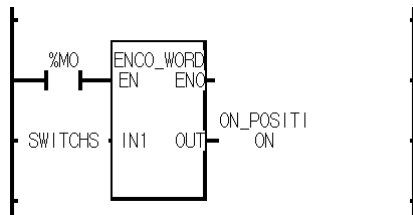
If EN is 1, the output is the highest on-bit position among IN bit string. Input data types are BYTE, WORD, DWORD and LWORD. LWORD is available only for GM1/2.

FUNCTION	IN type	Description
ENCO_BYTE	BYTE	You can select one of these functions according to the input data type.
ENCO_WORD	WORD	
ENCO_DWORD	DWORD	
ENCO_LWORD	LWORD	

■ **Error**

\_ERR and \_LER flags are set and OUT is -1 if no bit is 1.

■ **Program example**

LD	IL
	<pre>LD      %M0 JMPN   AAA LD      SWITCHS ENCO_W ST      ON_POSITION AAA:</pre>

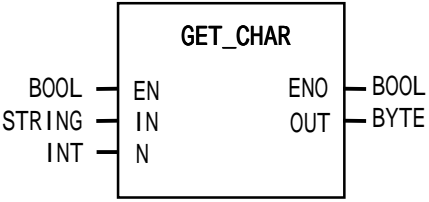
- (1) If the transition condition (%M0) is on, ENCO\_WORD function is executed.
- (2) If SWITCHS (WORD) = 2#0000 1000 0000 0010, then, the highest on-bit position is 11. Therefore, output ON\_POSITION (INT) is '11'.

## 8. Basic Function/Function Block Library

### GET\_CHAR

Gets one character from a character string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
 <p>The diagram shows a rectangular function block labeled 'GET_CHAR'. On the left side, there are three input terminals: 'EN' (labeled 'BOOL'), 'IN' (labeled 'STRING'), and 'N' (labeled 'INT'). On the right side, there are two output terminals: 'ENO' (labeled 'BOOL') and 'OUT' (labeled 'BYTE').</p>	<p><b>Input</b>            EN: executes the function in case of 1.            IN: STRING input            N: position in a character STRING</p> <p><b>Output</b>            ENO: without an error, it will be 1.            OUT: Byte Output</p>

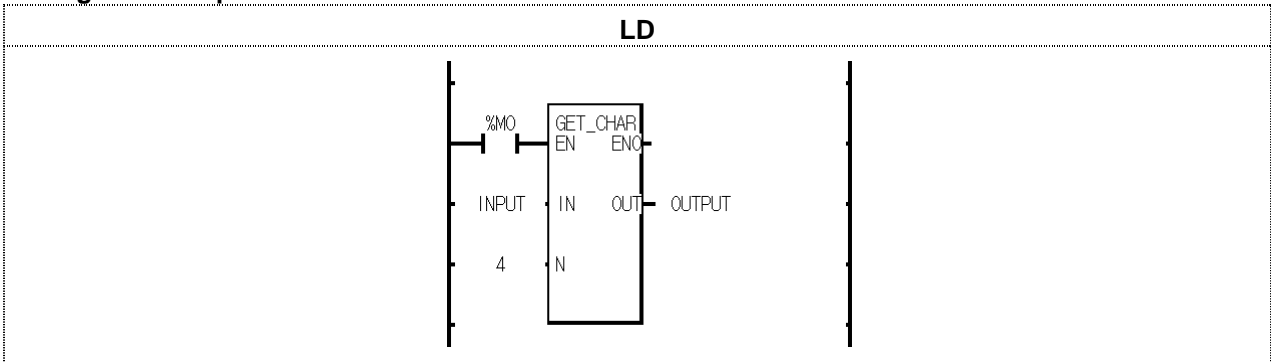
#### ■ Function

It extracts one byte from a character STRING starting from N.

#### ■ Error

\_ERR/\_LER flags are set if N exceeds the number of byte in STRING.  
 If an error occurs, the output is 16#00.

#### ■ Program example



(1) If the transition condition (%M0) is on, GET\_CHAT function is executed.

(2) When input INPUT (STRING) = "LG GLOFA PLC", if you extract 4<sup>th</sup> character from this string, output variable OUTPUT is 16#47 ("G").

**INC\_\*\*\***

Increase IN data by 1

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: Input data to increase</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1 OUT: result data after increase</p>

■ **Function**

If EN is 1, it increases IN bit string data by 1 and produces an output.  
An error does not occur when there's an overflow; the result is 16#0000 in case of 16#FFFF.  
Input data types are BYTE, WORD, DWORD and LWORD. LWORD is available only for GM1/2.

FUNCTION	IN/OUT type	Description
INC_BYTE	BYTE	You can select one of these functions according to the data type.
INC_WORD	WORD	
INC_DWORD	DWORD	
INC_LWORD	LWORD	

■ **Program example**

LD	IL
	<pre>LD      %M0 JMPN   BBB LD      %MW100 INC_WORD ST      %MW100 AAA:</pre>

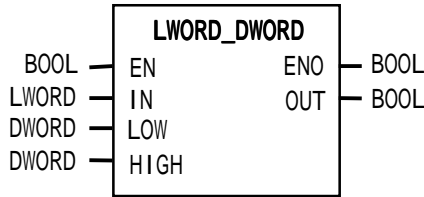
- (1) If the transition condition (%M0) is on, INC\_WORD function is executed.
- (2) If input variable %MW100 = 16#0007 (2#0000 0000 0000 0111), then output variable %MW100 = 16#0008 (2#0000 0000 0000 1000).

## 8. Basic Function/Function Block Library

### LWORD\_DWORD

Divides LWORD into two DWORD data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

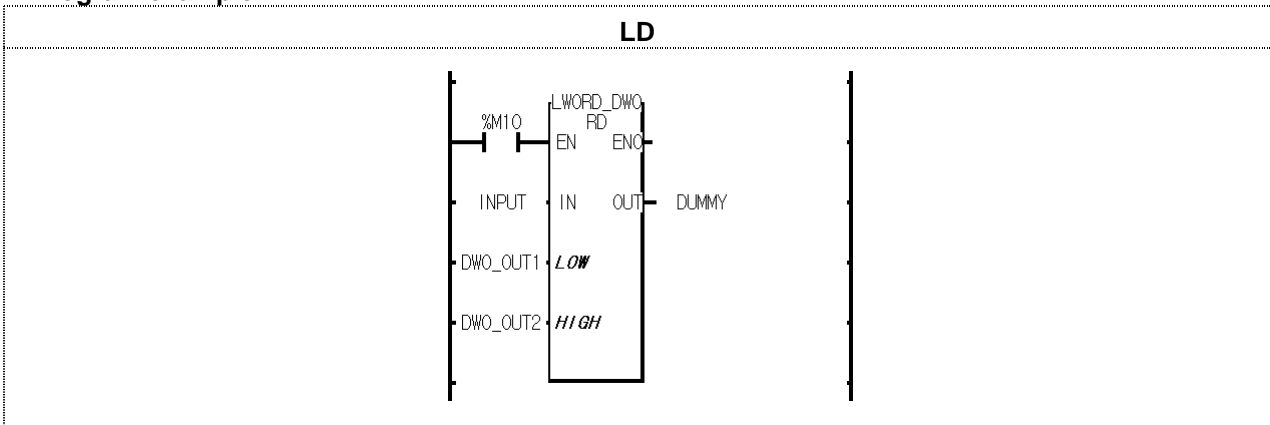
Function	Description
 <p>The diagram shows a rectangular function block labeled 'LWORD_DWORD'. On the left side, there are four input terminals: 'EN' (labeled 'BOOL'), 'IN' (labeled 'LWORD'), 'LOW' (labeled 'DWORD'), and 'HIGH' (labeled 'DWORD'). On the right side, there are two output terminals: 'ENO' (labeled 'BOOL') and 'OUT' (labeled 'BOOL').</p>	<p><b>Input</b>            EN: executes the function in case of 1.            IN: LWORD Input</p> <p><b>Output</b>            ENO: without an error, it will be 1.            OUT: Dummy Output</p> <p><b>In/Out</b>            LOW: lower DWORD Output            HIGH: upper DWORD Output</p>

#### ■ Function

It divides one LWORD into two DWORD data.

LOW: lower DWORD Output, HIGH: upper DWORD Output

#### ■ Program example

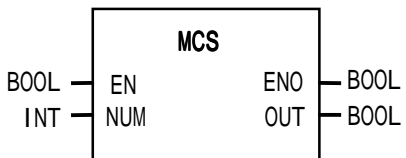


- (1) If the transition condition (%M10) is on, LWORD\_DWORD function is executed.
- (2) If the input variable INPUT = 16#AAAABBBBCCCCDDDDABCDABCDABCDABCD, then,  
 DWO\_OUT1 = 16#ABCDABCDABCDABCD  
 DWO\_OUT2 = 16#AAAABBBBCCCCDDDD.

## MCS

Master Control
----------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1.            NUM: Nesting (0~15)</p> <p><b>Output</b></p> <p>ENO: If MCS is executed, it will be 1            OUT: Dummy (always 0)</p>

### ■ Function

If EN is on, MCS function is executed and the program between MCS and MCSCLR function is normally executed.

If EN is off, the program between MCS and MCSCLR function is executed as follows:

Instruction	Description
Timer	Current value (CV) becomes 0 and the output (Q) becomes off.
Counter	Output (Q) becomes off and CV retains its present state.
Coil	All becomes off.
Negated coil	All becomes off.
Set coil, reset coil	All retains its current value.
Function, function block	All retains its current value.

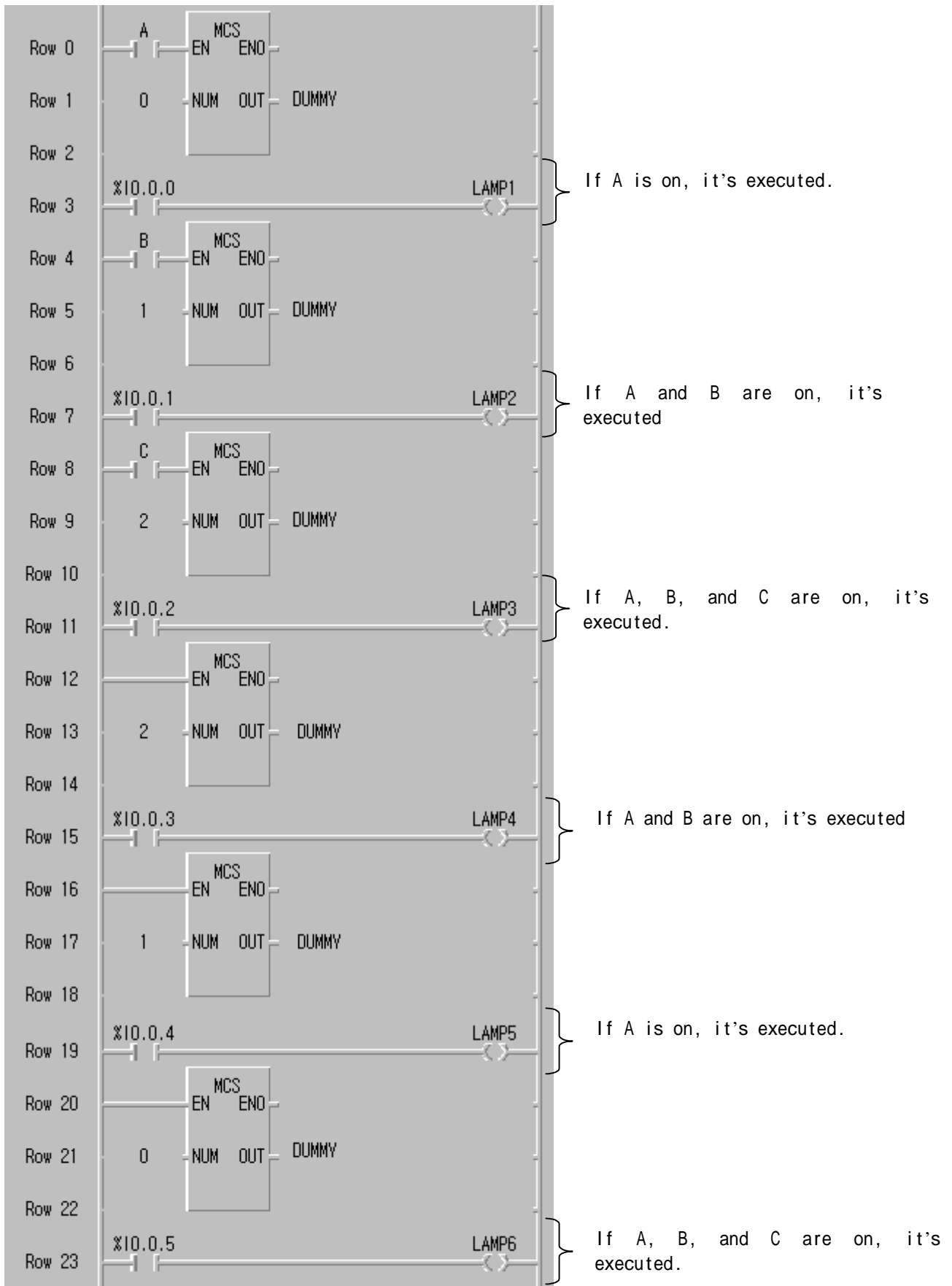
Even when EN is off, scan time is not shortened because the instructions between MCS and MCSCLR function are executed as the above.

Nesting is available in MCS. That is to say, Master Control is divided by Nesting (NUM). You can set up Nesting (NUM) from 0 to 15 and if you set it more than 16, MCS is not executed normally.

Note: if you use MSC without 'MCSCLR', MCS function is executed till the end of the program.

## 8. Basic Function/Function Block Library

### ■ Program example





## MCSCLR

Master Control Clear

Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1.            NUM: Nesting (0~15)</p> <p><b>Output</b></p> <p>ENO: if MCSCLR is executed, it will be 1            OUT: if MCSCLR is executed, it will be 1</p>

### ■ Function

It clears Master Control instruction. And it indicates the end of Master Control.

If MCSCLR function is executed, it clears all the MCS instructions which are less than or equal to Nesting (NUM).

\* There's no contact before MCSCLR function.

### ■ Program example

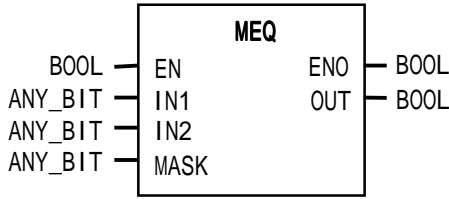
Refer to the MCS function example.

## 8. Basic Function/Function Block Library

### MEQ\_\*\*\*

Masked Equal

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
 <pre> graph LR     EN[BOOL] --&gt; ENO[BOOL]     IN1[ANY_BIT] --&gt; OUT[BOOL]     IN2[ANY_BIT] --&gt; OUT     MASK[ANY_BIT] --&gt; OUT     subgraph MEQ         ENO         OUT     end         </pre>	<p><b>Input</b></p> <p>EN: executes the function in case of 1.  IN1: Input1  IN2: Input2  MASK: input data to mask</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1.  OUT: when equal, it will be 1</p>

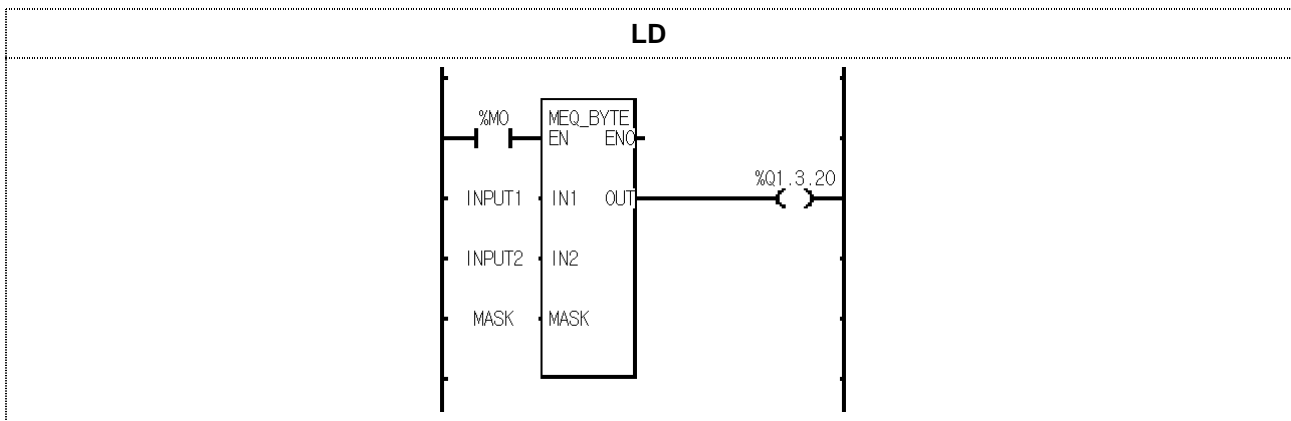
#### ■ Function

It compares whether two input variables are equal after masking. If it masks an 8-bit variable with 2#11111100, then, lower 2 bits are excluded when it compares input values.

It's available to see whether or not specific bits are on in a variable. For example, in case of comparing 8-bit variables, IN1 is an input variable, IN2 is 16#FF, and MASK for masking is a bit array 2#00101100. If IN1 and IN2 after masking are equal, then output OUT is 1.

Function	Input type	Description
MEQ_BYTE	BYTE	It compares whether two variables are equal after making.
MEQ_WORD	WORD	
MEQ_DWORD	DWORD	
MEQ_LWORD	LWORD	

### ■ Program example



(1) If the transition condition (%M0) is on, MEQ\_BYTE function is executed.

(2) Input variable INPUT1 (BYTE) = 2#01011100

INPUT2 (BYTE) = 2#01110101

MASK (BYTE) = 2#11010110

Then, the comparing bits of input variables after masking are as follows:

INPUT1 (BYTE) = 2#01010100

INPUT2 (BYTE) = 2#01010100

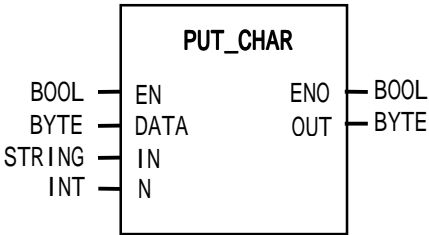
INPUT1 and INPUT2 are equal, therefore, output contact %Q1.3.20 is on.

## 8. Basic Function/Function Block Library

### PUT\_CHAR

Puts a character in a string

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <ul style="list-style-type: none"> <li>EN: executes the function in case of 1.</li> <li>DATA: Byte input to insert a string</li> <li>IN: string input</li> <li>N: setting position in a string</li> </ul> <p><b>Output</b></p> <ul style="list-style-type: none"> <li>ENO: without an error, it will be 1.</li> <li>OUT: string output</li> </ul>

#### ■ Function

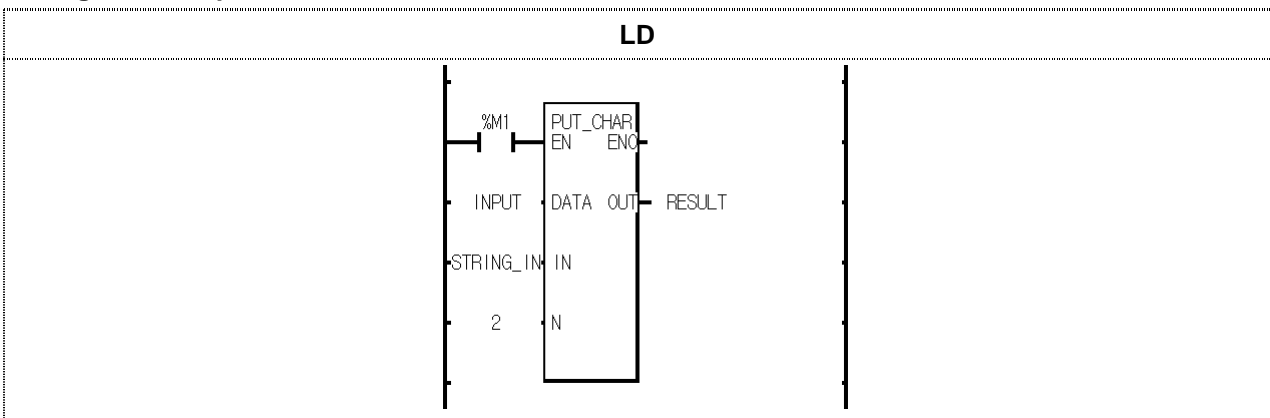
It overwrites one byte input on a specific position (N) string.

#### ■ Error

If N value exceeds a byte number of a string, `_ERR/_LER` flags are set.

If an error occurs, the output is 16#00.

#### ■ Program example




(1) If the transition condition (%M1) is on, PUT\_CHAR function is executed.

(2) If input variable INPUT = 16#41 ("A") and STRING\_IN = "TOKEN", and N = 2, then, output RESULT is "TAKEN".

**RAD\_\*\*\***

Converts degree into radian

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

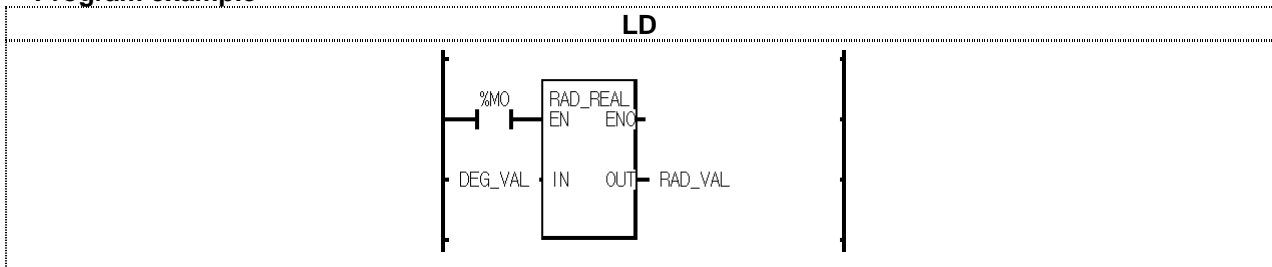
Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      IN: degree Input</p> <p><b>Output</b>                      ENO: without an error, it will be 1.                      OUT: radian output</p>

■ **Function**

It converts a degree value ( ° ) into a radian value.  
 If the degree is over 360°, its converts normally.  
 For example, if input is 370°, output is 370° - 360° = 10°.

Function	Input type	Output type	Description
RAD_REAL	REAL	REAL	It converts a degree value ( ° ) into a radian value.
RAD_LREAL	LREAL	LREAL	

■ **Program example**



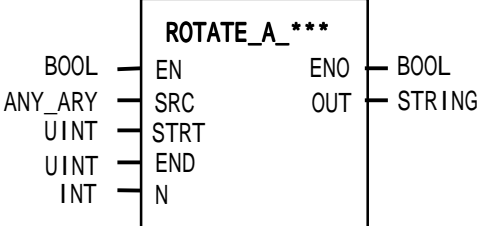
- (1) If the transition condition (%M0) is on, RAD\_REAL function is executed.
- (2) If input variable DEG\_VAL = 127( °), its output RAD\_VAL = 2.21656823.

## 8. Basic Function/Function Block Library

### ROTATE\_A\_\*\*\*

Rotates array elements

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

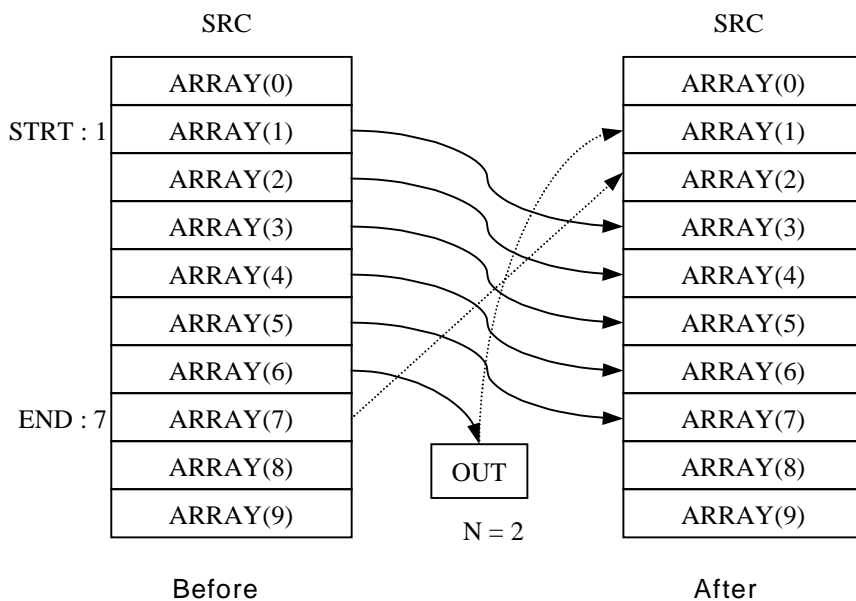
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            N: element number to rotate            STRT: starting position to rotate in an array block            END: ending position to rotate in an array block</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: overflowing data</p> <p><b>In/Out</b>            SRC: array block to rotate</p>

#### ■ Function

It rotates designated elements of an array block in the chosen direction.

Setting:

- Scope: STRT and END set a data array to rotate
- Rotation direction and time: rotates N times in the chosen direction set by STRT and END (STRT → END)
- Input data setting: fills an empty element after rotation with Input data (IN)
- Output: the result is written at ANY\_ARY designated by **SRC**, and the data to rotate from END to STRT is written at OUT.



## 8. Basic Function/Function Block Library

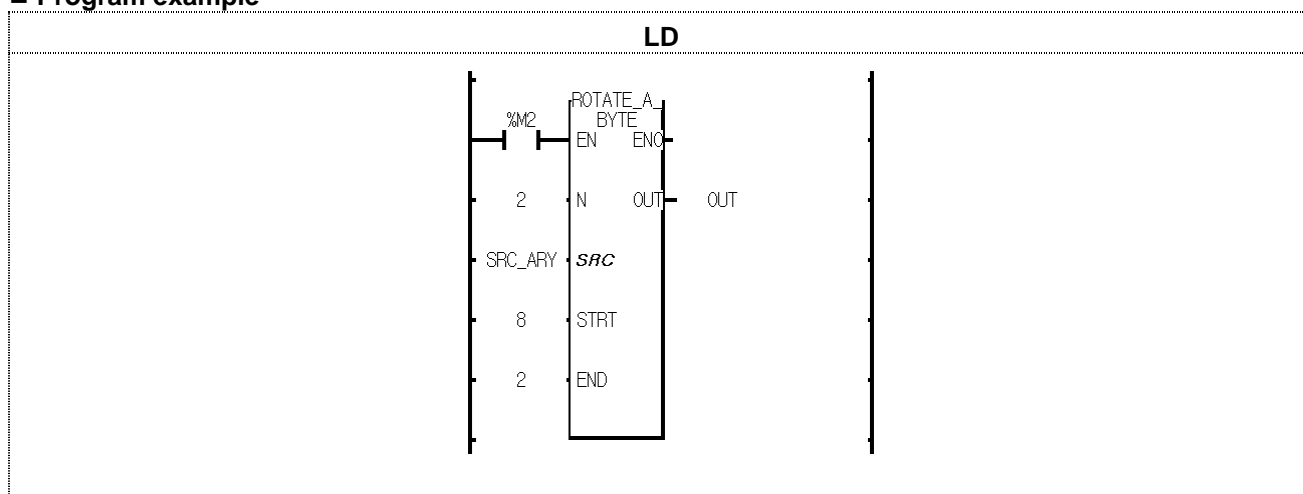
Function	In/Out array type	Description
ROTATE_A_BOOL	BOOL	It rotates designated elements of an array block in the chosen direction.
ROTATE_A_BYTE	BYTE	
ROTATE_A_WORD	WORD	
ROTATE_A_DWORD	DWORD	
ROTATE_A_LWORD	LWORD	
ROTATE_A_SINT	SINT	
ROTATE_A_INT	INT	
ROTATE_A_DINT	DINT	
ROTATE_A_LINT	LINT	
ROTATE_A_USINT	USINT	
ROTATE_A_UINT	UINT	
ROTATE_A_UDINT	UDINT	
ROTATE_A_ULINT	ULINT	
ROTATE_A_REAL	REAL	
ROTATE_A_LREAL	LREAL	
ROTATE_A_TIME	TIME	
ROTATE_A_DATE	DATE	
ROTATE_A_TOD	TOD	
ROTATE_A_DT	DT	

### ■ Error

If STRT or END exceed the range of SRC array element, \_ERR/\_LER flags are set.

If an error occurs, there's no change in SRC and output OUT is the initial value of each variable type (i.e. INT=0, TIME=T#0S).

### ■ Program example



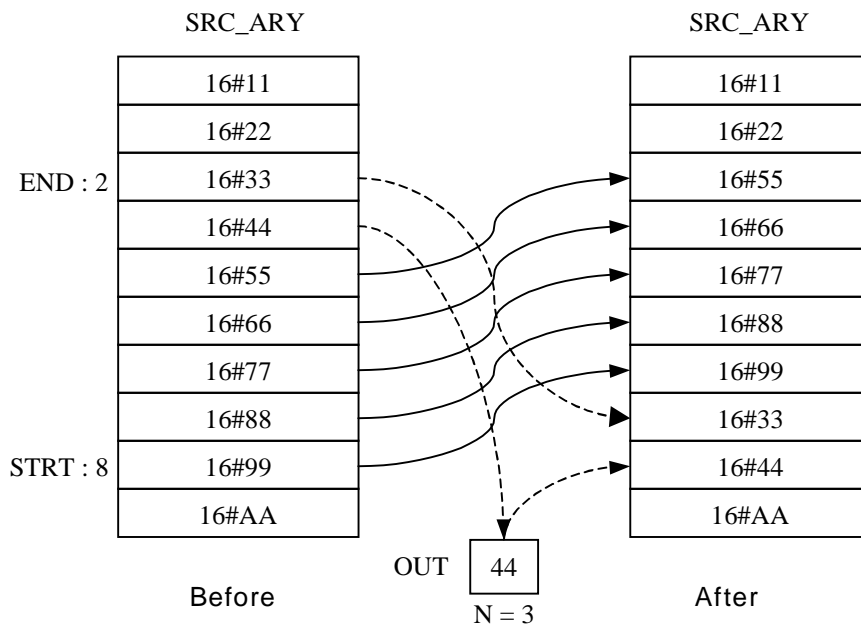
(1) If input condition (%M2) is on, ROTATE\_A\_BYTE function is executed.

(2) It rotates designated elements (from 2nd to 8th elements) of SRC\_ARY in the chosen direction set by STRT and END (from index 8 to index 2): refer to the diagram on the opposite page.

(3) The overflowing data (16#44) is written at OUT.

## 8. Basic Function/Function Block Library

---

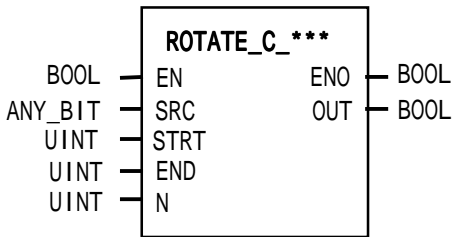




### ROTATE\_C\_\*\*\*

Rotates a designated bit array of SRC bit arrays

Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      STRT: starting bit position of SRC bit array to rotate                      END: ending bit position of SRC bit array to rotate                      N: bit number to shift</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: carry output</p> <p><b>In/Out</b>  <b>SRC</b>: variable for rotation</p>

■ **Function**

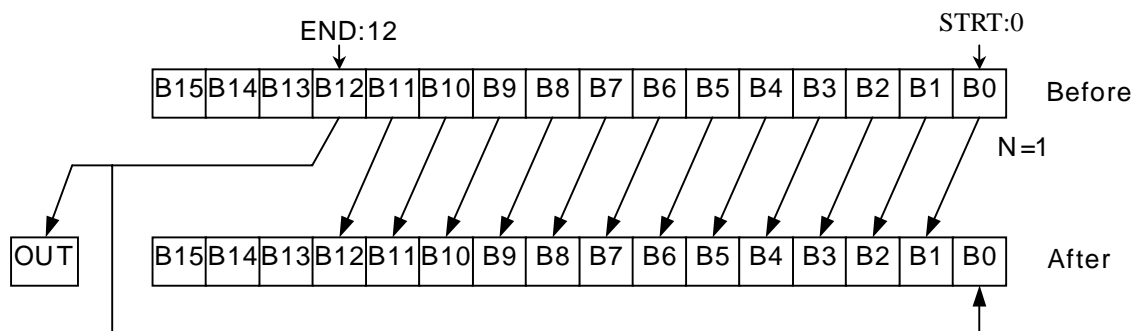
It rotates a designated bit array of SRC bit arrays in the chosen direction.

Setting:

- Scope: STRT and END set a bit data to rotate.

- Rotation direction and time: rotates N times in the chosen direction set by STRT and END (STRT → END)

- Output: the result is written at ANY\_ARY designated by SRC, and the data to rotate from END to STRT is written at OUT.



Function	SRC type	Description
ROTATE_C_BYTE	BYTE	It rotates a designated bit array of SRC bit arrays N times in the chosen direction.
ROTATE_C_WORD	WORD	
ROTATE_C_DWORD	DWORD	
ROTATE_C_LWORD	LWORD	

■ **Error**

If STRT or END exceed the bit number of SRC variable type, \_ERR and \_LER flags are set. There's no change in SRC data.



### RTC\_SET

Writes Time data

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>REQ: executes the function with rising pulse input            DATA: TIME data to input</p> <p><b>Output</b></p> <p>DONE: without an error, it will be 1            STAT: If an error occurs, an error code is written</p>

**Function**

It writes RTC data to Clock Device with a rising pulse input.

Variable	Content	Example	Variable	Content	Example
DATA[0]	Last 2-digit number of years	16#01	DATA[4]	Minutes	16#30
DATA[1]	Months	16#03	DATA[5]	Seconds	16#45
DATA[2]	Dates	16#15	DATA[6]	Days	16#03
DATA[3]	Hours	16#18	DATA[7]	First 2-digit number of years	16#20

\* The above example is "2001-03-15 18:30:45, Thursday".

\* Days are indicated as follows: Mon (0), Tue (1), Wed (2), Thu (3), Fri (4), Sat (5), Sun (6).

The above DATA variables are declared as array Byte variables and set as BCD data.

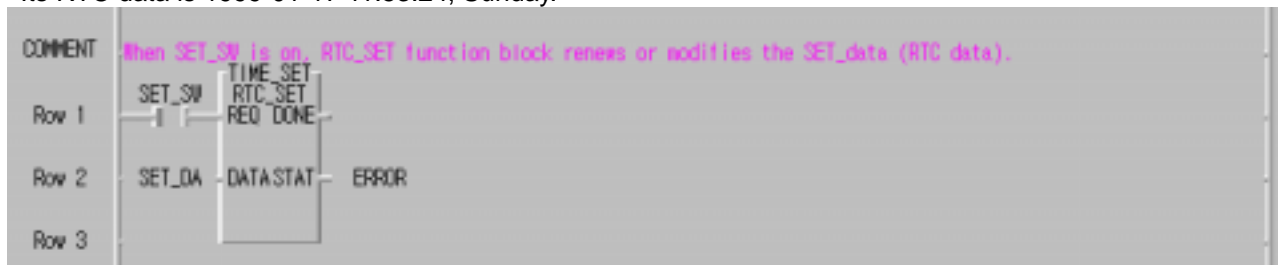
**Error**

If CPU does not support RTC function or RTC data is out of range, the output is 0 and the error code is written at STAT.

Error code	Description
00	No error
01	No RTC module installed. * GM6: GM6-CPUB and GM6-CPUC support RTC. * GM7: G7E-RTCA should be installed.
02	Wrong RTC data. Example: 14 (Months) 32 (Dates) 25 (Hours) * Modify RTC data.

**Program example**

Its RTC data is 1999-01-17 11:53:24, Sunday.

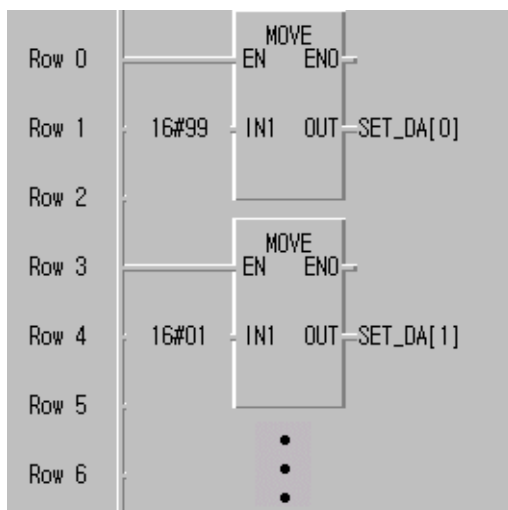


(1) When SET\_SW is on, RTC\_SET function block renews or modifies the SET\_data (RTC data).

(2) Variable setting is shown as below.



(3) You can set each TIME data using MOVE function.



## 8. Basic Function/Function Block Library

(4) Use the following flags to read RTC data.  
e.g. 1998-12-22 19:37:46, Tuesday

Flag	Type	Description	Data
_RTC_TOD	TOD	Current time of RTC	TOD#19:37:46
_RTC_WEEK	UINT	Current day of RTC *(0: Mon, 1: Tue, 2: Wed, 3: Thu, 4: Fri, 5: Sat, 6: Sun)	1
_RTC_DATE	DATE	Current date of RTC (1984-01-01 ~ 2083-12-31)	D#1998-12-22
_RTC_ERR	BOOL	When RTC data is wrong, it is 1.	0
_RTC_TIME[n] * n: 0 ~ 7	BCD	BCD data of current time of RTC _RTC_TIME[0]: Last 2-digit number of years _RTC_TIME[1]: Months _RTC_TIME[2]: Dates _RTC_TIME[3]: Hours _RTC_TIME[4]: Minutes _RTC_TIME[5]: Seconds _RTC_TIME[6]: Days _RTC_TIME[7]: First 2-digit number of years Days ( 0: Mon, 1: Tue, 2: Wed, 3: Thu, 4: Fri, 5: Sat, 6: Sun)	_RTC_TIME[0]: 16#98 _RTC_TIME[1]: 16#12 _RTC_TIME[2]: 16#22 _RTC_TIME[3]: 16#19 _RTC_TIME[4]: 16#37 _RTC_TIME[5]: 16#46 _RTC_TIME[6]: 16#1 _RTC_TIME[7]: 16#19

## 8. Basic Function/Function Block Library

### SEG

Converts BCD or HEX into 7 segment display code

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b></p> <p>EN: executes the function in case of 1. IN: Input data to convert into 7 segment code</p> <p><b>Output</b></p> <p>ENO: without an error, it will be 1. OUT: result data converted into 7 segment data</p>

#### ■ Function

If EN is 1, it converts BCD or HEX (hexadecimal) of IN into 7 segment display code as below and produces output OUT. If an input is BCD type, it is available to display a number between 0000 and 9999. And in case of HEX input, it's available to display a number between 0000 and FFFF on 4-digit 7 segment display.

#### Display example

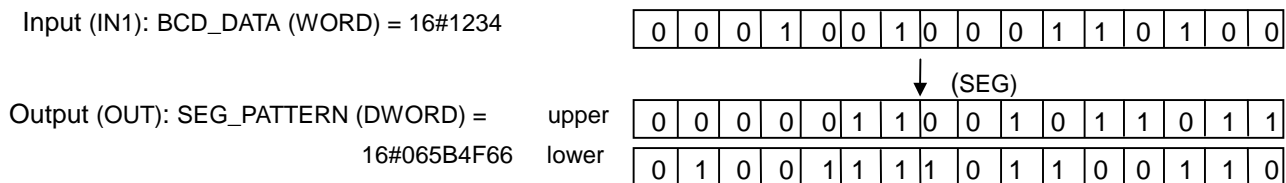
- 1) 4-digit BCD -> 4-digit 7 segment code: use SEG function
- 2) 4-digit HEX -> 4-digit 7 segment code: use SEG function
- 3) INT -> 4-digit BCD-type 7 segment code: use INT\_TO\_BCD function first and SEG function
- 4) INT -> 4-digit HEX-type 7 segment code: use INT\_TO\_WORD function first and SEG function
- 5) When 7 segment display digits are more than 4,
  - A) in case of BCD, HEX type, use SEG function, after dividing them into 4 digits.
  - B) INT -> 8-digit BCD-type 7 segment code:  
Divide INT by 10,000 and convert 'quotient' and 'remainder' into upper/lower 4-digit 7 segment code using INT\_TO\_BCD and SEG function.

#### ■ Program example

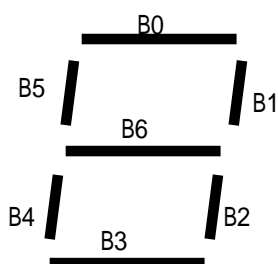
LD	IL
	<pre>LD      %M0 JMPN   BBB LD      BCD_DATA SEG ST      SEG_PATTERN BBB:</pre>

## 8. Basic Function/Function Block Library

- (1) If the transition condition (%M0) On SEGfunction is executed.  
 (2) If input variable BCD\_DATA (WORD) = 16#1234,  
 the output is '2#00000110\_01011011\_01001111\_01100110' which is displayed as a 7 segment code (1234)  
 and written at SEG\_PATTERN (DWORD).



### 7 segment configuration



### Conversion table for 7 segment code

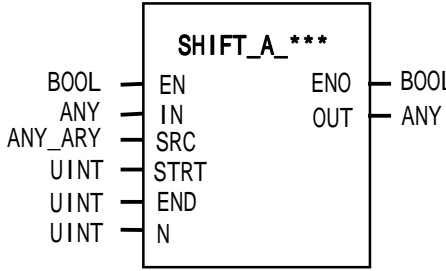
Input (BCD)	Input (HEX)	INT	Output								Display Data
			B7	B6	B5	B4	B3	B2	B1	B0	
0	0	0	0	0	1	1	1	1	1	1	<b>0</b>
1	1	1	0	0	0	0	0	1	1	0	<b>1</b>
2	2	2	0	1	0	1	1	0	1	1	<b>2</b>
3	3	3	0	1	0	0	1	1	1	1	<b>3</b>
4	4	4	0	1	1	0	0	1	1	0	<b>4</b>
5	5	5	0	1	1	0	1	1	0	1	<b>5</b>
6	6	6	0	1	1	1	1	1	0	1	<b>6</b>
7	7	7	0	0	1	0	0	1	1	1	<b>7</b>
8	8	8	0	1	1	1	1	1	1	1	<b>8</b>
9	9	9	0	1	1	0	1	1	1	1	<b>9</b>
	A	10	0	1	1	1	0	1	1	1	<b>A</b>
	B	11	0	1	1	1	1	1	0	0	<b>B</b>
	C	12	0	0	1	1	1	0	0	1	<b>C</b>
	D	13	0	1	0	1	1	1	1	0	<b>D</b>
	E	14	0	1	1	1	1	0	0	1	<b>E</b>
	F	15	0	1	1	1	0	0	0	1	<b>F</b>

## 8. Basic Function/Function Block Library

### SHIFT\_A\_\*\*\*

Shifts array elements

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

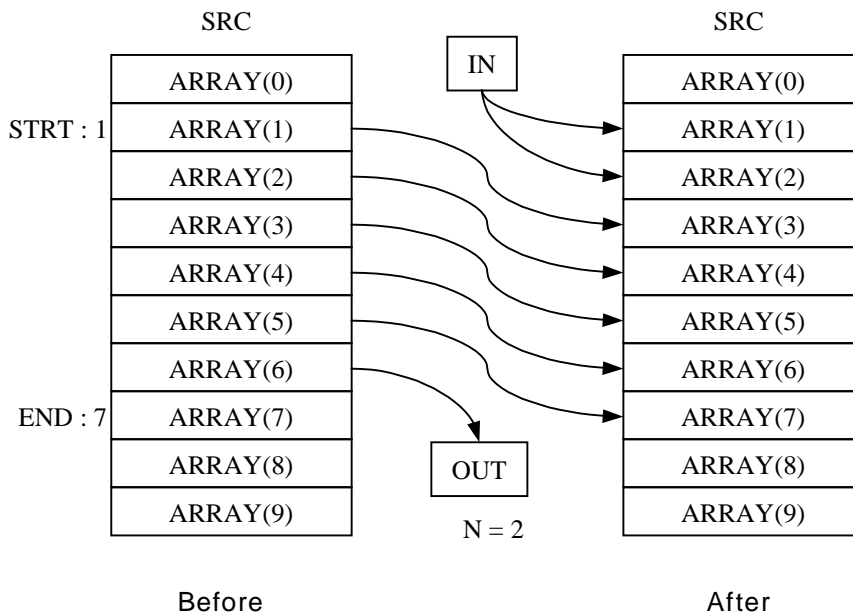
Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: Input data to empty element after shifting            N: number to shift            STRT: starting position to shift in an array block            END: ending position to shift in an array block</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: overflowing data</p> <p><b>In/Out</b>            SRC: array block to shift</p>

#### ■ Function

It shifts designated elements of an array block in the chosen direction.

Setting

- Scope: STRT and END set a data array to rotate.
- Shifting direction and time: rotates N times in the chosen direction set by STRT and END (STRT → END)
- Input data setting: fills an empty element after shifting with input data (IN).
- Output: the result is written at ANY\_ARY designated by **SRC**, and the overflowing data by shifting from END to STRT is written at OUT.





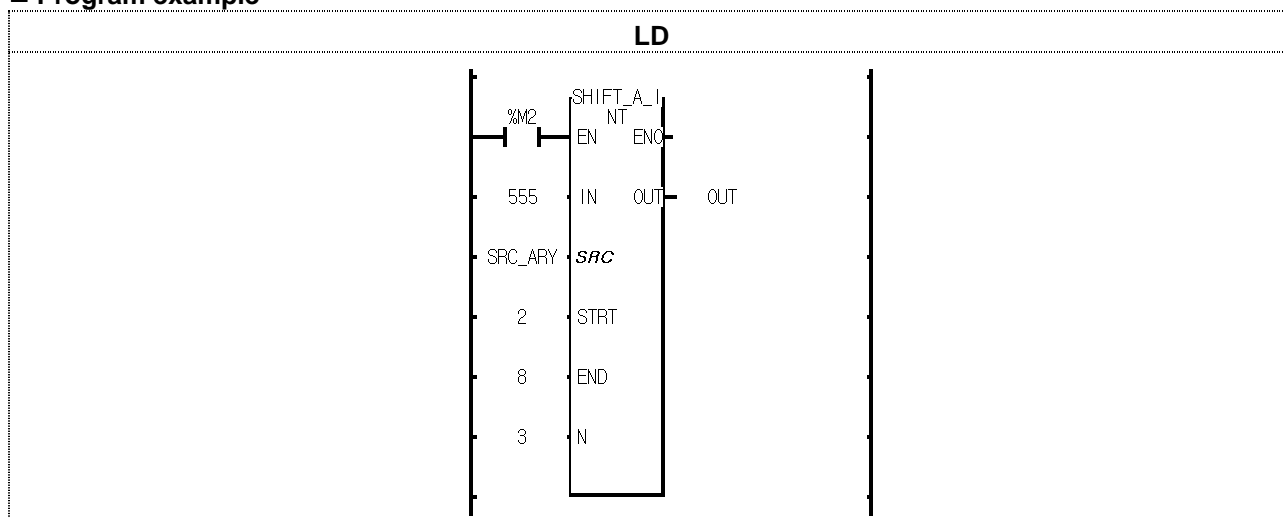
Function	In/Out Array Type	Description
SHIFT_A_BOOL	BOOL	It shifts designated elements of an array block in the chosen direction.
SHIFT_A_BYTE	BYTE	
SHIFT_A_WORD	WORD	
SHIFT_A_DWORD	DWORD	
SHIFT_A_LWORD	LWORD	
SHIFT_A_SINT	SINT	
SHIFT_A_INT	INT	
SHIFT_A_DINT	DINT	
SHIFT_A_LINT	LINT	
SHIFT_A_USINT	USINT	
SHIFT_A_UINT	UINT	
SHIFT_A_UDINT	UDINT	
SHIFT_A_ULINT	ULINT	
SHIFT_A_REAL	REAL	
SHIFT_A_LREAL	LREAL	
SHIFT_A_TIME	TIME	
SHIFT_A_DATE	DATE	
SHIFT_A_TOD	TOD	
SHIFT_A_DT	DT	

### ■ Error

If STRT or END exceed the range of SRC array element, \_ERR and \_LER flags are set.

If an error occurs, there's no change in SRC and output OUT is the initial value of each variable type (i.e. INT=0, TIME=T#0S).

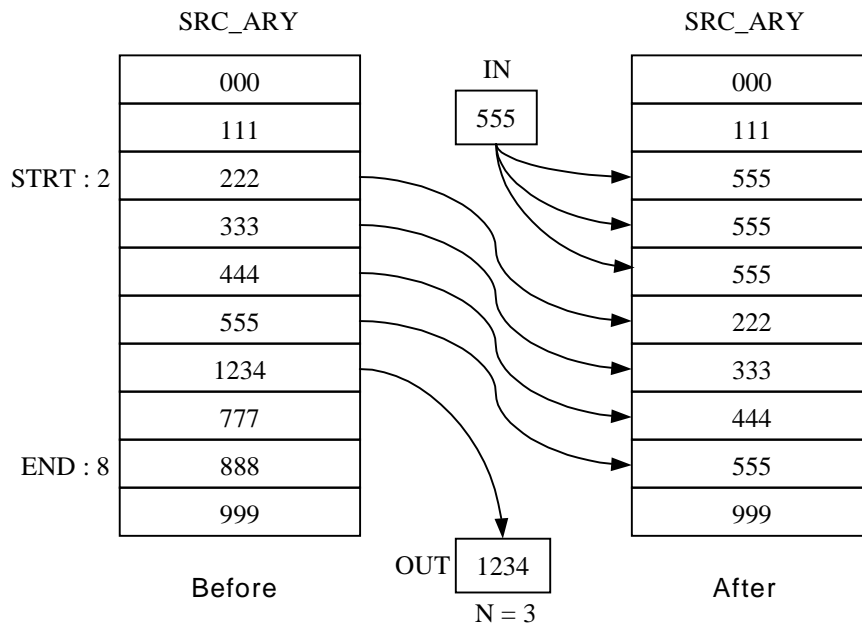
### ■ Program example



- (1) If the input condition (%M2) is on, SHIFT\_A\_INT function is executed.
- (2) It shifts designated elements (from 2nd to 8th elements) of SRC\_ARY.
- (3) It shifts three times the designated elements.
- (4) The empty elements after shifting, from array index 2 to array index 3, are filled with input '555'.
- (5) The overflowing data (1234), carry output, is written at OUT.

## 8. Basic Function/Function Block Library

---



**SHIFT\_C\_\*\*\***

Shift with Carry

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

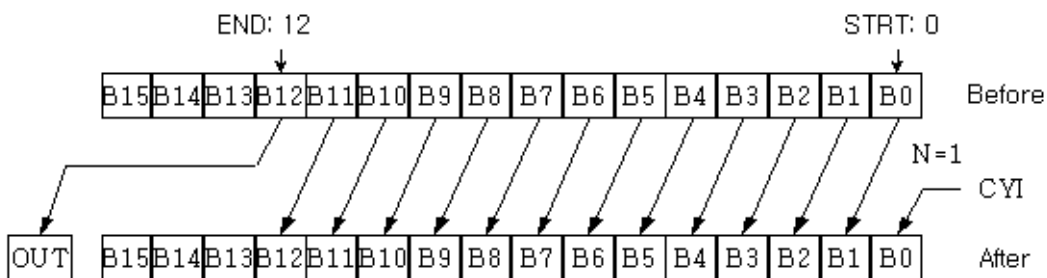
Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.  <b>CYI: Carry Input</b>                      STRT: starting bit position of SRC bit array to shift                      END: ending bit position of SRC bit array to shift                      N: bit number to shift</p> <p><b>Output</b>                      ENO: without an error, it will be 1                      OUT: carry output</p> <p><b>In/Out</b>  <b>SRC</b>: variable for shifting</p>

■ **Function**

It shifts a designated bit array of SRC bit arrays N times in the chosen direction.

Setting:

- Scope: STRT and END set a bit data to shift.
- Shifting direction and time: shifts N times from STRT to END.
- Input data setting: fills empty bit after shifting with input data (CYI).
- Output: the result is written at ANY\_BIT designated by **SRC**, and the overflowing bit data by shifting from END to STRT is written at OUT.



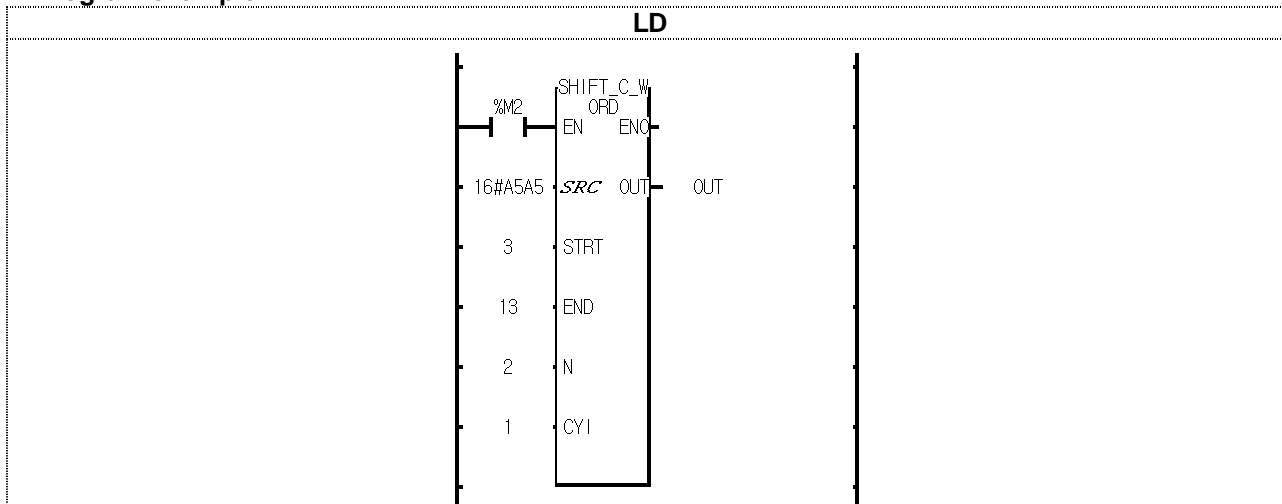
Function	SRC type	Description
SHIFT_C_BYTE	BYTE	It shifts a designated bit array of SRC bit arrays N times.
SHIFT_C_WORD	WORD	
SHIFT_C_DWORD	DWORD	
SHIFT_C_LWORD	LWORD	

## 8. Basic Function/Function Block Library

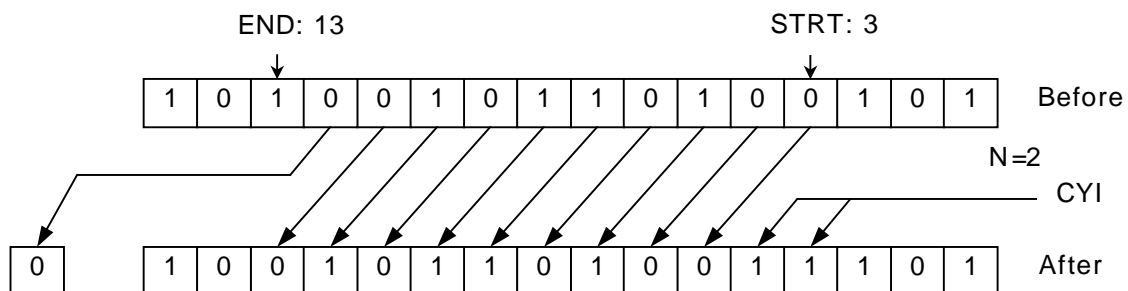
### ■ Error

If STRT or END exceed the bit number of SRC variable type, \_ERR and \_LER flags are set.  
There's no change in SRC data.

### ■ Program example



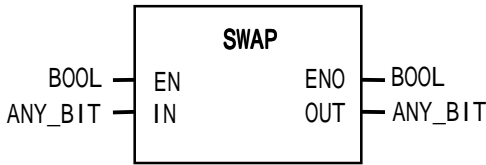
- (1) If the transition condition (%M2) is on, SHIFTC\_WORD function is executed.
- (2) 16#A5A5 is shifted from STRT to END by 2 bits and the empty bits after shifting are filled with 1 (CYI).
- (3) SRC after shifting is 16#969D and the overflowing bit data (0) is written at OUT after 2-bit shifting.



**SWAP\_\*\*\***

Swaps upper data for lower data

Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

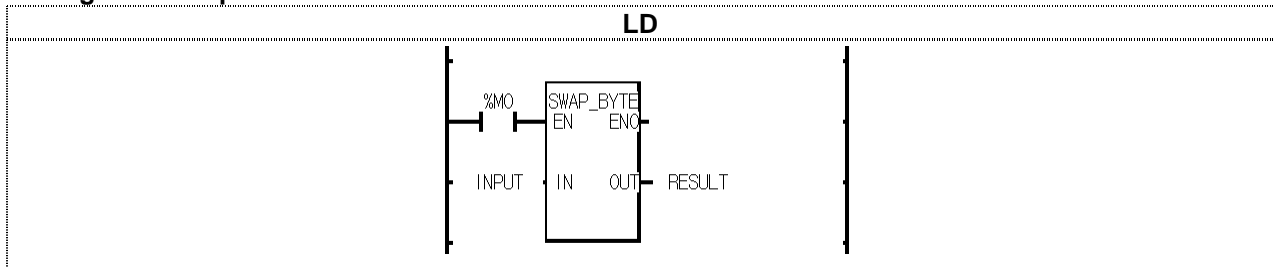
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1. IN: Input</p> <p><b>Output</b> ENO: without an error, it will be 1. OUT: swapped data</p>

■ **Function**

It swaps upper data for lower data.

Function	Input type	Description
SWAP_BYTE	BYTE	Swaps upper nibble for lower nibble data.
SWAP_WORD	WORD	Swaps upper byte for lower byte data.
SWAP_DWORD	DWORD	Swaps upper word for lower word data.
SWAP_LWORD	LWORD	Swaps upper double word for lower double word data.

■ **Program example**



- (1) If the transition condition (%M0) is on, SWAP\_BYTE function is executed.
- (2) If INPUT (BYTE) = 16#5F, RESULT (BYTE) = 16#F5.

## 8. Basic Function/Function Block Library

### UNI\_\*\*\*

Unites data

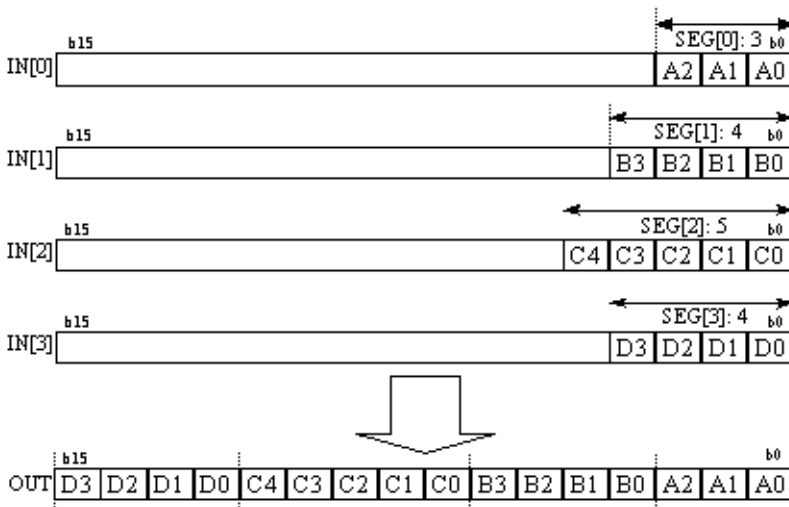
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: input data array            SEG: bit-number-designate array to unite data</p> <p><b>Output</b>            ENO: without an error, it will be 1            OUT: united data</p>

#### ■ Function

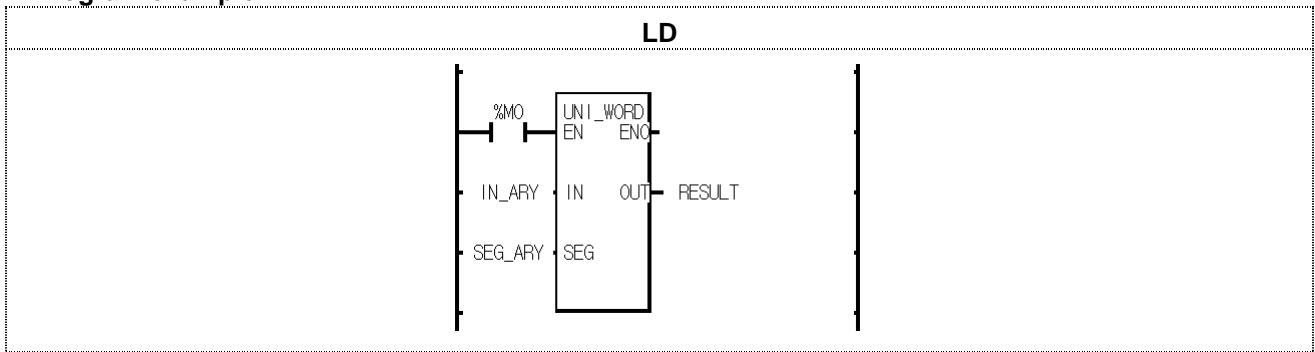
It unites an input data array from the lower bit to a designated bit set by SEG and produces an output.

Function	Input type	Output type	Description
UNI_BYTE	BYTE	BYTE	It cuts an input array into bit data set by SET and produces an output (united data) with the same array type of input.
UNI_WORD	WORD	WORD	
UNI_DWORD	DWORD	DWORD	
UNI_LWORD	LWORD	LWORD	



If the sum of value set by SEG exceeds the bit number of input data type, \_ERR and \_LER flags are set.  
 If the number of arrays of IN and SEG is different, output OUT is 0 and \_ERR and \_LER flags are set.

■ Program example



(1) If the transition condition (%M0) is on, UNI\_WORD function is executed.

(2) If input IN\_ARY and SEG\_ARY are as below,

IN_ARY[0]	A 3 B 5	SEG_ARY[0]	3
IN_ARY[1]	B 4 C 6	SEG_ARY[1]	4
IN_ARY[2]	C 5 D 7	SEG_ARY[2]	7
IN_ARY[3]	D 6 E 8	SEG_ARY[3]	2

output RESULT = 2#00 1010111 0110 101 = 16#2BB5.

## 8. Basic Function/Function Block Library

### WORD\_BYTE

Divides WORD into two bytes

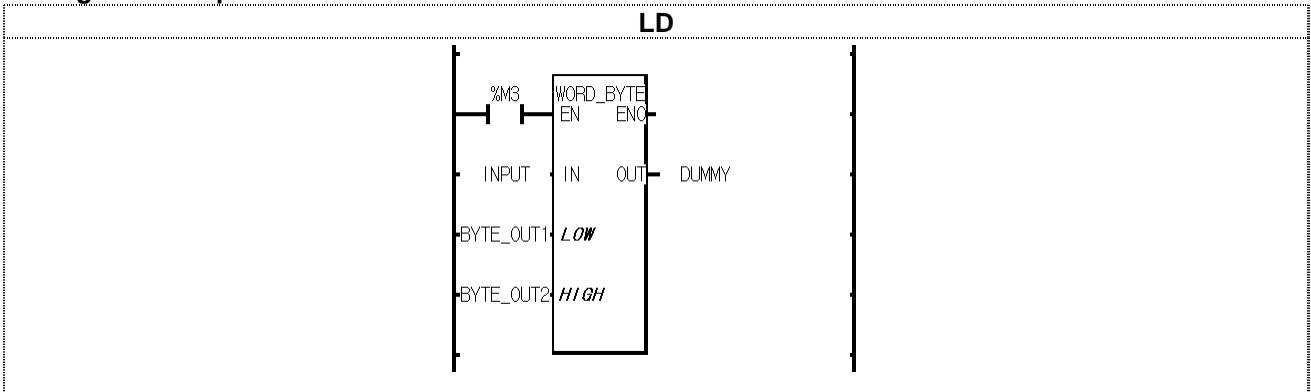
Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>            EN: executes the function in case of 1.            IN: WORD Input</p> <p><b>Output</b>            ENO: without an error, it will be 1.            OUT: dummy output</p> <p><b>In/Output</b>            LOW: lower BYTE output            HIGH: upper BYTE output</p>

#### ■ Function

It divides one word data into two byte data.  
 LOW: lower byte output, HIGH: upper byte output

#### ■ Program example



- (1) If the transition condition (%M3) is on, WORD\_BYTE function is executed.
- (2) If input variable INPUT is 16#ABCD, then BYTE\_OUT1 = 16#CD and BYTE\_OUT2 = 16#AB.



## WORD\_DWORD

Combines two WORD data into DWORD

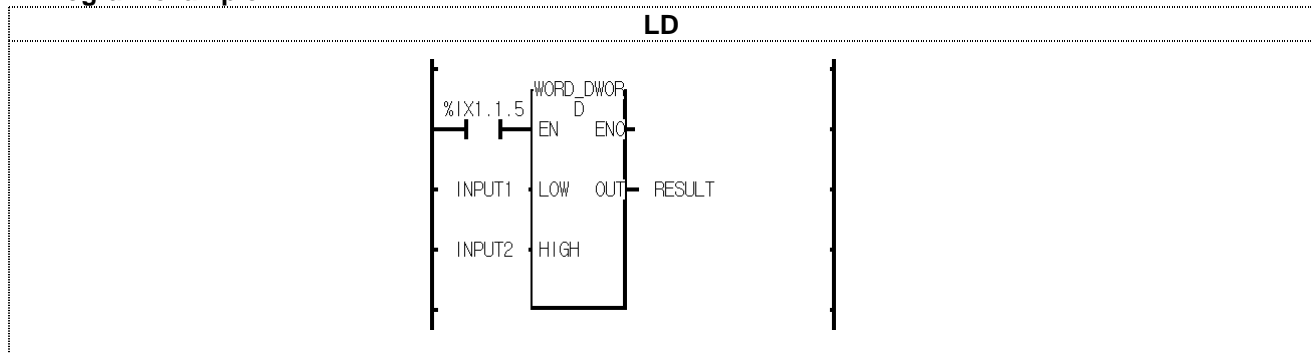
Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function	Description
	<p><b>Input</b>                      EN: executes the function in case of 1.                      LOW: lower WORD input                      HIGH: upper WORD input</p> <p><b>Output</b>                      ENO: without an error, it will be 1.                      OUT: DWORD output</p>

■ **Function**

It combines two WORD data into one DWORD.  
 LOW: lower WORD input, HIGH: upper WORD input

■ **Program example**



- (1) If the transition condition (%IX1.1.5) is on, WORD\_DWORD function is executed.
- (2) If input variable INPUT1 = 16#10203040 and INPUT2 = 16#A0B0C0D0, output variable RESULT = 16#A0B0C0D010203040.

## 8. Basic Function/Function Block Library

### XCHG\_ \*\*\*

Exchanges two input data

Mode l	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Appl ication							

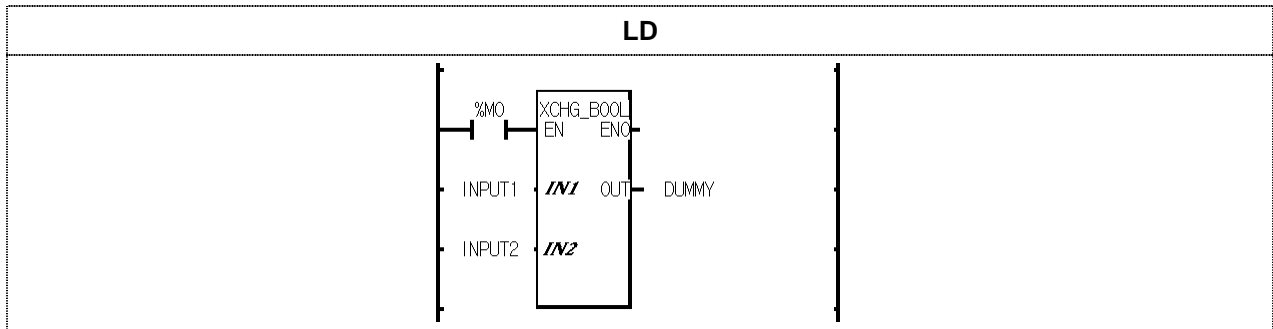
Function	Description
	<p><b>Input</b> EN: executes the function in case of 1.</p> <p><b>Output</b> ENO: Without an error, it will be 1. OUT: Dummy Output</p> <p><b>In/Out</b> IN1: In/Output 1 IN2: In/Output 2</p>

#### ■ Function

Exchanges input1 data with input2 data.

Function	In/Out type	Description
XCHG_BOOL	BOOL	Exchanges two BOOL input data.
XCHG_BYTE	BYTE	Exchanges two BYTE input data.
XCHG_WORD	WORD	Exchanges two WORD input data.
XCHG_DWORD	DWORD	Exchanges two DWORD input data.
XCHG_LWORD	LWORD	Exchanges two LWORD input data.
XCHG_SINT	SINT	Exchanges two SINT input data.
XCHG_INT	INT	Exchanges two INT input
XCHG_DINT	DINT	Exchanges two DINT input data.
XCHG_LINT	LINT	Exchanges two LINT input data.
XCHG_USINT	USINT	Exchanges two USINT input data.
XCHG_UINT	UINT	Exchanges two UINT input data.
XCHG_UDINT	UDINT	Exchanges two UDINT input data.
XCHG_ULINT	ULINT	Exchanges two ULINT input data.
XCHG_REAL	REAL	Exchanges two REAL input data.
XCHG_LREAL	LREAL	Exchanges two LREAL input data.
XCHG_TIME	TIME	Exchanges two TIME input data.
XCHG_DATE	DATE	Exchanges two DATE input data.
XCHG_TOD	TOD	Exchanges two TOD input data.
XCHG_DT	DT	Exchanges two DT input data.
XCHG_STRING	STRING	Exchanges two STRING input data.

■ Program example



- (1) If the transition condition (%M0) is on, XCHG\_BOOL function is executed.
- (2) If INPUT1 = 0 and INPUT2 = 1, it will exchange two input data. After the function execution, INPUT1 = 1 and INPUT2 = 0.

## ***8. Basic Function/Function Block Library***

---

### **8.3 Basic Function Block Library**

1. This chapter describes basic function blocks respectively.
2. It's much easier to apply function block library to your program after grasping the general of function blocks.

**CTD**

Down Counter (function block)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function block	Description
	<p><b>Input</b></p> <p>CD: down counter pulse input            LD: loads a preset value            PV: preset value</p> <p><b>Output</b></p> <p>Q: down counter output            CV: current value</p>

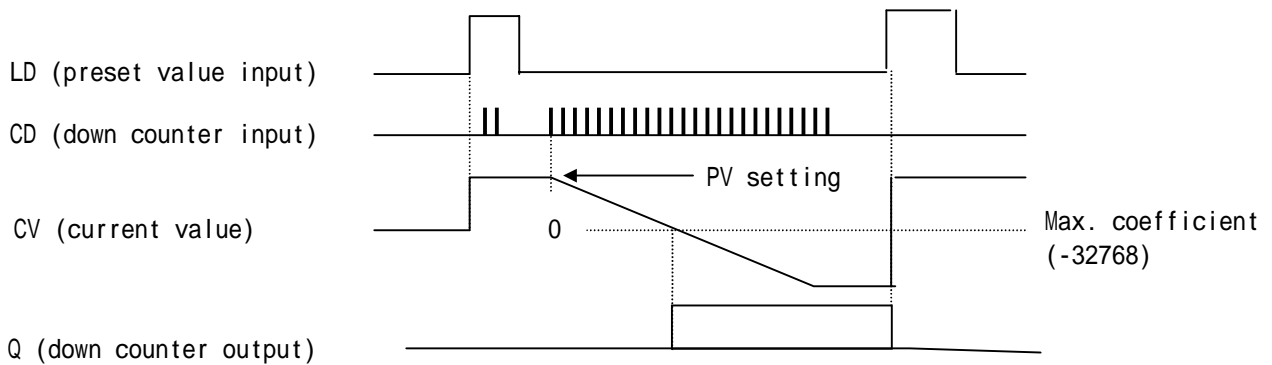
■ **Function**

Down counter function block CTD decreases the current value (CV) by 1 with every rising pulse input. CV decreases only when CV is more than the minimum value of INT (-32768); after reaching it, CV does not change its value.

When LD is 1, PV is loaded into CV (CV=PV).

Output Q is 1 when CV is 0 or a negative number.

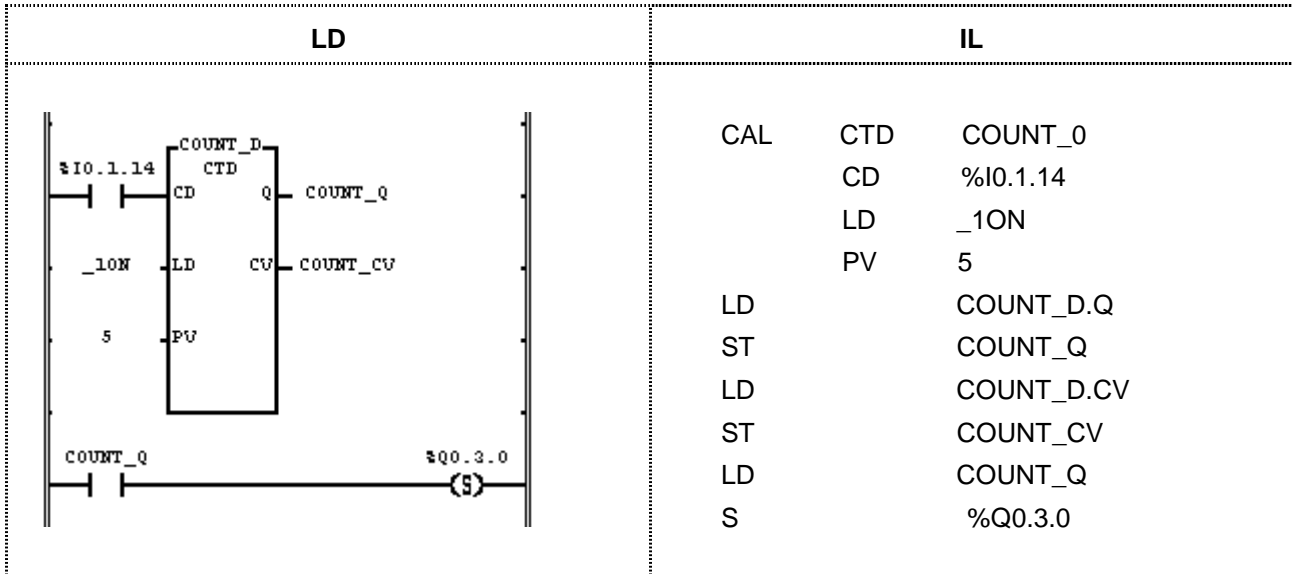
■ **Time Chart**



## 8. Basic Function/Function Block Library

### ■ Program Example

This is the program that sets the output contact (%Q0.3.0) when the down counter pulse input enters the input contact (%I0.1.14) five times.



- (1) Register the name of CTD function block (COUNT\_D).
- (2) Make the input contact (%I0.1.14) attached to CD.
- (3) Make the flag \_10N (1 scan ON contact) that loads PV into CV.
- (4) Set the PV value as 5.
- (5) Set the CV value as the random output variable (COUNT\_CV).
- (6) Set the Q value as the random output variable (COUNT\_Q).
- (7) Compile and write your program to the PLC after completing the program.
- (8) After writing, change the PLC mode (Stop -> Run).
- (9) If program runs, PV 5 will be loaded into CV (Count\_CV).
- (10) The current value CV (COUNT\_CV) decreases by 1 when the pulse input enters the input contact (%I0.1.14).
- (11) When the down counter pulse input enters the input contact (%I0.1.14) five times, CV (COUNT\_CV) will be 0 and Q (COUNT\_Q) 1
- (12) If Q (COUNT\_Q) is 1, the output contact (%Q0.3.0) will be set.

**CTU**

Up Counter (function block)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function Block	Description
	<p><b>Input</b> CU: up counter pulse input R: reset input PV: loads a preset value</p> <p><b>Output</b> Q: increase counter output CV: current value</p>

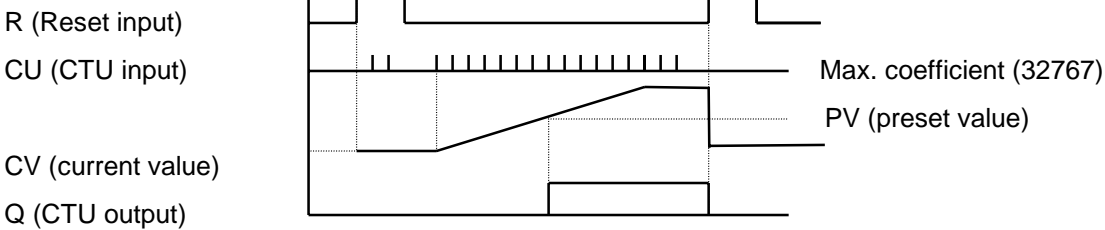
■ **Function**

Up counter function block CTU increases the current value (CV) by 1 with every rising pulse input. CV increases only when CV is less than the maximum value of INT (32767); after reaching it, CV does not change its value.

When the reset input (R) is 1, CV is cleared (0).

Output Q is 1 when CV is equal to or more than PV.

■ **Time Chart**



■ **Program Example**

This is the program that sets the output contact (%Q0.3.1) when the increase counter pulse input enters the input contact (%I0.1.15) ten times.

LD	IL
	<pre> CAL   CTU   COUNT_U       CU   %I0.1.15       R   %I0.1.5       PV   10  LD    COUNT_V.Q ST    COUNT_Q LD    COUNT_CV.Q ST    COUNT_CV LD    COUNT_Q S     %Q0.3.0         </pre>

## **8. Basic Function/Function Block Library**

---

- 
- (1) Register the name of CTU function block (COUNT\_U).
  - (2) Make the input contact %I0.1.15 attached to CU.
  - (3) Set the PV value as 10.
  - (4) Assign input contact %I0.1.5 to the reset input R.
  - (5) Set the CV value as the random output variable (COUNT\_CV).
  - (6) Set the Q value as the random output variable (COUNT\_Q).
  - (7) Compile and write your program to the PLC after completing the program.
  - (8) After writing, change the PLC mode (Stop - Run).
  - (9) The current value CV (COUNT\_CV) increases by 1 when the pulse input enters the input contact (%I0.1.15).
  - (10) When the up counter pulse input enters the input contact (%I0.1.15) ten times, CV (COUNT\_CV) will be 10 and Q (COUNT\_Q) 1
  - (12) If Q (COUNT\_Q) is 1, the output contact (%Q0.3.0) will be set.



## CTUD

Up/Down Counter (function block)

Mode I	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b></p> <ul style="list-style-type: none"> <li>CU: up counter pulse input</li> <li>CD: down counter pulse input</li> <li>R: reset</li> <li>LD: loads a preset value</li> <li>PV: preset value</li> </ul> <p><b>Output</b></p> <ul style="list-style-type: none"> <li>QU: up counter output</li> <li>QD: down counter output</li> <li>CV: current value</li> </ul>

### ■ Function

Up/Down counter function block CTUD increases the current value (CV) by 1 with every rising up-counter pulse input (CU) and decreases CV by 1 with every rising down-counter pulse input (CD). Note that CV is between -32768 and 32767 (INT).

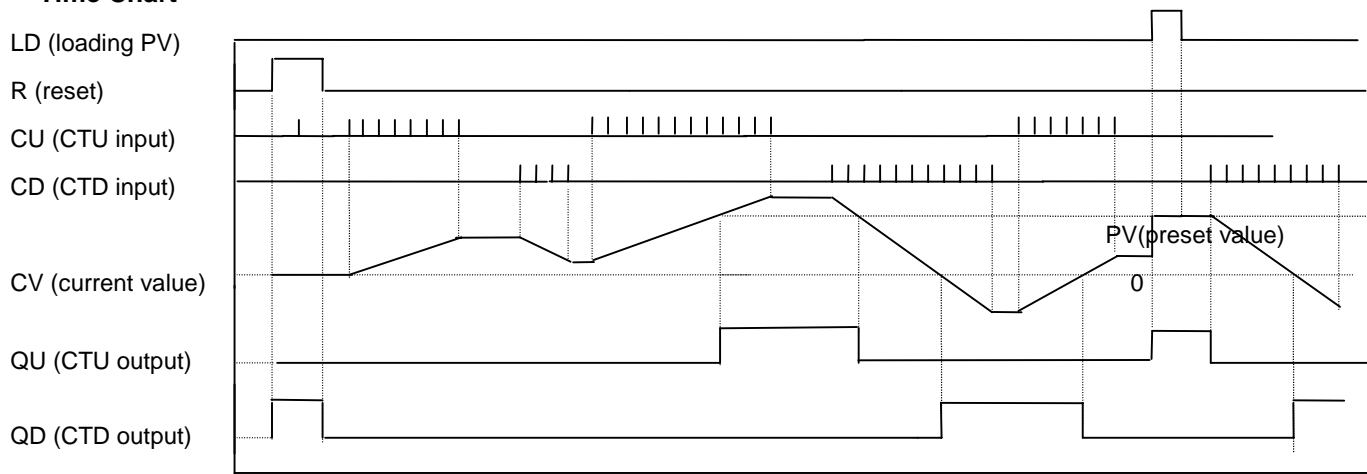
When LD is 1, PV is loaded into CV (CV=PV).

When the reset input R is 1, CV is cleared (0).

When CV reaches PV, the output QV is 1; when CV is 0 or a negative integer, the output QD is 1.

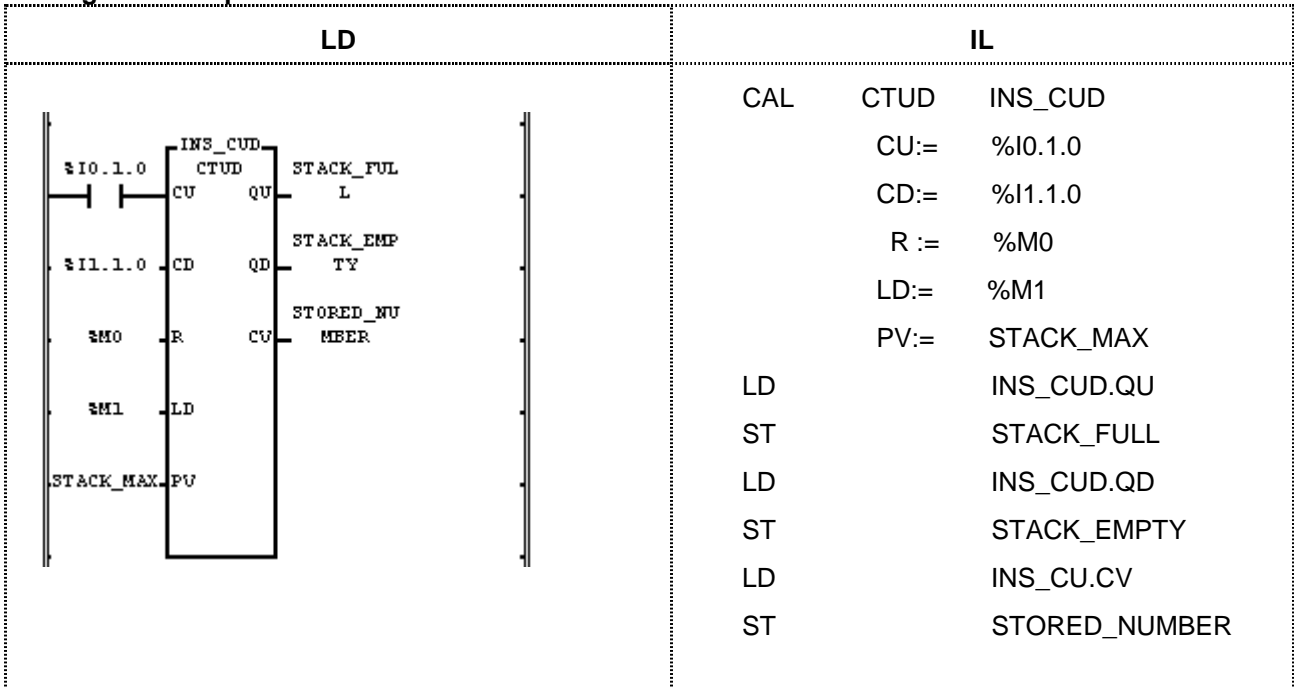
The operation for each input signal is executed in order of R > LD > CU > CD. Note that if the input signals are fed to the input (CU, CD, R, and LD) of CTUD at the same time, the operation of CTU follows the above priority.

### ■ Time Chart

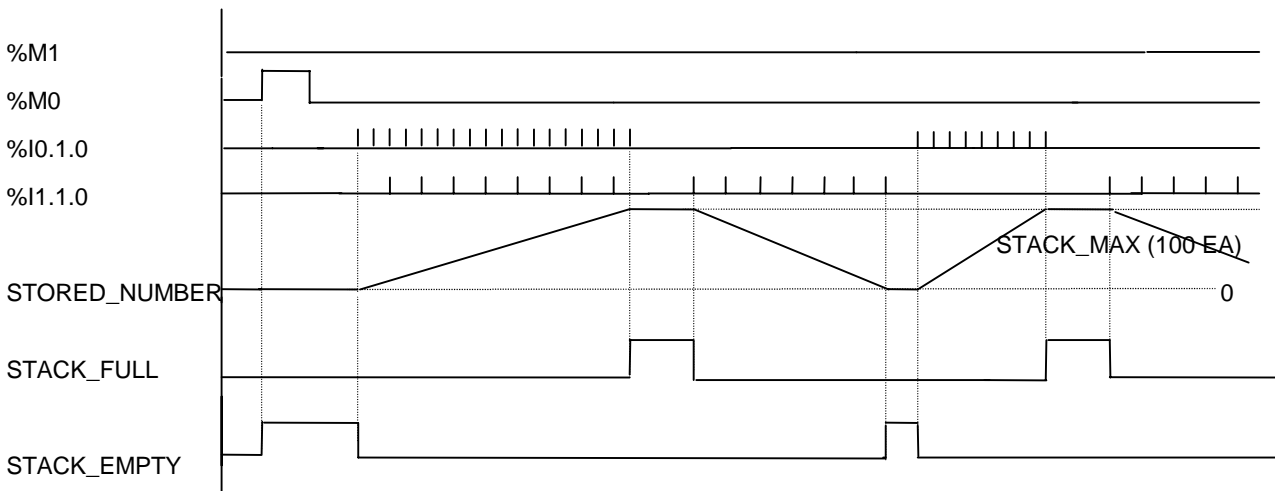


## 8. Basic Function/Function Block Library

### ■ Program Example



Conditions are: the temporary loading part STACK\_MAX is 100; IN is 1 with every material-input signal while OUT is 1 with every material-output signal. If the material input process is faster than the material-output one and every material is loaded so that the STACK\_MAX is equal to or more than 100, then QU is 1 (STACK\_FULL = 1); if there's no material left in the loading part, QD is 1 (STACK\_EMPTY = 1). At the STORED\_NUMBER, the number of remaining material in the loading part is shown.



### F\_TRIG

Falling Edge Detection (function block)

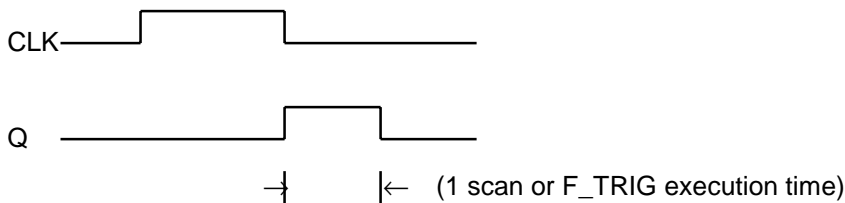
Model	GM1	GM2	GM3	GM4	GM6	GM7
Application						

Function Block	Description
	<p><b>Input</b> CLK: input signal</p> <p><b>Output</b> Q: falling edge detection result</p>

#### ■ Function

The output Q of function block F\_TRIG is 1 with the falling pulse input to CLK. And 1 scan later, without further falling pulse input, the output Q is 0 ever after.

#### ■ Time Chart



#### ■ Program Example

LD	IL
	<pre> CAL    F_TRIG    INS_FT         CLK:=    %I0.0.0 LD      INS_FT.Q ST      FALL_DETECT                     </pre>

If the input variable (%I0.0.0) changes from 1 to 0, while detecting its state, the output variable FALL\_DETECT will be 1. And 1 scan later, the output variable FALL\_DETECT will be 0.

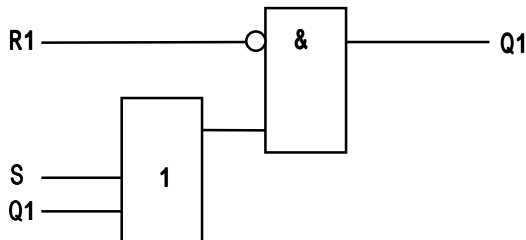
## 8. Basic Function/Function Block Library

### RS

Reset Priority Bistable (function block)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function Block	Description
	<p><b>Input</b> R1: Reset condition S: Set condition</p> <p><b>Output</b> Q1: Operation result</p>

#### ■ Function

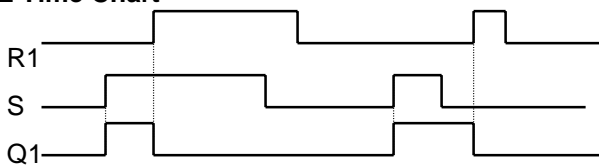


If R1 is 1, output Q1 will be 0 regardless of the state of S.

The output variable Q1 is 1 when it maintains the previous state, R1 is 0, and S is 1, it will be 1.

The initial state of Q1 is 0.

#### ■ Time Chart



#### ■ Program Example

LD	IL
	<pre> CAL   RS   INS_R       R1:=  RESET1       S:=   SET1  LD    INS_R.Q1 ST    RESULT </pre>

(1) The output variable RESULT is 0 and maintains its value when the input variables SET1 and RESET1 become simultaneously ON.

(2) The output variable RESULT is 0 and maintains its value when RESET1 becomes ON and SET1 is OFF.

(3) The output variable RESULT is 1 and maintains its value when SET1 becomes ON and RESET1 is OFF,

## R\_TRIG

Rising Edge Detection (function block)

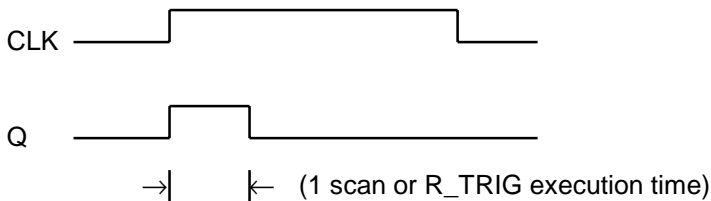
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b> CLK: input signal</p> <p><b>Output</b> Q: rising edge detection result</p>

### ■ Function

The output Q of function block R\_TRIG is 1 with the rising pulse input to CLK. And 1 scan later, without further falling pulse input, the output Q is 0 ever after.

### ■ Time Chart



### ■ Program Example

LD	IL
	<p>CAL R_TRIG INS_RT</p> <p>CLK: = IN_SIGNAL</p> <p>LD INS_RT.Q</p> <p>ST RISE_DETECT</p>

If the input variable IN\_SIGNAL changes from 0 to 1, while detecting its state, the output variable RISE\_DETECT will be 1. And 1 scan later, the output variable RISE\_DETECT will be 0.

## 8. Basic Function/Function Block Library

### SEMA

Semaphore (System resource allocation)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function Block	Description
	<p><b>Input</b> CLAIM: signal to claim a resource monopoly RELEASE: release signal</p> <p><b>Output</b> BUSY: waiting signal not to obtain the claimed resource</p>

#### ■ Function

This function block is used to get an exclusive control right for system resources.

BUSY is 1 when SEMA function is executed (CLAIM = 1 or 0, RELEASE = 0) and other program is using the resource. If you want to obtain the resource control right, wait until BUSY will be 0 after executing SEMA function block (CLAIM = 1, RELEASE = 0). When BUSY is 0, it controls the associate resource and after completing the control, it transfers the control right executing SEMA function block once again with CLAIM = 0 and RELEASE = 1. (At this time, the program that has the control right can execute SEMA function block with CLAIM = 0 and RELEASE = 1)

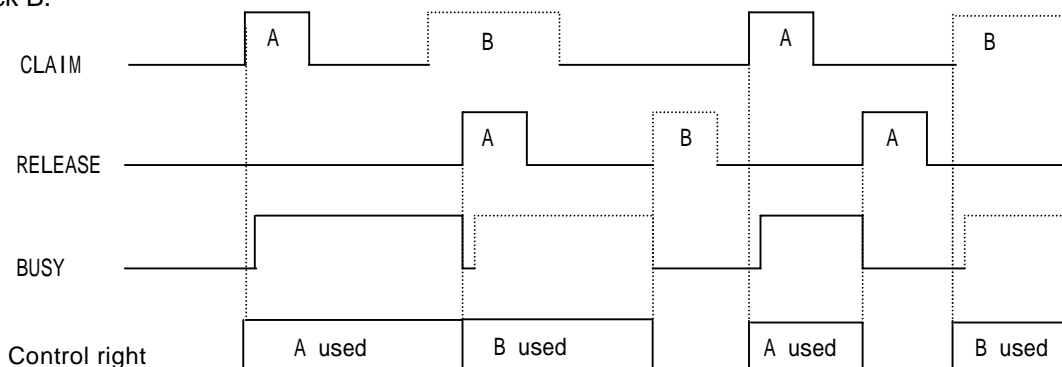
- The instance of SEMA should be declared as "GLOBAL" so that its access is available in the programs requiring the resource.
- Each program to claim the same resource should be designated as the same priority.
- Not available to use between multi-CPU modules in GM1.
- Internal execution structure of SEMA function block

```

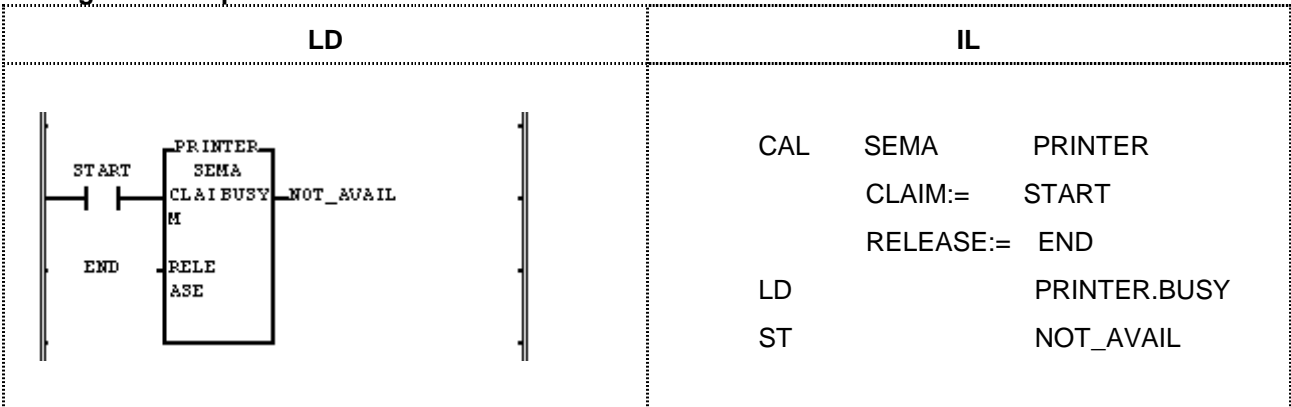
VAR X : BOOL := 0 ; END_VAR
    BUSY := X ;
    IF CLAIM THEN X := 1 ;
    ELSIF RELEASE THEN BUSY := 0 ; X := 0 ;
    END_IF
    
```

#### ■ Time Chart

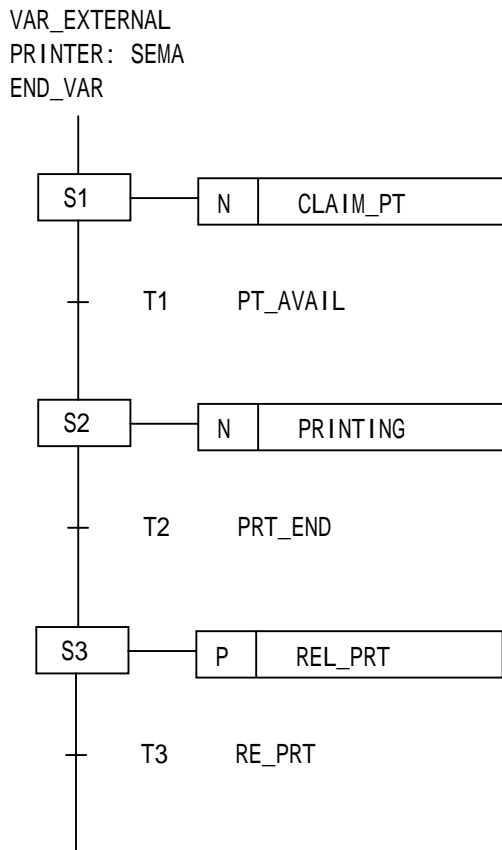
The access right to control the same resource is transferred between the program block A and the program block B.



■ Program Example

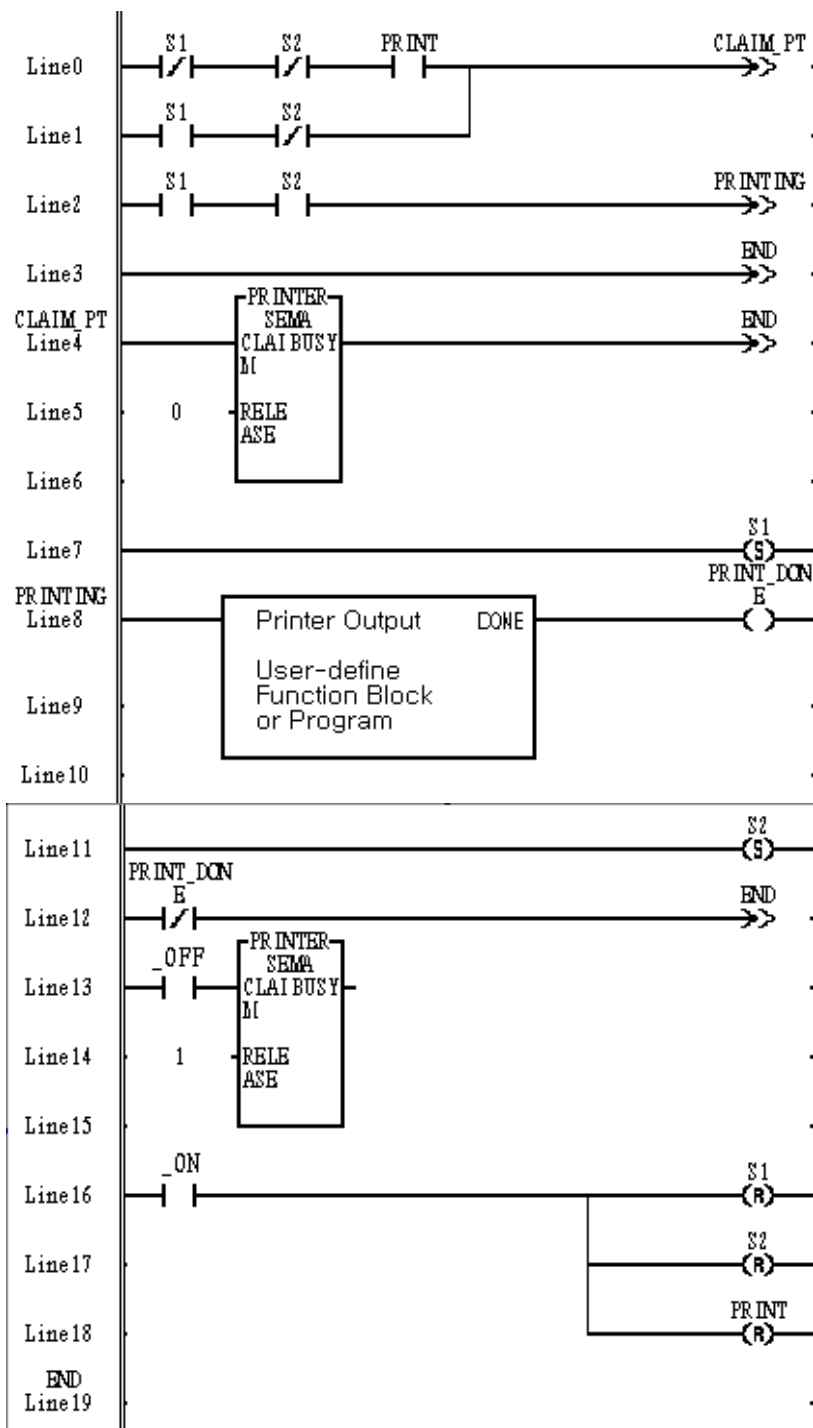


When you want to produce a printer output in different program blocks with the printer attached to the PLC system, you can easily control it by declaring the instance 'PRINTER' 'GLOBAL' and using SEMA function block named as 'PRINTER' in each program. If you execute SEMA function block (PRINTER), when START is 1 and END is 0, and claim the right to control the printer, while the printer is used in other program block, BUSY is 1. If the printer is not used in other program block, BUSY will be 0, which means you can start the program to produce the printer output with it. After completing the print control, execute SEMA with START = 0 and END = 1 so that other program can get the right to control it.



S1	CLAIM_PT; claim the printer control right
	<pre> CAL   SEMA   PRINTER       CLAIM:= 1       RELEASE:= 0                     </pre>
T1	PT_AVAIL; printer control right check
	<pre> LDN   PRINTER.BUSY ST    TRANS                     </pre>
S2	PRINTING; printer output
	Printer control program If print is completed, PRINT_DONE:=1
T2	PRT_END; print completion check
	<pre> LD    PRINTER_DONE ST    TRANS                     </pre>
S3	REL_PRT; transfer printer control
	<pre> CAL   SEMA   PRINTER       CLAIM:= 0       RELEASE:= 1                     </pre>
T3	RE_PRT; printer request again
	<pre> LD    PRT_REQ ST    TRANS                     </pre>

## 8. Basic Function/Function Block Library





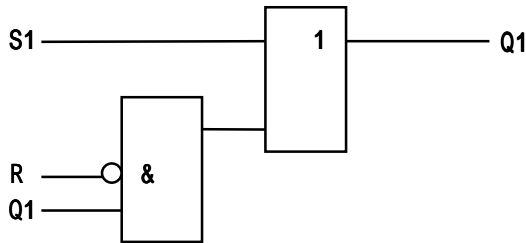
**SR**

Set Priority Bistable (function block)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b> S1: set condition R: reset condition</p> <p><b>Output</b> Q1: operation result</p>

■ **Function**

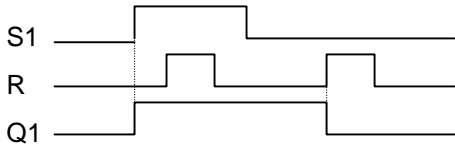


If S1 is 1, output Q1 will be 1 regardless of the state of R.

The output variable Q1 is 0 and it maintains the previous state when S1 is 0, and R is 1.

The initial state of Q1 is 0.

■ **Time Chart**



■ **Program Example**

LD	IL
	<pre> CAL   SR   INS_S       S1:=  SET1       R:=   RESET1 LD     INS_S.Q1 ST     RESULT                     </pre>

(1) If input variable SET1 becomes 1, output variable RESULT will be ON.

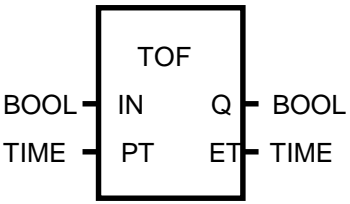
(2) The output variable RESULT becomes 0 when input variable SET1 becomes 0 and RESET1 ON.

## 8. Basic Function/Function Block Library

### TOF

OFF Delay Timer (function block)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

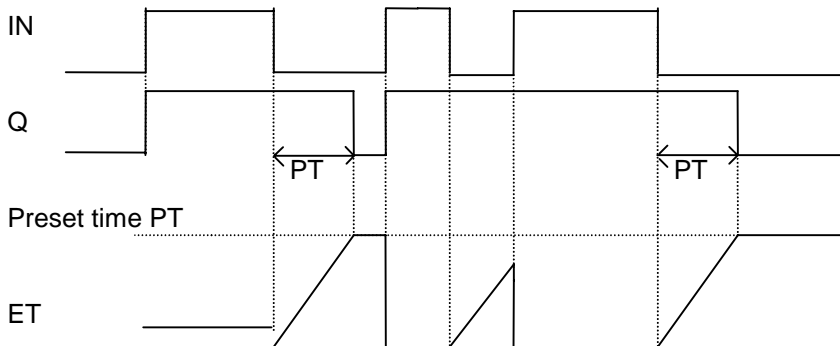
Function Block	Description
	<p><b>Input</b> IN: timer operation condition PT: preset time</p> <p><b>Output</b> Q: timer output ET: elapsed time</p>

#### ■ Function

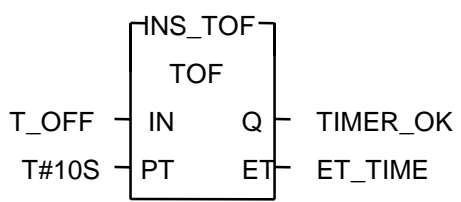
If IN is 1, Q will be 1. And after IN becomes 0 and the preset time (PT) of TOF passes, Q becomes 0.

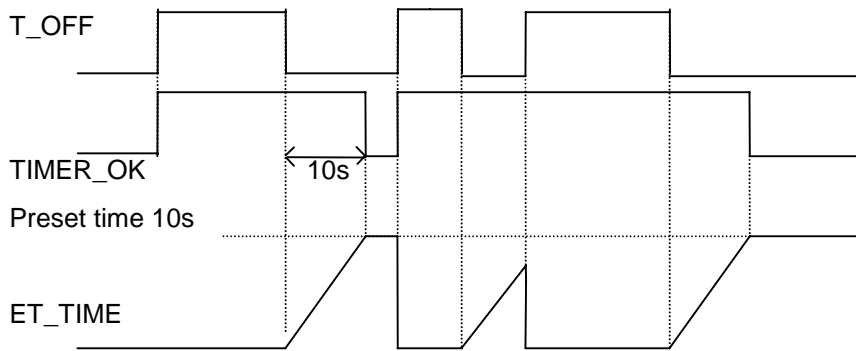
After IN becomes 0, the elapsed time (ET) will be shown. If IN becomes 1 before ET reaches the preset time, ET will be 0 again.

#### ■ Time Chart



#### ■ Program Example

LD	IL
	<pre> CAL  TOF  INS_TOF       IN:=  T_OFF       PT:=  T#10S LD    INS_TOF.Q ST    TIMER_OK LD    INS_TOF.ET ST    ET_TIME </pre>



- (1) Output variable TIMER\_OK is 1 when input variable T\_OFF becomes 1.
- (2) TIMER\_OK is 0 only if 10 seconds passes after T\_OFF becomes 0.
- (3) If T\_OFF becomes 1 again in 10 seconds after it turned OFF, TOF will be initialized (TIMER\_OK is 1).
- (4) After T\_OFF becomes 0, the elapsed time (ET\_TIME) will be measured and shown.

## 8. Basic Function/Function Block Library

### TON

ON Delay Timer (function block)

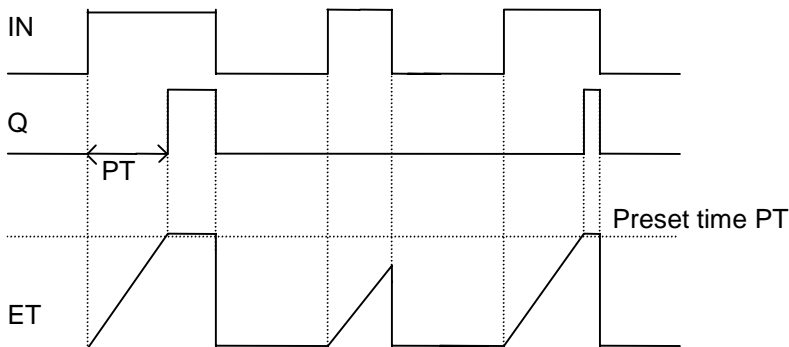
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b> IN: timer operation condition PT: preset time</p> <p><b>Output</b> Q: timer output ET: elapsed Time</p>

#### ■ Function

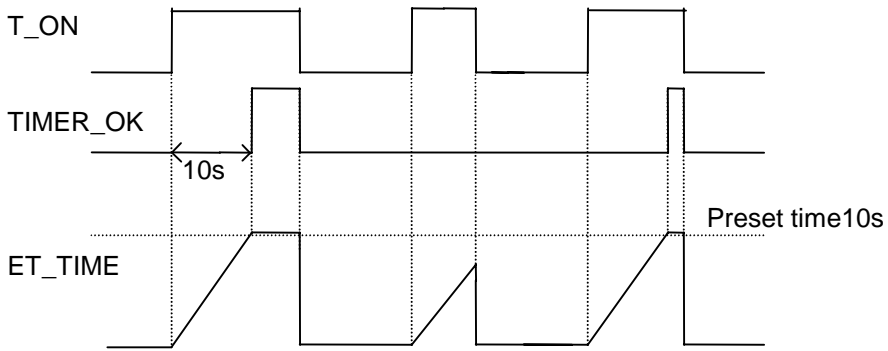
Elapsed time (ET) is measured and shown after IN becomes 1. When IN becomes 0 before ET reaches the preset time, ET will be 0. If IN becomes 0 after Q is 1, Q will be 0.

#### ■ Time Chart



#### ■ Program Example

LD	IL
	<pre> CAL  TON  INS_TON       IN: = T_ON       PT: = T#10S LD    INS_TON.Q ST    TIMER_OK LD    INS_TON.ET ST    ET_TIME           </pre>

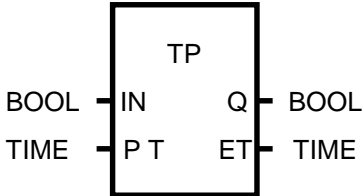


- (1) The output `TIMER_OK` = 1 ten seconds later after the input `T_ON` is asserted (`T_ON` = 1).
- (2) Elapsed time `ET_TIME` is measured and shown after the input `T_ON` becomes 1.
- (3) When `T_ON` = 0 before `ET_TIME` reaches the preset time (10s), `ET_TIME` will be 0.
- (4) If `T_ON` = 0 after `TIMER_OK` = 1, then `TIMER_OK` = 0 and `ET_TIME` = 0.

## 8. Basic Function/Function Block Library

### TP

Pulse timer (function block)	Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
	Application							

Function Block	Description
	<p><b>Input</b> IN: timer operation condition PT: preset time</p> <p><b>Output</b> Q: timer output ET: elapsed Time</p>

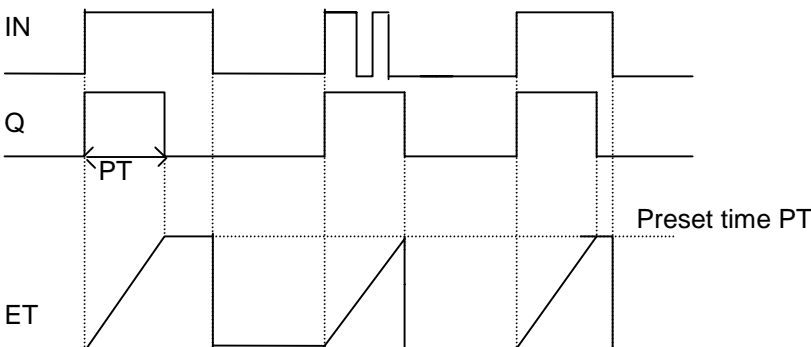
#### ■ Function

If IN = 1, Q will be 1 only during the preset time PT; if ET reaches PT, Q will be 0.

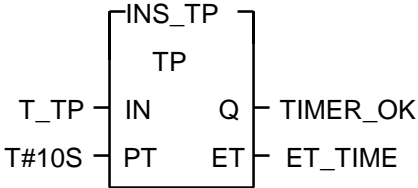
If IN = 1, elapsed time ET starts to be measured and maintains its value after when it reaches PT; if IN = 0 after ET reaches PT, ET = 0.

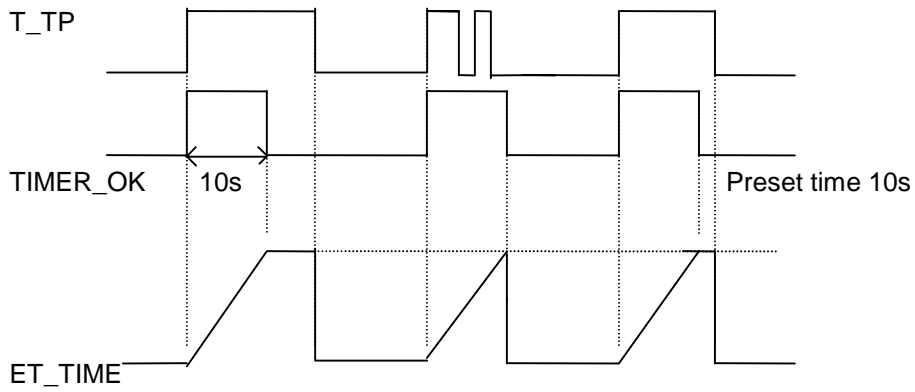
The state of IN doesn't matter while ET is measured (increased).

#### ■ Time Chart



#### ■ Program Example

LD	IL
	<pre> CAL  TP  INS_TP       IN: =  T_TP       PT: =  T#10S LD    INS_TP.Q ST    TIMER_OK LD    INS_TP.ET ST    ET_TIME </pre>



- (1) TIMER\_OK is 1 during 10 seconds after input T\_TP was asserted (T\_TP = 1). While ET\_TIME increases during 10 seconds, the state of input T\_TP doesn't affect TIMER\_OK.
- (2) ET\_TIME increases when it reaches T#10S and then it becomes 0 when T\_TP = 0.

## ***8. Basic Function/Function Block Library***

---

### **8.4 Application Function Block Library**

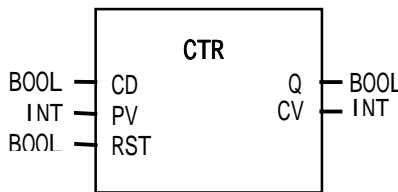
1. This chapter describes each application function block library (MASTER-K and others).
2. It's much easier to apply function block library to your program after grasping the general of function blocks.



## CTR

Ring Counter

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular block labeled 'CTR'. On the left side, there are three input terminals: 'CD' (labeled 'BOOL'), 'PV' (labeled 'INT'), and 'RST' (labeled 'BOOL'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'CV' (labeled 'INT').</p>	<p><b>Input</b>            CD: pulse input of Ring Counter            PV: preset value            RST: reset</p> <p><b>Output</b>            Q: Ring Counter output            CV: current value</p>

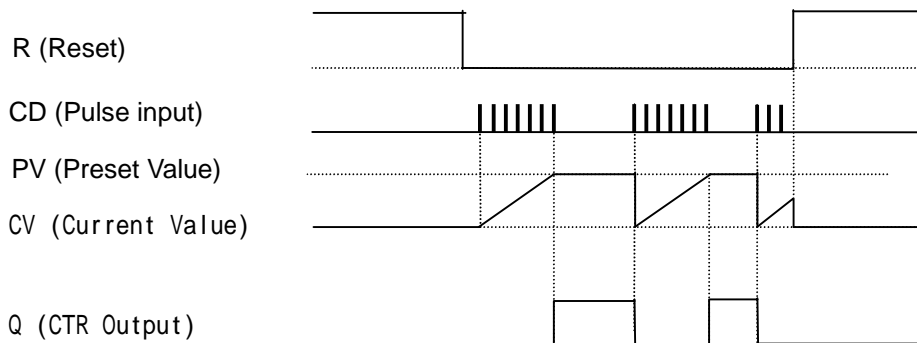
### ■ Function

CTR function block (Ring Counter) functions: current value (CV) increases with the rising pulse input (CD) and if, after CV reaches PV, CD becomes 1, then CV is 1.

When CV reaches PV, output Q is 1.

If CV is less than PV or reset input (RST) is 1, output Q is 0.

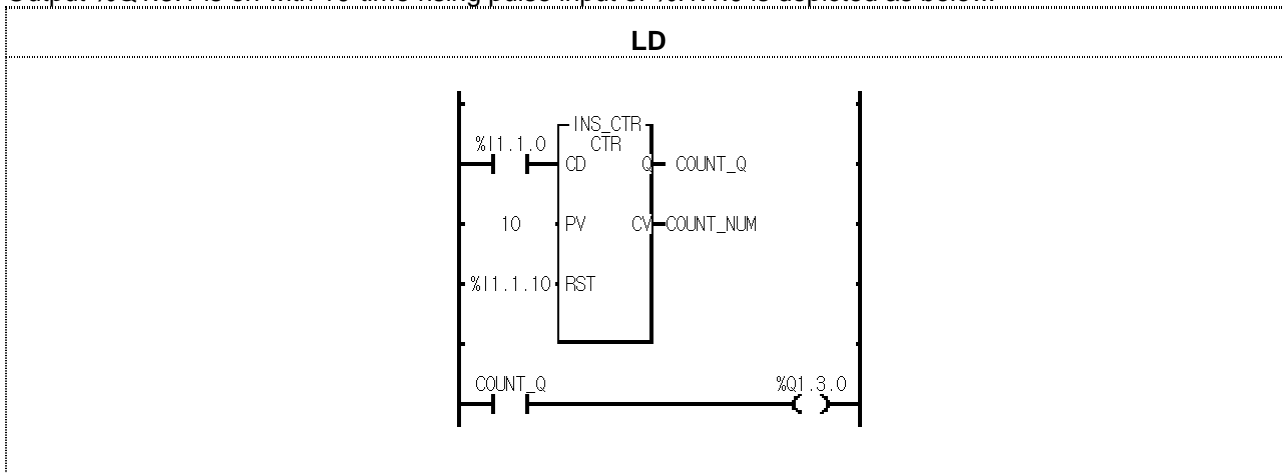
### ■ Timing Chart



## 8. Basic Function/Function Block Library

### ■ Program Example

Output %Q1.3.1 is on with 10-time rising pulse input of %I1.1.0 is depicted as below.

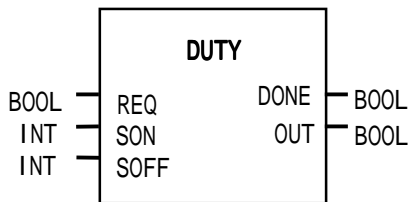


- (1) Define CTR function block as INS\_CTR.
- (2) Set %I1.1.0 to the input contact of CD referring to the above.
- (3) Set 10 to PV.
- (4) Set %I1.1.10 to RST resetting CV.
- (5) Set random variable COUNT\_NUM to CV.
- (6) Set random output variable COUNT\_Q to Q.
- (7) After a program is complete, compile and write it to PLC.
- (8) When 'Write' is complete, do 'Mode Change' (Stop → Run).
- (9) CV (COUNT\_NUM) increases by 1 in number with the rising input pulse of %I1.1.0, CD
- (10) With 10-time rising input pulse of input contact, CV is 10 which is the same as PV and output variable COUNT\_Q is 1.
- (11) If Q (COUNT\_Q) is 1, output contact %Q1.3.0 is on.
- (12) If the rising input pulse is loaded into input contact %I1.1.0, then Q (COUNT\_Q) is 0 and output contact %Q1.3.0 is off.

### DUTY

Scan setting On/Off

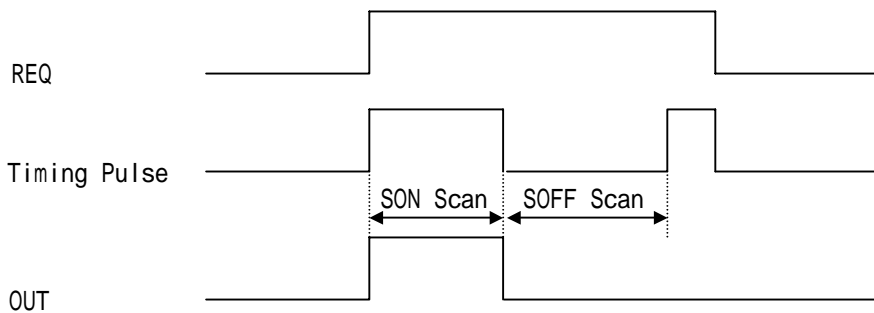
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b>                      REQ: requires to execute the function block                      SON: scan number to turn on                      SOFF: scan number to turn off</p> <p><b>Output</b>                      DONE: it is 1 when REQ is on and both input variables are not less than 0.                      OUT: output is 1 during on scan time</p>

■ **Function**

DUTY function block produces a pulse which is on during the SON scan time and off during the SOFF scan time while REQ is on.  
 If SON = 0, OUT is always off.  
 If SON > 0 and SOFF = 0, OUT is always on.  
 If REQ is off, OUT is off.  
 If SON < 0 or SOFF < 0, then DONE is off and OUT is 0.

■ **Timing Chart**

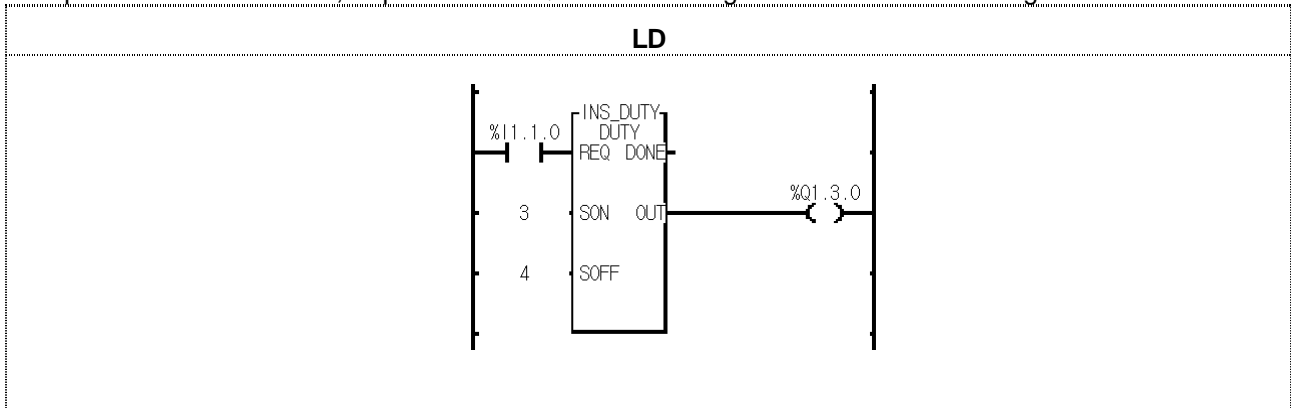


## 8. Basic Function/Function Block Library

---

### ■ Program Example

If input contact %I1.1.0 is set, output contact %Q1.3.0 is on during 3 scan times and off during 4 scan times.

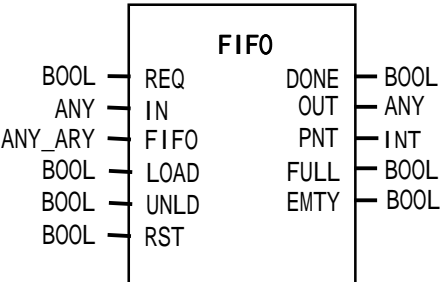


- (1) Define DUTY function block as DUTY\_C.
- (2) Set %I1.1.0 to REQ (the input contact) of DUTY.
- (3) Set 3 to SON.
- (4) Set 4 to SOFF.
- (5) Set %Q1.3.0 to output OUT.
- (6) After a program is complete, compile and write it to PLC.
- (7) When 'Write' is complete, do 'Mode Change' (Stop → Run).
- (8) If input contact %I1.1.0 is on, output contact %Q1.3.0 is on during 3 scan times and off during 4 scan times.

**FIFO** \*\*\*

Load/Unload data to FIFO stack  
(First In First Out)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

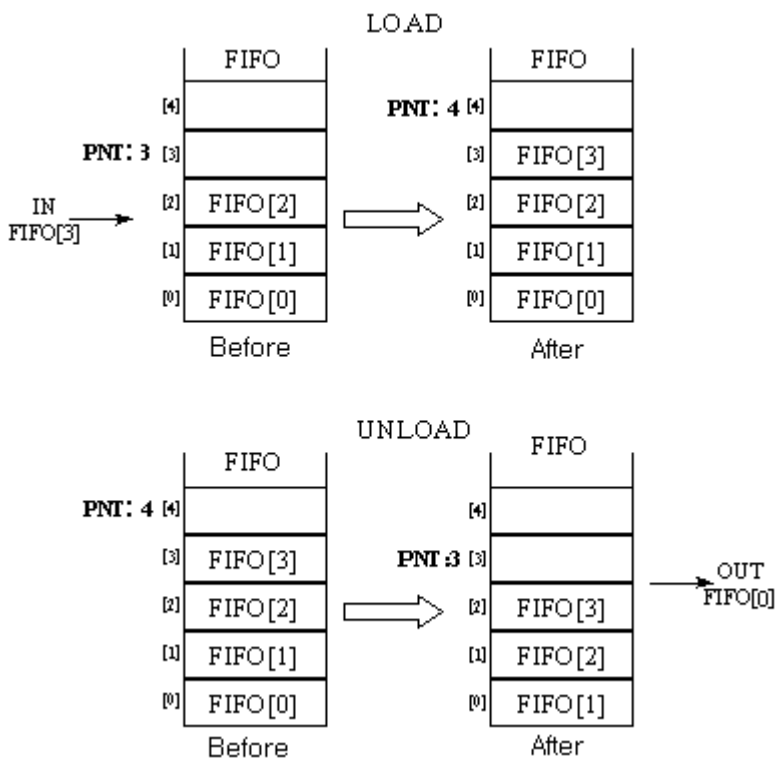
Function Block	Description
	<p><b>Input</b>                      REQ: requires to execute the function block                      IN: input data to be stored at FIFO stack                      LOAD: FB is on the input mode, if it's on.                      UNLD: FB is on the output mode, if it's on,                      RST: pointer value reset</p> <p><b>Output</b>                      DONE: it's 1 after first execution                      OUT: on output mode, it's the data from FIFO stack                      PNT: pointer for input data of FIFO stack                      FULL: if FIFO stack is full, it's 1                      EMTY: if FIFO stack is empty, it's 1</p> <p><b>In/Output</b>                      FIFO: array used as FIFO stack</p>

■ **Function**

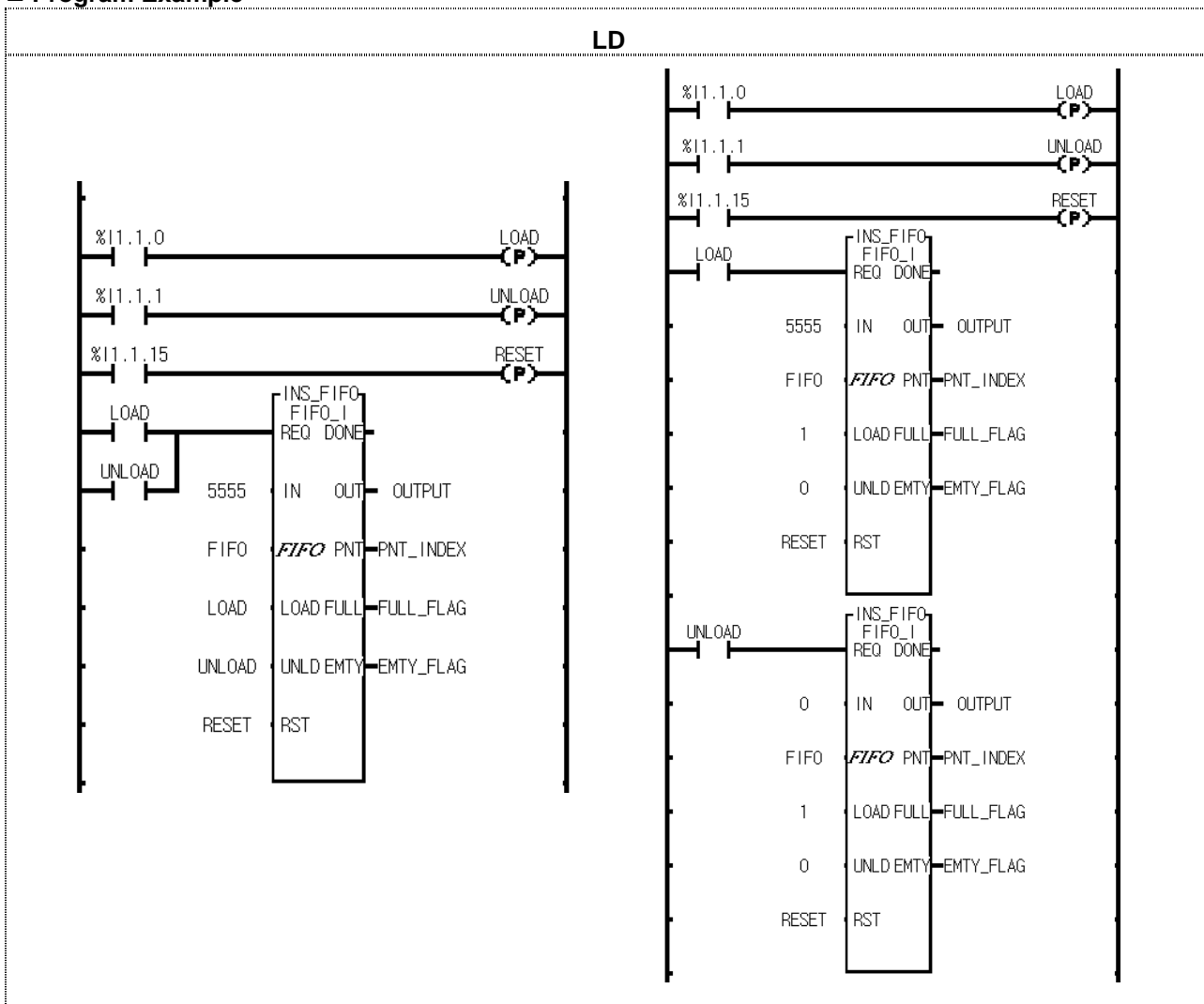
It loads IN to FIFO or unloads data from FIFO.  
 If Input and Output mode are set at the same time, it executes In/Output simultaneous.  
 If data is unloaded from FIFO, then the output is the lowest element of stack, the rest elements are shifts, PNT value is decreased by 1, and the element position of PNT is cleared (0).  
 If RST is loaded to FIFO, PNT is initialized as 0, EMTY is on and all the data of FIFO stack are cleared as 0.  
 The stack number is the input array number set by In/Output variable FIFO.  
 If you want to keep the data of FIFO array variables and FIFO function block instance in case that power is off or power failure occurs, set them as 'RETAIN'.  
 Reset functions without REQ input.  
 PNT shows the position of IN to be loaded next time, or the number of pointers to be loaded.  
 If it's on the input mode, output OUT is 0.

## 8. Basic Function/Function Block Library

Function	FIFO variable type	Description
FIFO_Q	BOOL	It functions as FIFO for BOOL-type data
FIFO_B	BYTE	It functions as FIFO for BYTE-type data
FIFO_W	WORD	It functions as FIFO for WORD-type data
FIFO_DW	DWORD	It functions as FIFO for DWORD-type data
FIFO_LW	LWORD	It functions as FIFO for LWORD-type data
FIFO_SI	SINT	It functions as FIFO for SINT-type data
FIFO_I	INT	It functions as FIFO for INT-type data
FIFO_DI	DINT	It functions as FIFO for DINT-type data
FIFO_LI	LINT	It functions as FIFO for LINT-type data
FIFO_USI	USINT	It functions as FIFO for USINT-type data
FIFO_UI	UINT	It functions as FIFO for UINT-type data
FIFO_UDI	UDINT	It functions as FIFO for UDINT-type data
FIFO_ULI	ULINT	It functions as FIFO for ULINT-type data
FIFO_R	REAL	It functions as FIFO for REAL-type data
FIFO_LR	LREAL	It functions as FIFO for LREAL-type data
FIFO_TM	TIME	It functions as FIFO for TIME-type data
FIFO_DAT	DATE	It functions as FIFO for DATE-type data
FIFO_TOD	TOD	It functions as FIFO for TOD-type data
FIFO_DT	DT	It functions as FIFO for DT-type data



■ Program Example



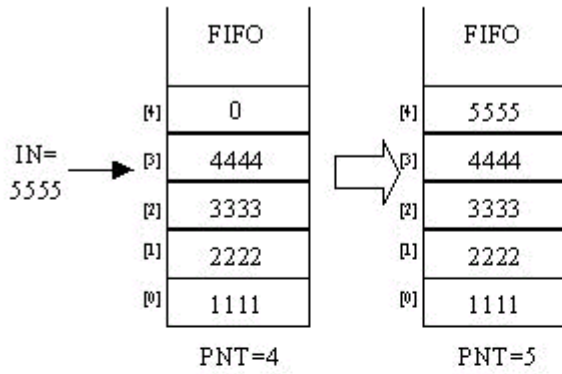
FIFO\_\*\*\* function block is used as the above. The two examples of the above execute the same operation. The left one is a program which executes input and output functions at the same time to use only one function block while the right one is a program which executes input and output functions independently to use input function and output function respectively. Note that the instance name should be the same on the right program.

- (1) If the input conditions (%I1.1.0, %I1.1.1, %I1.1.15) are on, FIFO\_INT is executed.
- (2) If input contact %I1.1.0 is on, load function is executed. 5555 is loaded to FIFO stack and PNT\_INDEX increased by 1.
- (3) If input contact %I1.1.1 is on, unload function is executed. 1111 is unloaded from FIFO stack and PNT\_INDEX decreased by 1.
- (4) If input contact %I1.1.15 is on, reset function is executed. All the stack of FIFO is cleared as 0, PNT\_INDEX is initialized as 0 and EMTY\_FLAG is on.

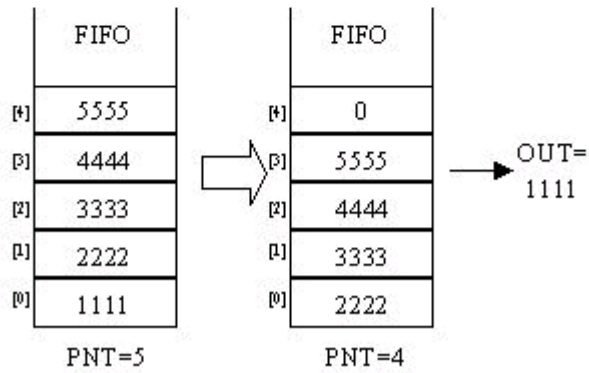
## 8. Basic Function/Function Block Library

---

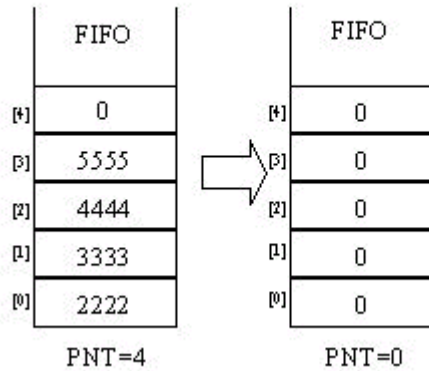
### LOAD (%I1.1.0 is ON)



### UNLOAD (%I1.1.1 is ON)



### RESET (%I1.1.15 is ON)

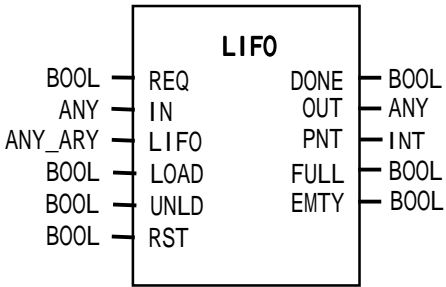




**LIFO\_\*\*\***

Load/Unload data to LIFO stack  
(Last In First Out)

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

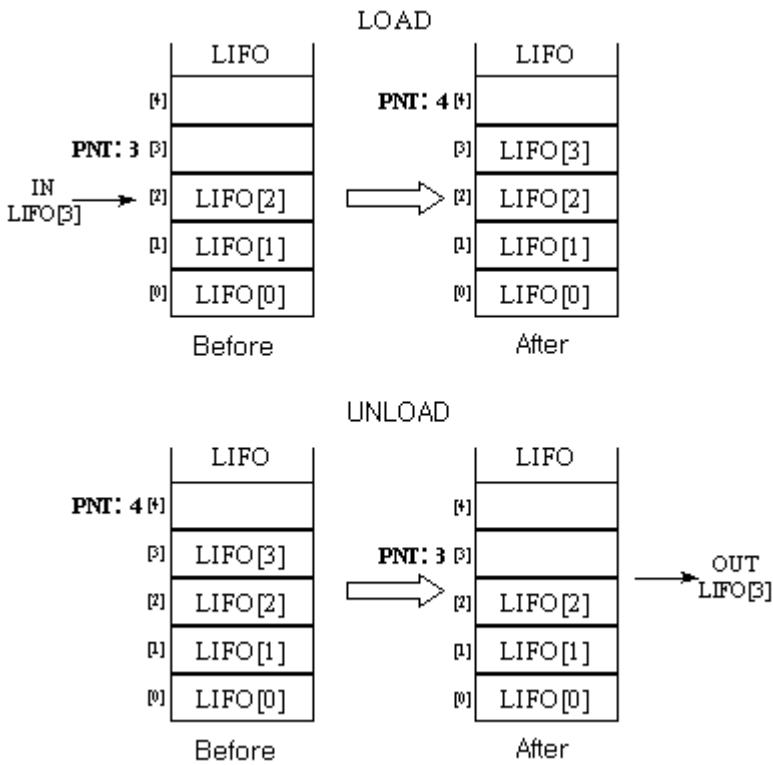
Function Block	Description
	<p><b>Input</b>                      REQ: requires to execute the function block                      IN: input data to be stored at LIFO stack                      LOAD: FB is on the input mode, if it's on                      UNLD: FB is on the output mode, if it's on                      RST: pointer value reset</p> <p><b>Output</b>                      DONE: it's 1 after first execution                      OUT: on output mode, it's the data from LIFO stack                      PNT: pointer for input data of LIFO stack                      FULL: if LIFO stack is full, it's 1                      EMTY: if LIFO stack is empty, it's 1</p> <p><b>In/Output</b>                      LIFO: array used as LIFO stack</p>

■ **Function**

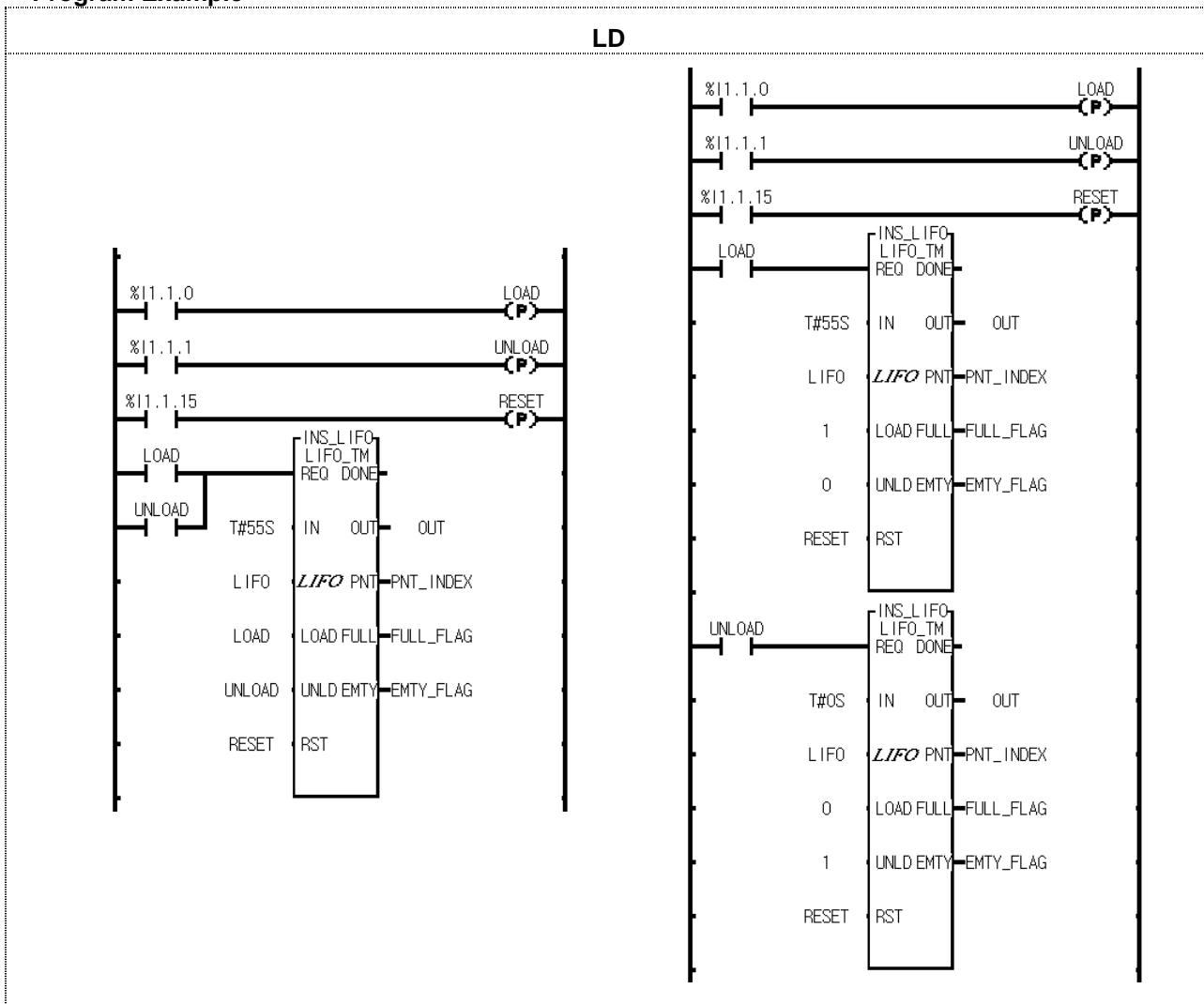
It loads IN to LIFO or unloads data from LIFO.  
 If LOAD and UNLD are on at the same time, input IN is produced as output OUT.  
 If data is unloaded from LIFO by unload function of LIFO\_\*\*\*, unloaded data is deleted in stack and initialized as 0.  
 If RST is loaded to LIFO, PNT is initialized as 0, EMTY is on and all the data of LIFO stack are cleared as 0.  
 The stack number is the array number set by In/Output variable LIFO.  
 If you want to keep the data of LIFO array variables and LIFO function block instance in case that power is off or power failure occurs, set them as 'RETAIN'.  
 Reset functions without REQ input.  
 PNT shows the position of IN to be loaded next time, or the number of pointers to be loaded.  
 If it's on the input mode, output OUT is 0.

## 8. Basic Function/Function Block Library

Function	FIFO variable type	Description
LIFO_Q	BOOL	It functions as LIFO for BOOL-type data
LIFO_B	BYTE	It functions as LIFO for BYTE-type data
LIFO_W	WORD	It functions as LIFO for WORD-type data
LIFO_DW	DWORD	It functions as LIFO for DWORD-type data
LIFO_LW	LWORD	It functions as LIFO for LWORD-type data
LIFO_SI	SINT	It functions as LIFO for SINT-type data
LIFO_I	INT	It functions as LIFO for INT-type data
LIFO_DI	DINT	It functions as LIFO for DINT-type data
LIFO_LI	LINT	It functions as LIFO for LINT-type data
LIFO_USI	USINT	It functions as LIFO for USINT-type data
LIFO_UI	UINT	It functions as LIFO for UINT-type data
LIFO_UDI	UDINT	It functions as LIFO for UDINT-type data
LIFO_ULI	ULINT	It functions as LIFO for ULINT-type data
LIFO_R	REAL	It functions as LIFO for REAL-type data
LIFO_LR	LREAL	It functions as LIFO for LREAL-type data
LIFO_TM	TIME	It functions as LIFO for TIME-type data
LIFO_DAT	DATE	It functions as LIFO for DATE-type data
LIFO_TOD	TOD	It functions as LIFO for TOD-type data
LIFO_DT	DT	It functions as LIFO for DT-type data



■ Program Example



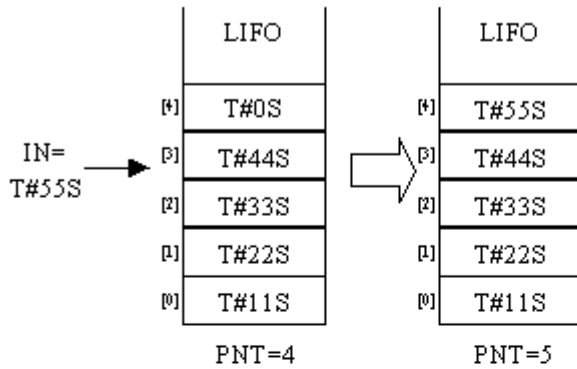
LIFO\_\*\*\* function block is used as the above. The two examples of the above execute the same operation. The left one is a program which executes input and output functions at the same time to use only one function block while the right one is a program which executes input and output functions independently to use input function and output function respectively. Note that the instance name should be the same on the right program.

- (1) If the input conditions (%I1.1.0, %I1.1.1, %I1.1.15) are on, LIFO\_TM is executed.
- (2) If input contact %I1.1.0 is on, load function is executed. T#55S is loaded to LIFO stack and PNT\_INDEX increased by 1.
- (3) If input contact %I1.1.1 is on, unload function is executed. T#55S is unloaded from LIFO stack and PNT\_INDEX decreased by 1.
- (4) If input contact %I1.1.15 is on, reset function is executed. All the stack of LIFO is cleared as T#0S, PNT\_INDEX is initialized as 0 and EMTY\_FLAG is on.

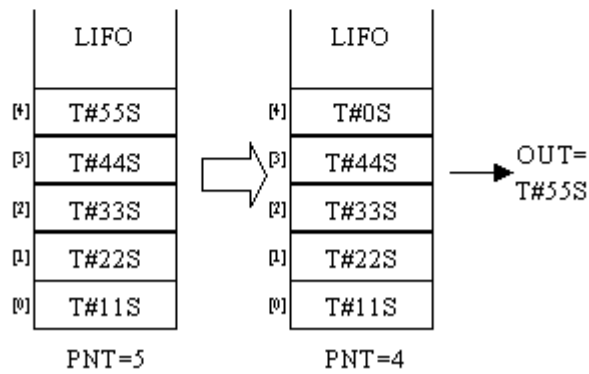
## 8. Basic Function/Function Block Library

---

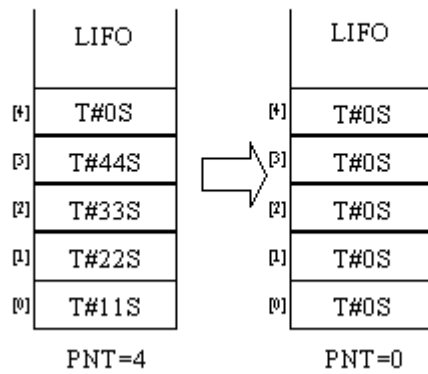
### LOAD (%I1.1.0 is ON)



### UNLOAD (%I1.1.1 is ON)



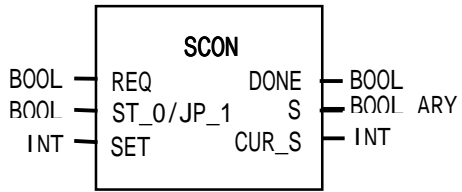
### RESET (%I1.1.15 is ON)



## SCON

Step Controller
-----------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular function block labeled 'SCON'. On the left side, there are three input lines: 'REQ' (labeled 'BOOL'), 'ST_0/JP_1' (labeled 'BOOL'), and 'SET' (labeled 'INT'). On the right side, there are three output lines: 'DONE' (labeled 'BOOL'), 'S' (labeled 'BOOL ARY'), and 'CUR_S' (labeled 'INT').</p>	<p><b>Input</b>            REQ: if it's 1, the function block is executed            S/O: if 0, SET function is enabled;                  if 1, OUT function is enabled.            SET: step number (0 ~ 99)</p> <p><b>Output</b>            DONE: without an error, it will be 1            S: produces an set bit array            CUR_S: produces a current step number</p>

### ■ Function

Setting of step controller group

- The instance name of function block is the name of step controlling group.  
 (Examples of FB declaration: S00, G01, Manu1  
 Examples of step contacts: S00.S[1], G01.S[1], Manu1.S[1])

In case of SET function (ST\_0/JP\_1 = 0)

- In the same step controller group, the present step number can be on when the previous step number is on.
- If the present step number is on, it keeps its state even when the input is off.
- Only one step number is on even when several input conditions are on at the same time.
- If Sxx.S[0] is on, all the SET output is cleared.

In case of JUMP function (ST\_0/JP\_1 = 1)

- In the same step controller group, only one step number is on, even when several input conditions are on.
- If input conditions are on at the same time, last programmed one is produced.
- If the present step number is on, it keeps its state even when the input is off.
- If Sxx.S[0] is on, it returns to its first step.

### ■ Error

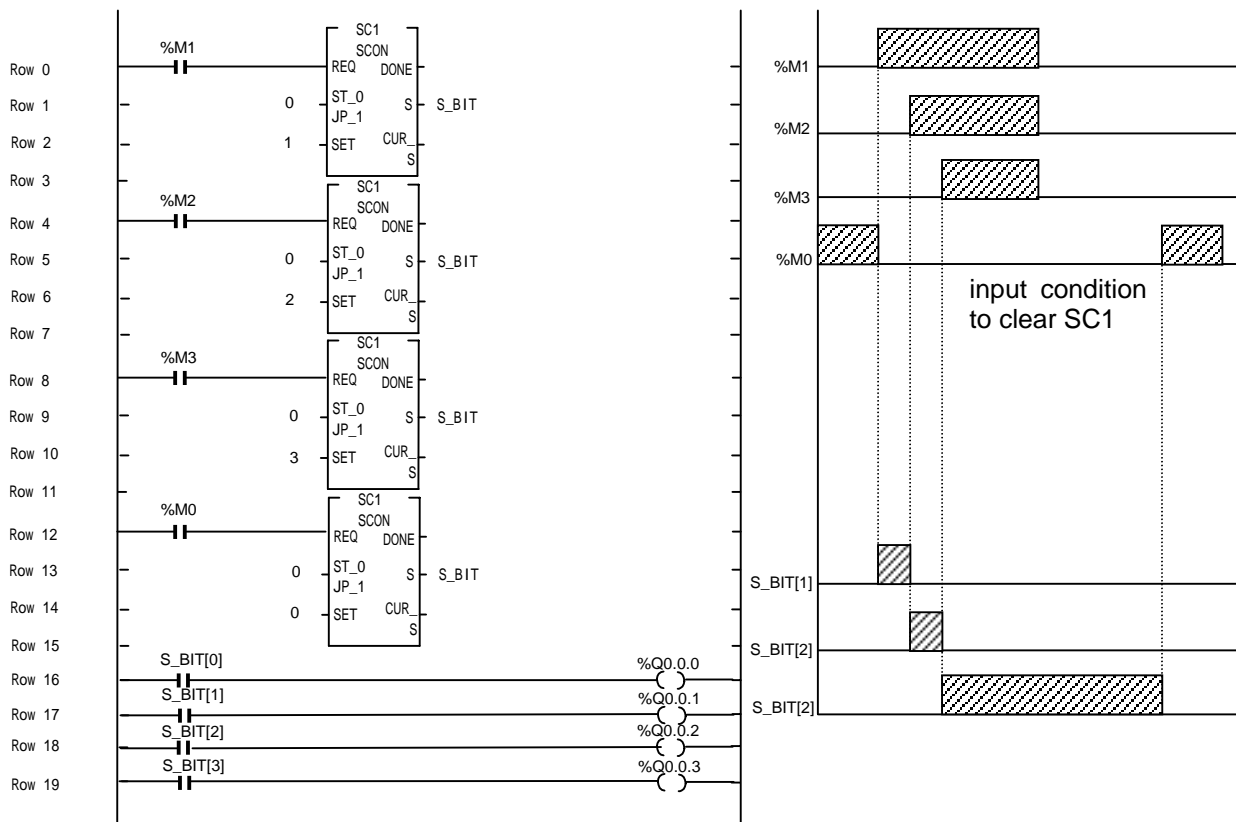
An error occurs when step setting (SET) is out of its range (0 ~ 99).

If an error occurs, DONE is off and step output maintains its previous step.

## 8. Basic Function/Function Block Library

### ■ Program Example

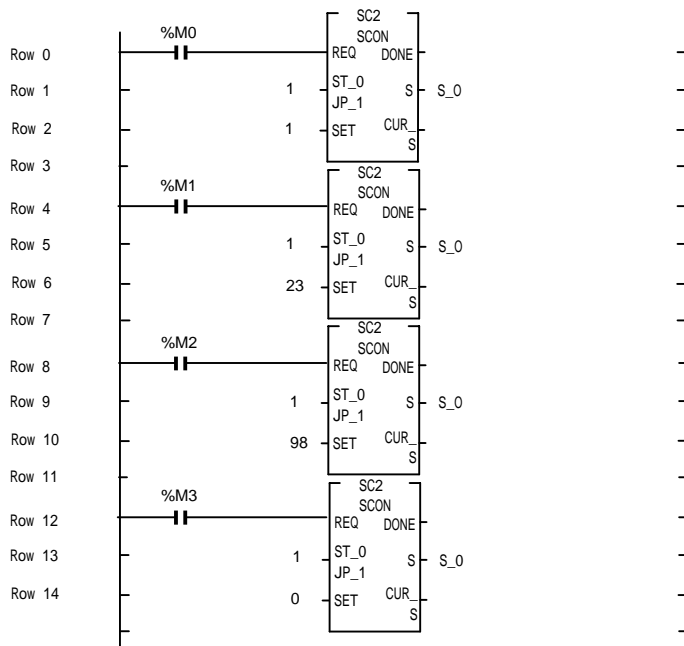
In case of SET function (ST\_0/JP\_1 = 0), using SC1 group



Step control produces an output when the previous step is on and its present condition is on.

■ Program Example

In case of JUMP function (ST\_0/JP\_1 = 1), using SC2 group (last input priority)



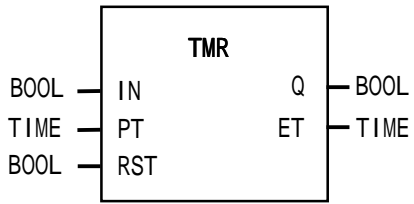
NO	%M1	%M2	%M3	%M4	S_O[1]	S_O[23]	S_O[98]	S_O[0]
1	On	Off	Off	Off	○			
2	On	On	Off	Off		○		
3	On	On	On	Off			○	
4	On	On	On	On				○

## 8. Basic Function/Function Block Library

### TMR

Integration Timer
-------------------

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular block labeled 'TMR'. On the left side, there are three input terminals: 'IN' (labeled 'BOOL'), 'PT' (labeled 'TIME'), and 'RST' (labeled 'BOOL'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'ET' (labeled 'TIME').</p>	<p><b>Input</b></p> <p>IN: operation condition for Timer            PT: preset time            RST: reset</p> <p><b>Output</b></p> <p>Q: timer output            ET: elapsed time</p>

#### ■ Function

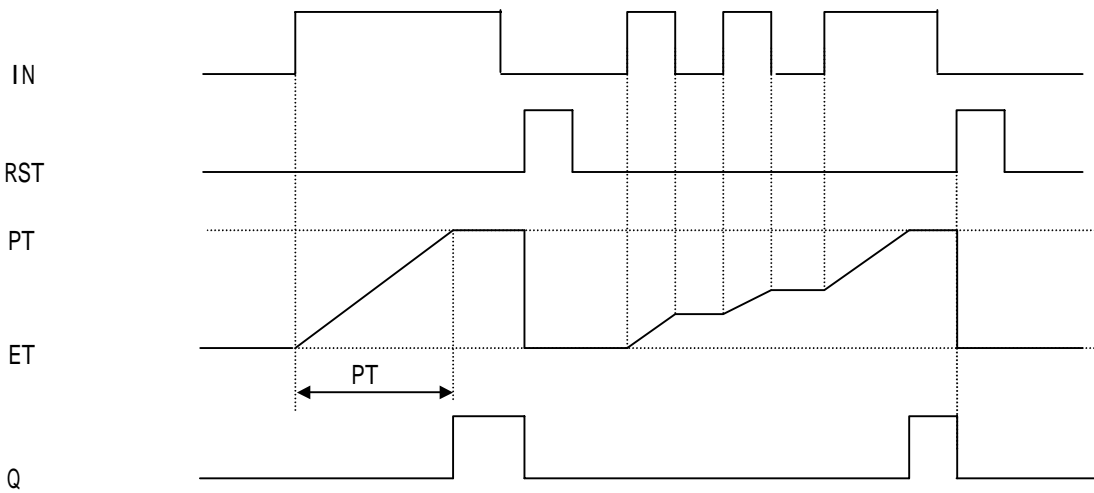
When IN is 1, elapsed time is produced at ET.

Even if IN is 0 before ET reaches PT, ET keeps its value. If IN is 1 again, elapsed time is produced at ET integrating its previous value.

If ET reaches PT, Q is 1..

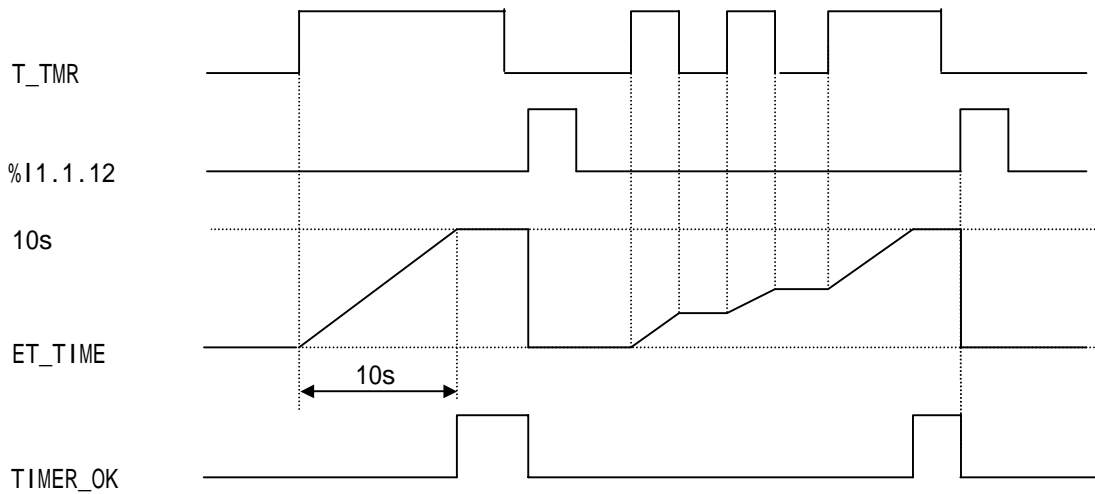
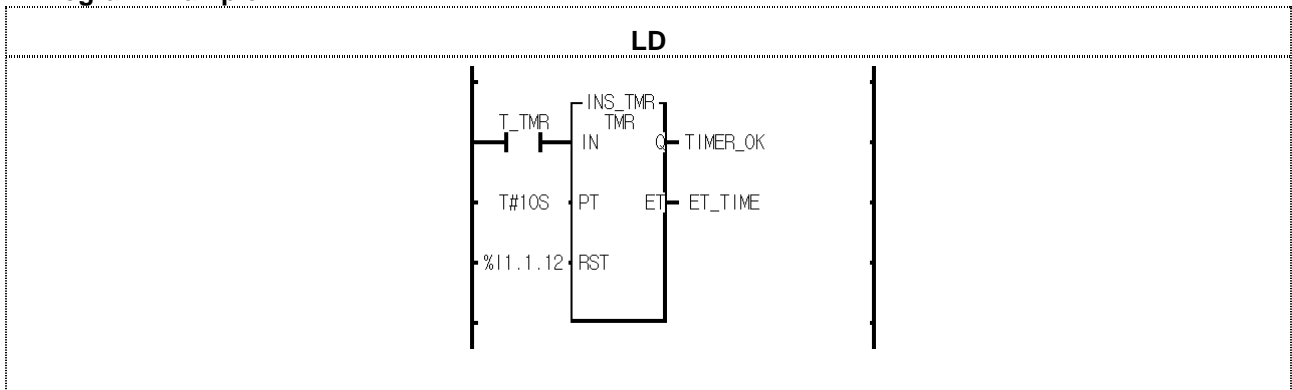
If RST is 1, Q and ET are 0.

#### ■ Timing Chart





■ Program Example



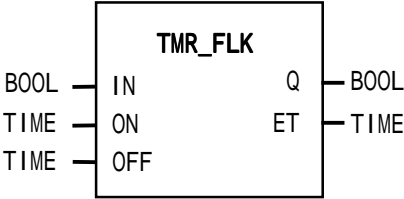
- (1) If 10 seconds passes after input variable T\_TMR is 1, output variable TIMER\_OK is 1.
- (2) Elapsed time is produced at ET\_TIME after T\_TMR is 1.
- (3) ET\_TIME keeps its value even if T\_TMR is 0 before ET\_TIME reaches its preset time 10 seconds.
- (4) If T\_TMR is 1, elapsed time is produced at ET\_TIME integrating its previous value.
- (5) If input contact %I1.1.12 is 1, elapsed time ET\_TIME and output variable TIMER\_OK are all cleared.

## 8. Basic Function/Function Block Library

### TMR\_FLK

TMR with Flicker

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b></p> <p>IN: operation condition for Timer            ON: TON setting time            OFF: TOF setting time</p> <p><b>Output</b></p> <p>Q: Timer output            ET: elapsed time</p>

#### ■ Function

If IN is 1, Q is 1 and maintains its value during TON setting time.

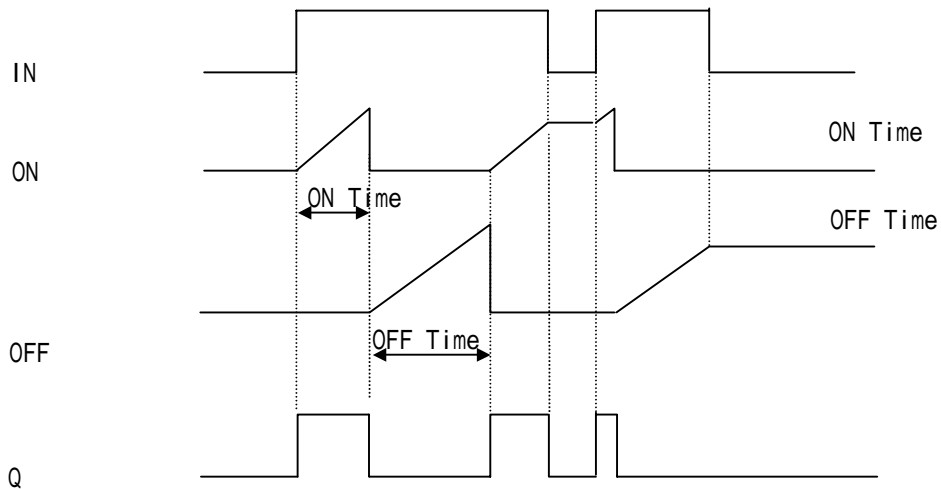
After TON setting time set by ON, Q is 0 during TOF setting time.

If IN is 0, it stops its function of either on or off operation and keeps its time. If IN is 1 again, it is executed with its previous data.

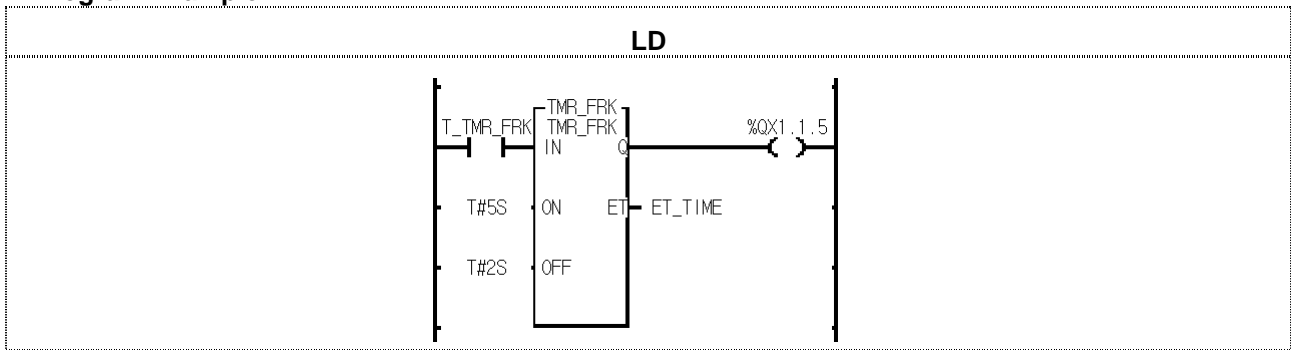
Output Q is 0 while IN is 0.

If ON is 0, output Q is always 0.

#### ■ Timing Chart



■ Program Example



- (1) If input variable T\_TMR\_FRK is 1, TMR\_FRK function block is executed.
- (2) Output contact %QX1.1.5 is 1 during 5 seconds set by ON after input variable T\_TMR\_FRK is 1.
- (3) Output contact %QX1.1.5 is 0 during 2 seconds set by OFF after 5 seconds set by ON.
- (3) TON time (ON) when Q is 1 and TOF time (OFF) when Q is 0 are produced at ET\_TIME by turns while T\_TMR\_FRK is 1.
- (4) If input variable T\_TMR\_FRK is 0, then it keeps its time and output contact %QX1.1.5 is 0. If T\_TMR\_FRK is 1, it is executed again.

## 8. Basic Function/Function Block Library

### TMR\_UNIT

TMR with integer setting

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b></p> <p>IN: operation condition for Timer            PT: preset time            UNIT: time unit of setting time            RST: reset input</p> <p><b>Output</b></p> <p>Q: timer output            ET: elapsed time</p>

#### ■ Function

Elapsed time is produced at ET after IN is 1.

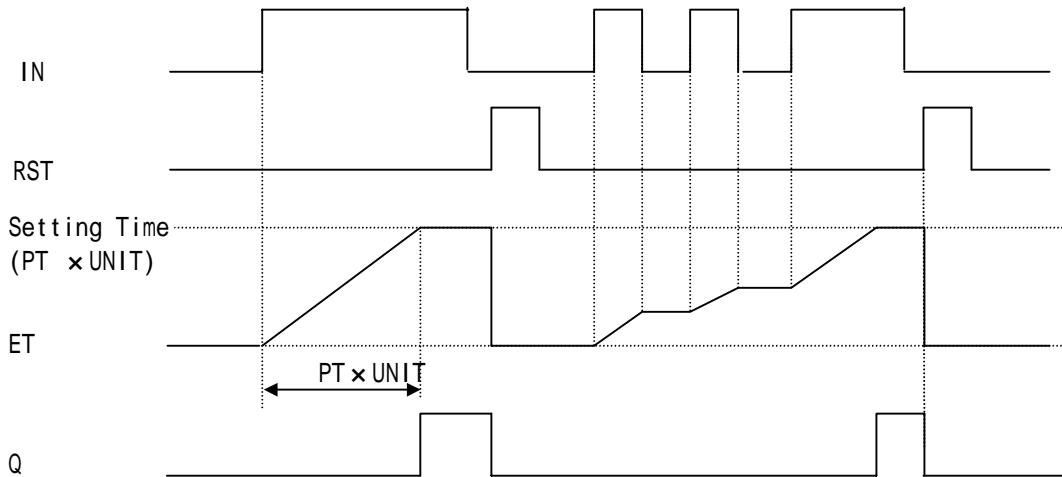
Even if IN is 0 before ET reaches PT, ET keeps its value. If IN is 1 again, elapsed time is produced at ET integrating its previous value.

Q is 1 when elapsed time reaches preset time.

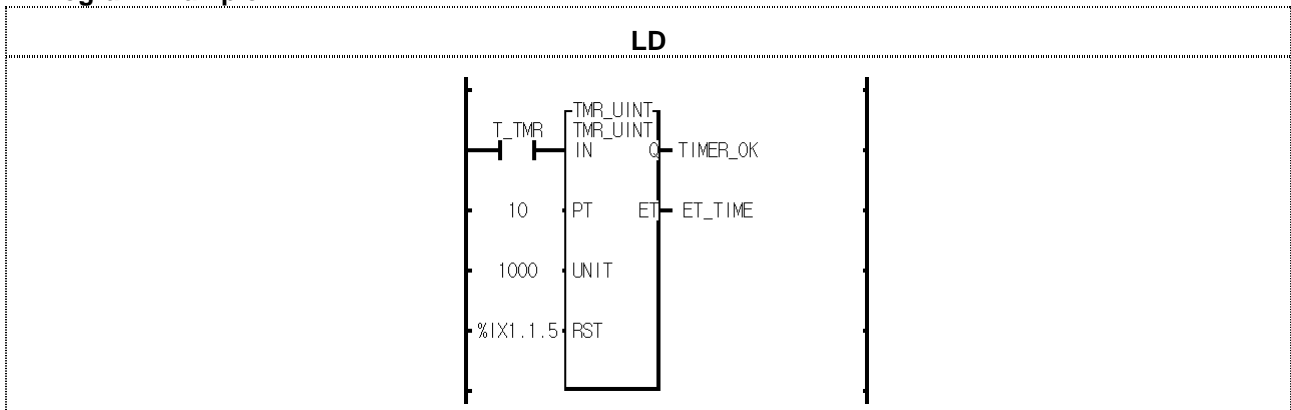
If RST is 1, Q and ET are 0.

Setting time is  $PT \times UNIT$  (ms).

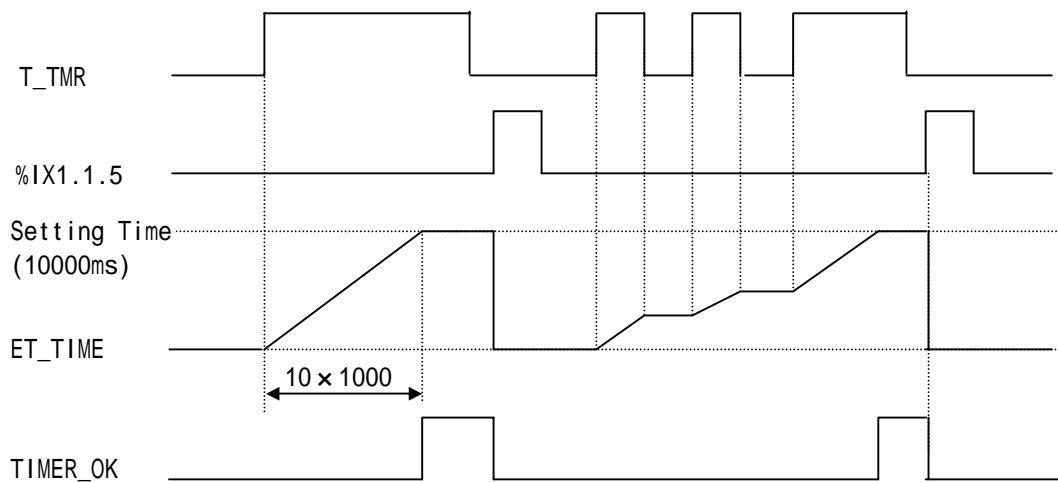
#### ■ Timing Chart



■ Program Example



- (1) Setting time is  $PT \times UNIT[ms] = 10 \times 1000[ms] = 10[s]$ .
- (2) Output variable TIMER\_OK is 1, if 10 seconds passes after input variable T\_TMR is 1.
- (3) Elapsed time is produced at ET\_TIME after input variable T\_TMR is 1.
- (4) Even if T\_TMR is 0 before ET\_TIME reaches preset time 10 seconds, ET\_TIME keeps its value.
- (5) If input variable T\_TMR is 1 again, elapsed time is produced at ET integrating its previous value.
- (6) If input contact %IX1.1.5 is 1, elapsed time ET\_TIME and output contact TIMER\_OK are all cleared.

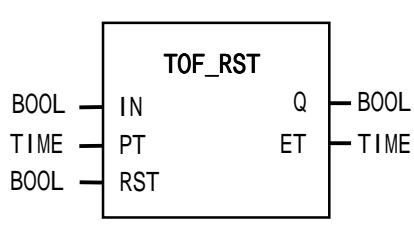


## 8. Basic Function/Function Block Library

### TOF\_RST

TOF with Reset

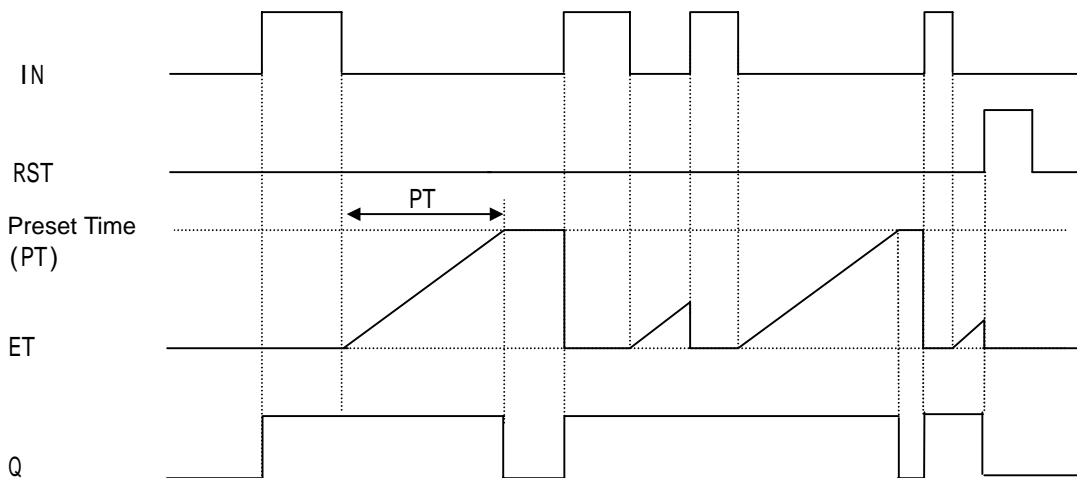
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular function block labeled 'TOF_RST'. On the left side, there are three input terminals: 'IN' (labeled 'BOOL'), 'PT' (labeled 'TIME'), and 'RST' (labeled 'BOOL'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'ET' (labeled 'TIME').</p>	<p><b>Input</b>            IN: operation condition for Timer            PT: preset time            RST: reset</p> <p><b>Output</b>            Q: Timer output            ET: elapsed time</p>

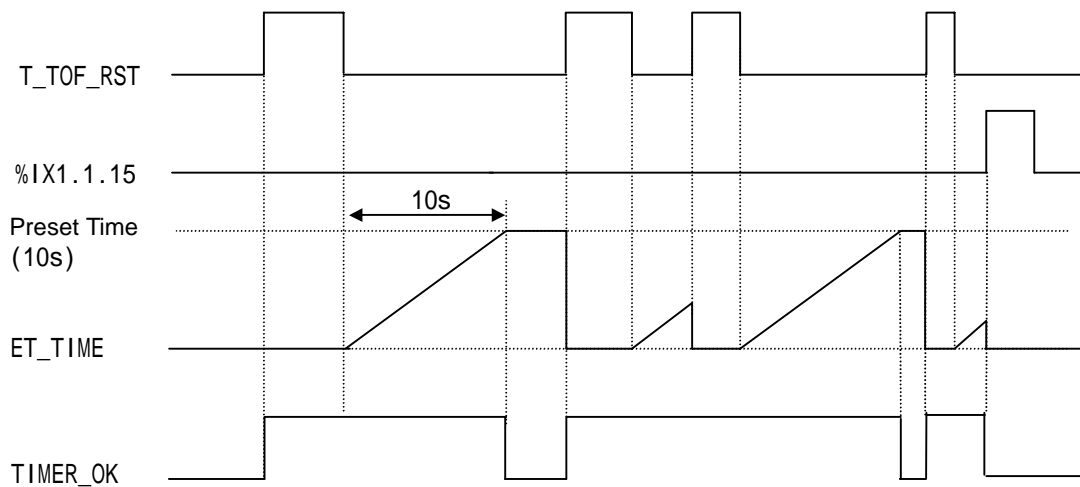
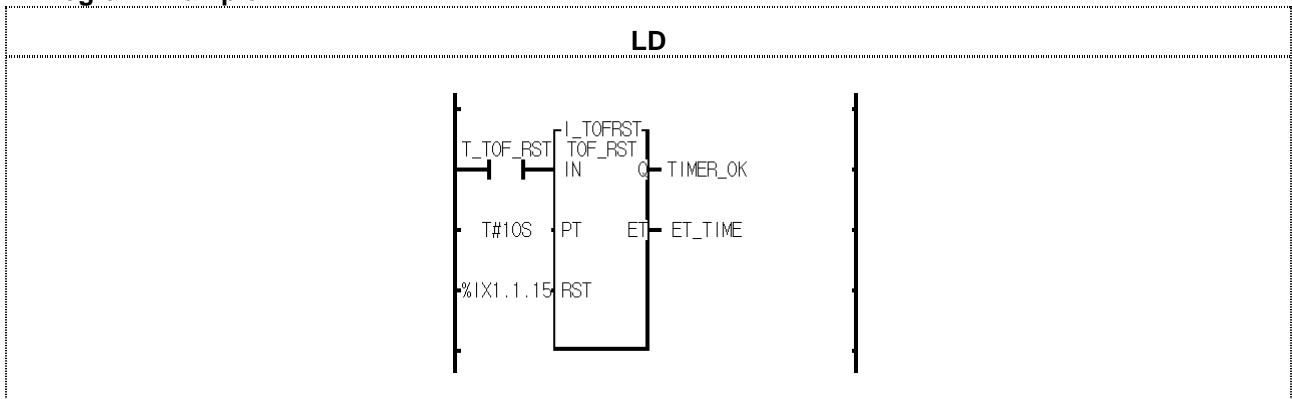
#### ■ Function

Q is 1 when IN is 1 and Q is 0 after preset time (PT) after IN is 0.  
 Elapsed time is produced at ET after IN is 0.  
 Elapsed time is 0 if IN is 1 before ET reaches PT.  
 If RST is 1, Q and ET are 0.

#### ■ Timing Chart



■ Program Example



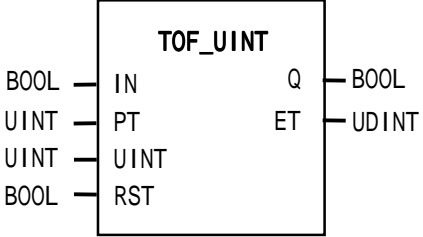
- (1) If input variable T\_TOF\_RST is 1, output variable TIMER\_OK is 1. And TIMER\_OK is 0 after 10 seconds after T\_TOF\_RST is 0.
- (2) If T\_OF\_RST is 1 within 10 seconds after it turns off, TOF\_RST is initialized.
- (3) Elapsed time is produced at ET\_TIME.
- (4) If input contact %IX1.1.15 is 1, elapsed time ET\_TIME and output contact TIMER\_OK are all cleared.

## 8. Basic Function/Function Block Library

### TOF\_UINT

TOF with integer setting

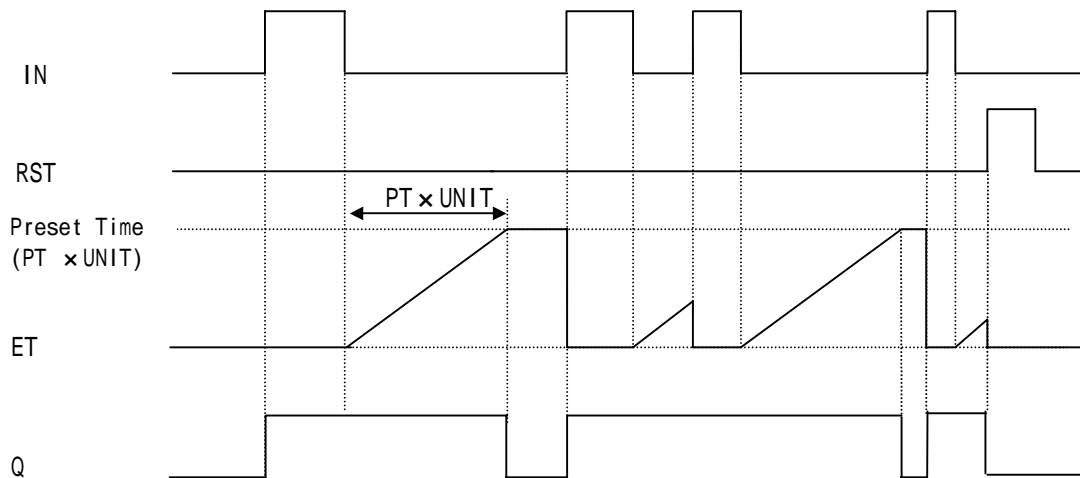
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular block labeled TOF_UINT. On the left side, there are four input lines: IN (labeled BOOL), PT (labeled UINT), UUNIT (labeled UINT), and RST (labeled BOOL). On the right side, there are two output lines: Q (labeled BOOL) and ET (labeled UDINT).</p>	<p><b>Input</b></p> <p>IN: operation condition for Timer  PT: preset time  UNIT: time unit of setting time  RST: reset</p> <p><b>Output</b></p> <p>Q: Timer output  ET: elapsed time</p>

#### ■ Function

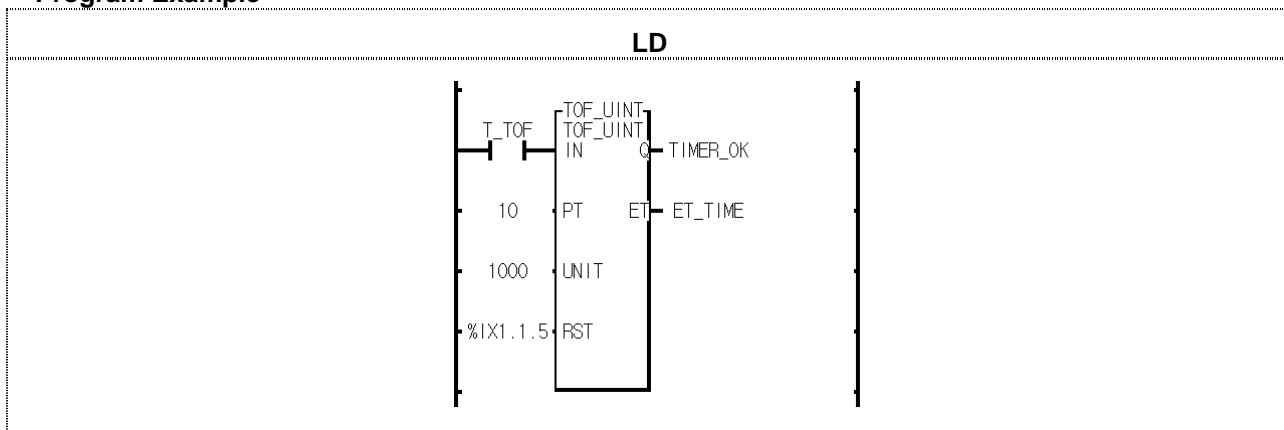
Q is 1 when IN is 1. And Q is 0, if setting time (PT) passes after IN is 0.  
Elapsed time is produced at ET after IN is 0.  
If IN is 1 before ET reaches PT, ET is 0.  
If RST is 1, Q and ET are 0.  
Setting time is  $PT \times UNIT$  (ms).

#### ■ Timing Chart

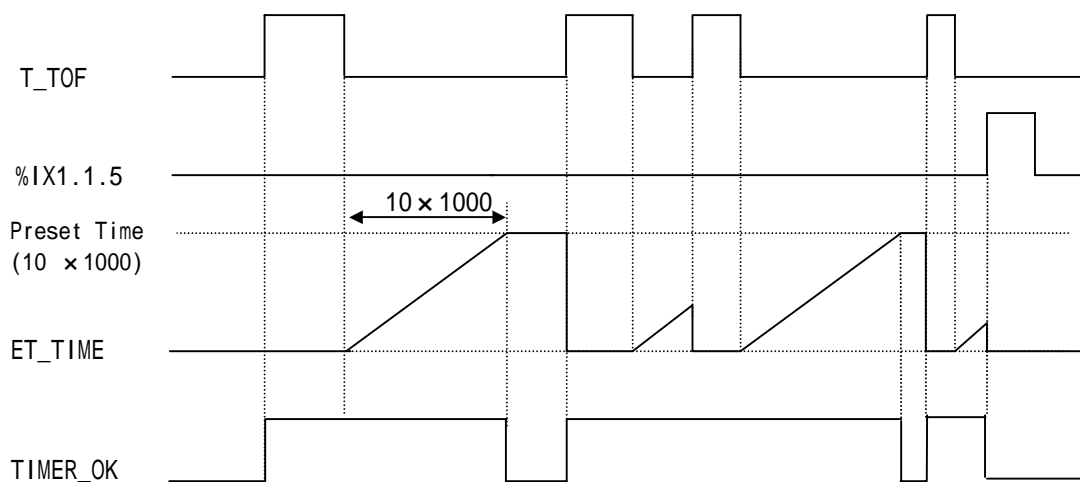




■ Program Example



- (1) Preset time  $PT \times UNIT[ms] = 10 \times 1000[ms] = 10[s]$ .
- (2) If input variable T\_TOF is 1, output variable TIMER\_OK is 1. TIMER\_OK is 0, if 10 seconds pass after T\_TOF is 0.
- (3) If T\_TOF is 1 within 10 seconds, TOF\_UINT is initialized.
- (4) Elapsed time is produced at ET\_TIME.
- (5) If input contact %IX1.1.5 is 1, TIMER\_OK and ET\_TIME are all cleared.

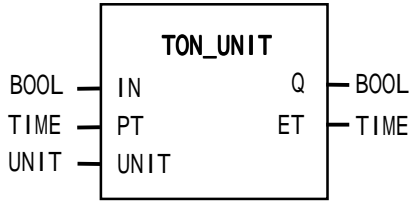


## 8. Basic Function/Function Block Library

### TON\_UNIT

TON with integer setting

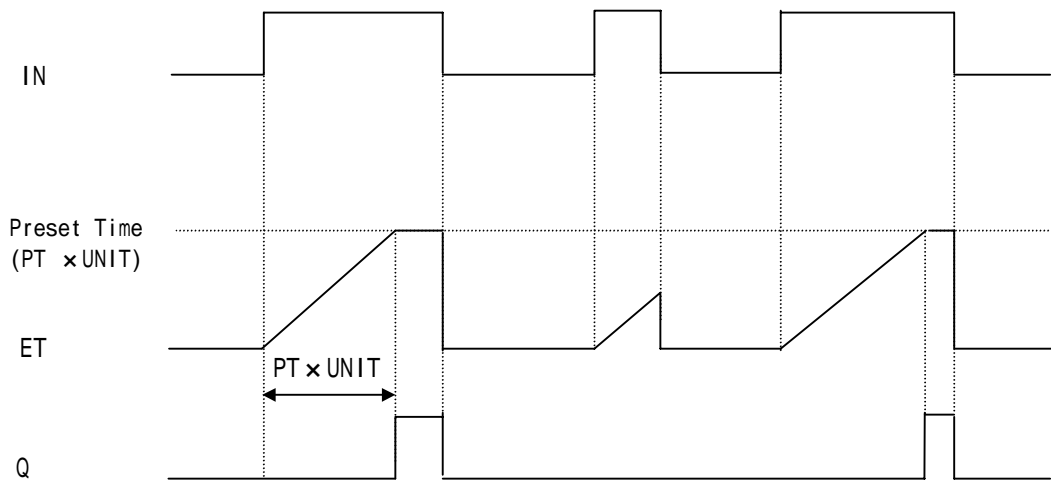
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular function block labeled 'TON_UNIT'. On the left side, there are three input terminals: 'IN' (labeled 'BOOL'), 'PT' (labeled 'TIME'), and 'UNIT' (labeled 'TIME'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'ET' (labeled 'TIME').</p>	<p><b>Input</b></p> <p>IN: operation condition for Timer  PT: preset time  UNIT: time unit of setting time</p> <p><b>Output</b></p> <p>Q: timer output  ET: elapsed time</p>

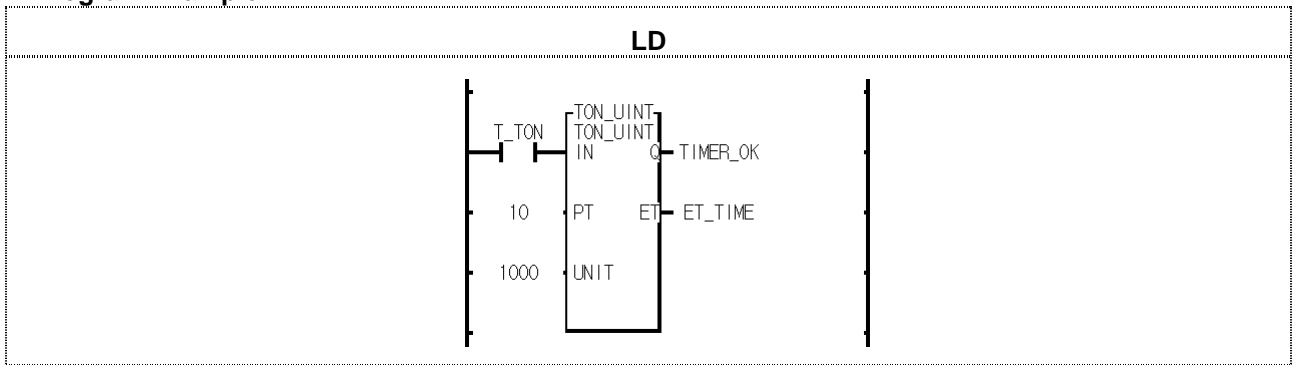
#### ■ Function

- Elapsed time is produced at ET after IN is 1.
- Elapsed time ET is 0, if IN is 0 before ET reaches PT.
- Q is 0, if IN is 0 after Q is 1.
- Preset time is  $PT \times UNIT$ [ms].

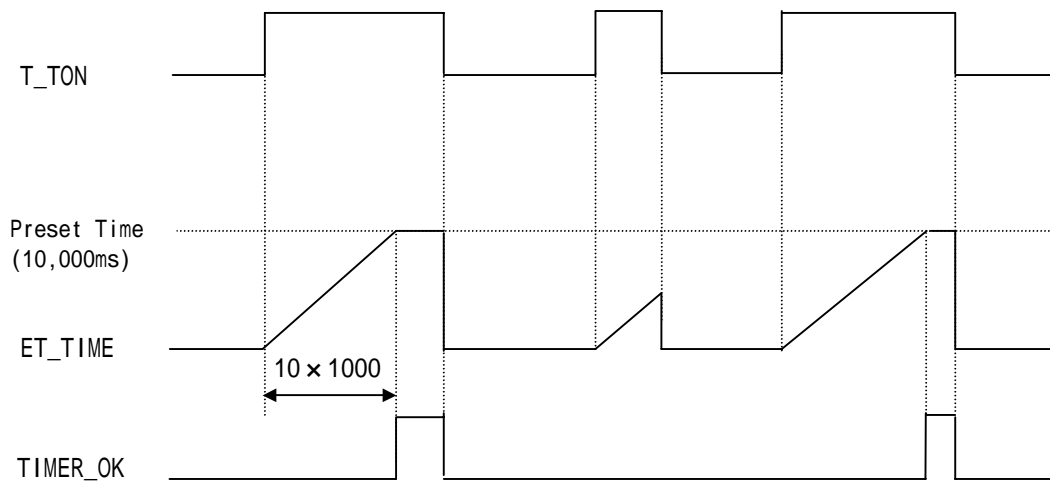
#### ■ Timing Chart



■ Program Example



- (1) Preset time is  $PT \times UNIT[s] = 10 \times 1000[s] = 10[s]$ .
- (2) If 10 seconds passes after input variable T\_TON is on, output variable TIMER\_OK is 1.
- (3) Elapsed time is produced at ET\_TIME after input variable T\_TON is on.
- (4) If T\_TON is 0 before elapsed time ET\_TIME reaches 10 seconds, ET\_TIME is 0.
- (5) If T\_TON is 0 after TIMER\_OK is 1, TIMER\_OK and ET\_TIME are 0.

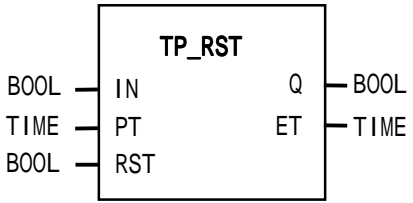


## 8. Basic Function/Function Block Library

### TP\_RST

TP with Reset
---------------

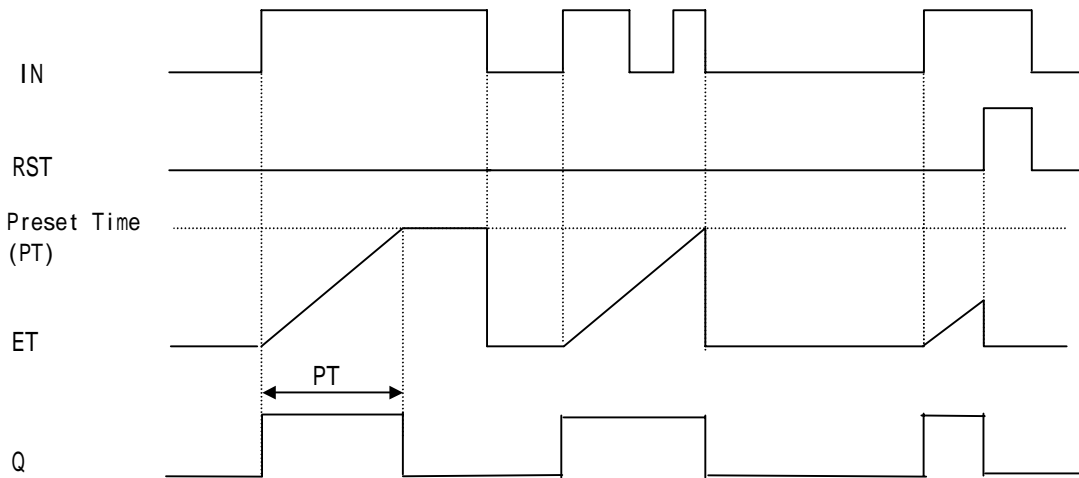
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular block labeled 'TP_RST'. On the left side, there are three input terminals: 'IN' (labeled 'BOOL'), 'PT' (labeled 'TIME'), and 'RST' (labeled 'BOOL'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'ET' (labeled 'TIME').</p>	<p><b>Input</b>            IN: operation condition for Timer            PT: preset time            RST: reset</p> <p><b>Output</b>            Q: timer output            ET: elapsed time</p>

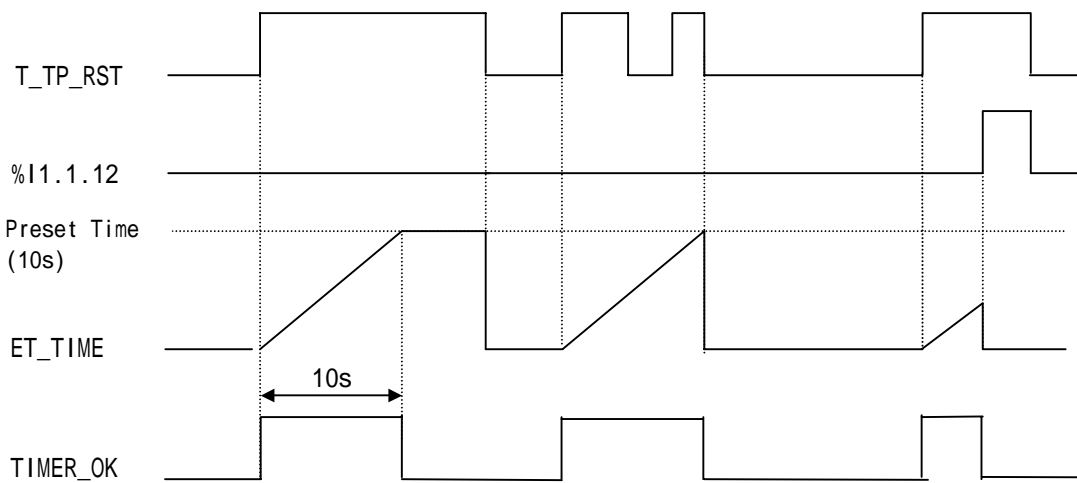
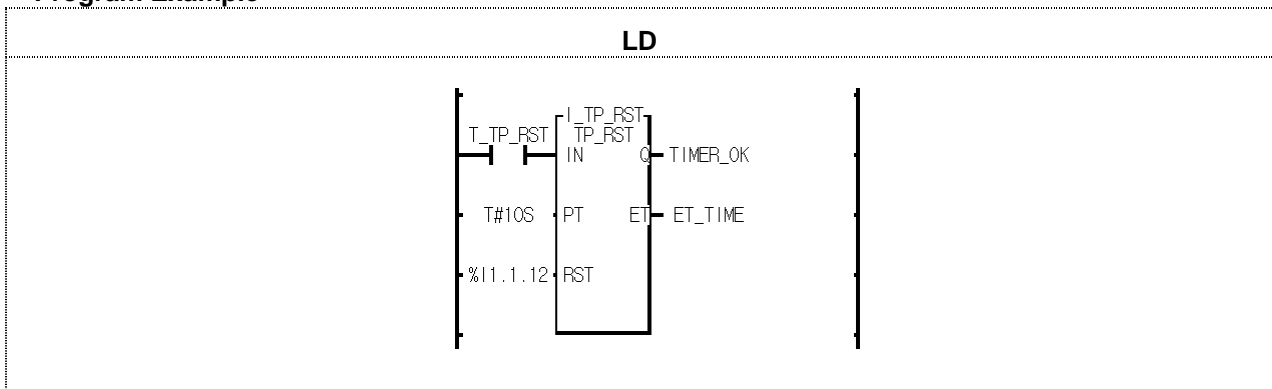
#### ■ Function

If IN is 1, Q is 1. And if elapsed time reaches preset time, timer output Q is 0.  
 ET increases its value from when IN is 1, keeps its value at PT and is cleared when IN is 0.  
 It doesn't matter whether IN changes its state or not while timer output Q is 1 (during a pulse output).  
 If RST is 1, output Q and ET are 0.

#### ■ Timing Chart



■ Program Example



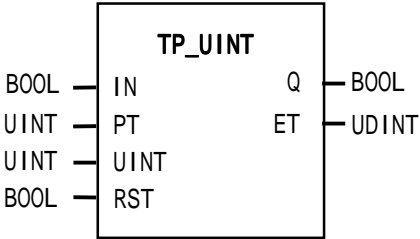
- (1) If input variable `T_TP_RST` is 1, output variable `TIMER_OK` is 1. And 10 seconds later, `TIMER_OK` is 0. Once `TP_RST` timer is executed, input `T_TP_RST` doesn't matter.
- (2) `ET_TIME` value increases and stops at 10S. And if `T_TP_RST` is 0, it is 0.
- (3) If input contact `%I1.1.12` is 1, `TIMER_OK` and `ET_TIME` are all cleared.

## 8. Basic Function/Function Block Library

### TP\_UINT

TP with integer setting

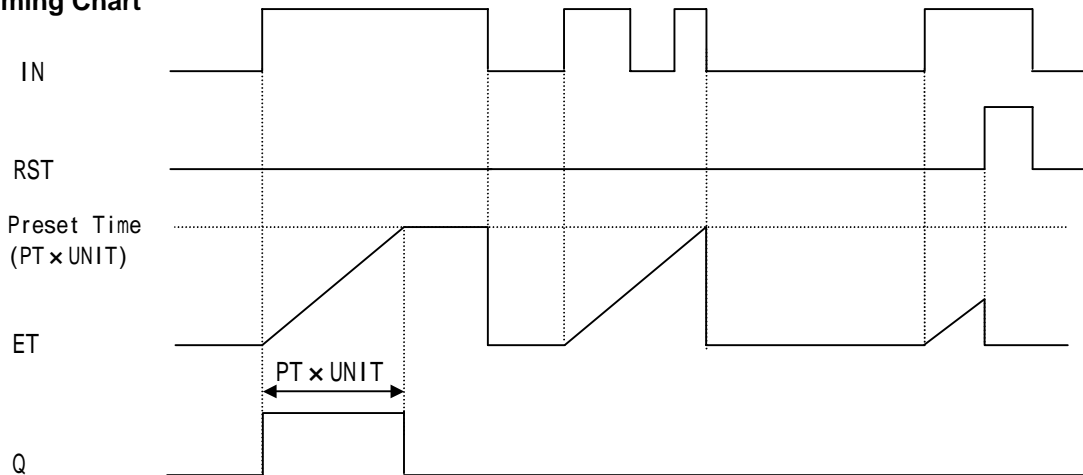
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b></p> <p>IN: operation condition for Timer            PT: preset time            UNIT: time unit of setting time            RST: reset</p> <p><b>Output</b></p> <p>Q: timer output            ET: elapsed time</p>

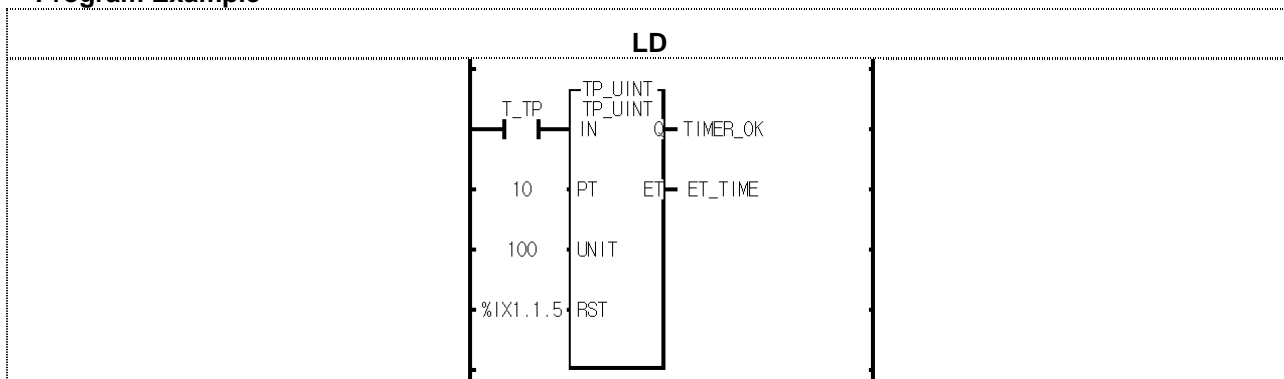
#### ■ Function

If IN is 1, Q is 1. And if elapsed time reaches preset time, timer output Q is 0.  
 ET increases its value from when IN is 1, keeps its value at PT and is cleared when IN is 0.  
 It doesn't matter whether IN changes its state or not while timer output Q is 1 (during a pulse output).  
 If RST is 1, output Q and ET are 0.  
 Preset time is  $PT \times UNIT[ms]$ .

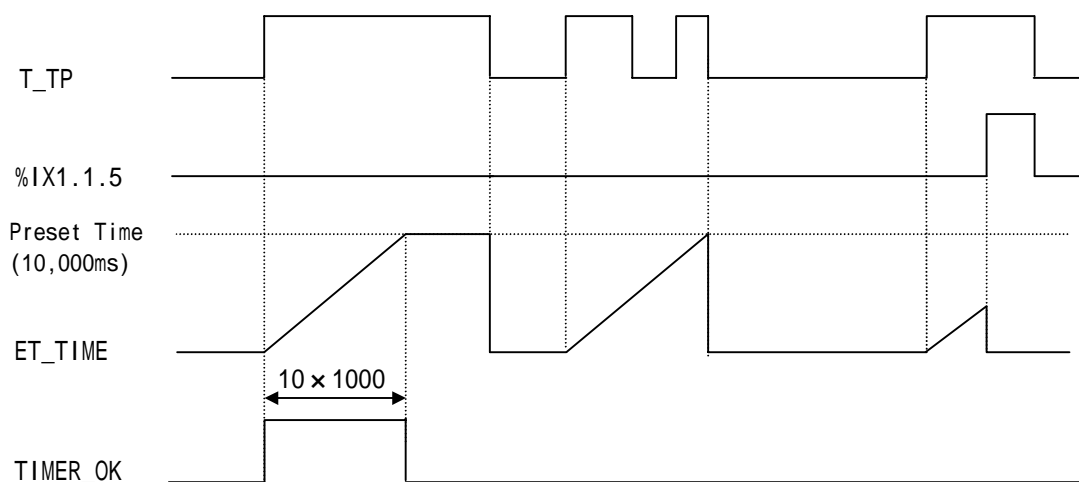
#### ■ Timing Chart



■ Program Example



- (1) Preset time is  $PT \times UNIT[s] = 10 \times 1000[s] = 10[s]$ .
- (2) If input variable **T\_TP** is 1, output variable **TIMER\_OK** is 1. And 10 seconds later, **TIMER\_OK** is 0. Once **TP\_UINT** timer is executed, input **T\_TP** doesn't matter.
- (3) **ET\_TIME** value increases and stops at 10000. And if **T\_TP** is 0, it is 0.
- (4) If input contact **%IX1.1.5** is 1, **TIMER\_OK** and **ET\_TIME** are all cleared.

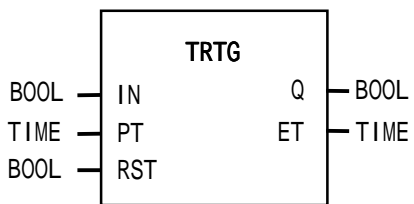


## 8. Basic Function/Function Block Library

### TRTG

Retriggerable Timer

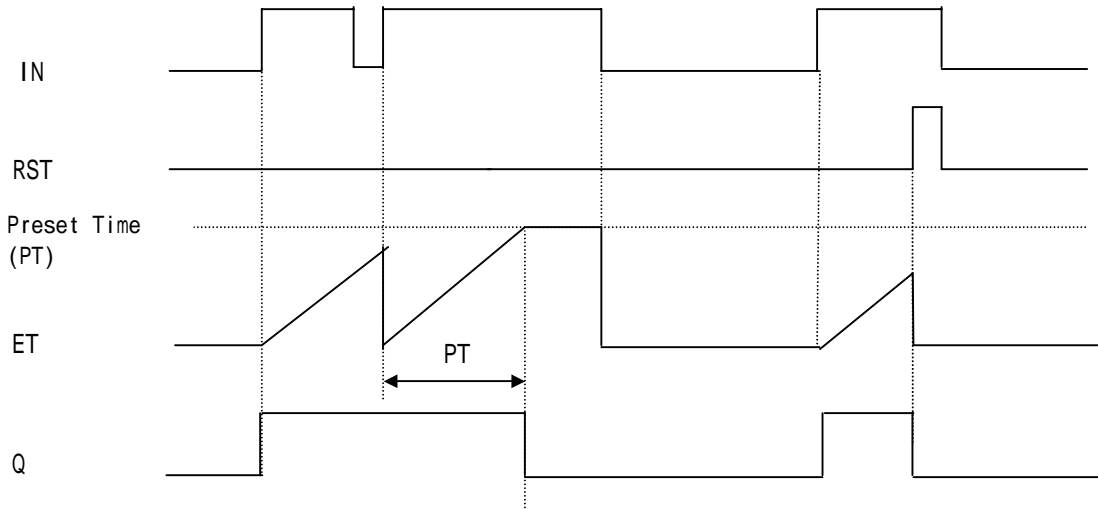
Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
 <p>The diagram shows a rectangular block labeled 'TRTG'. On the left side, there are three input terminals: 'IN' (labeled 'BOOL'), 'PT' (labeled 'TIME'), and 'RST' (labeled 'BOOL'). On the right side, there are two output terminals: 'Q' (labeled 'BOOL') and 'ET' (labeled 'TIME').</p>	<p><b>Input</b>            IN: operation condition for Timer            PT: preset time            RST: reset</p> <p><b>Output</b>            Q: timer output            ET: elapsed time</p>

#### ■ Function

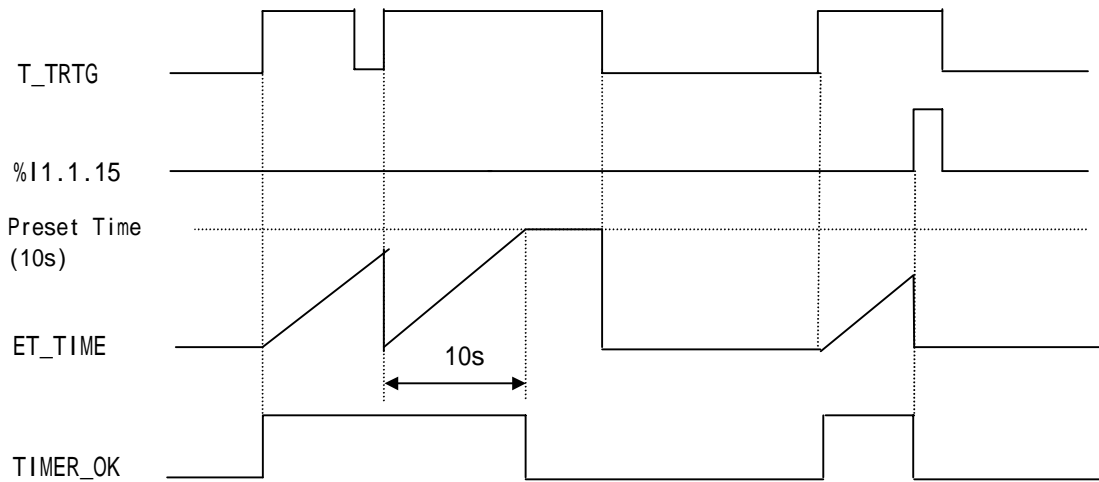
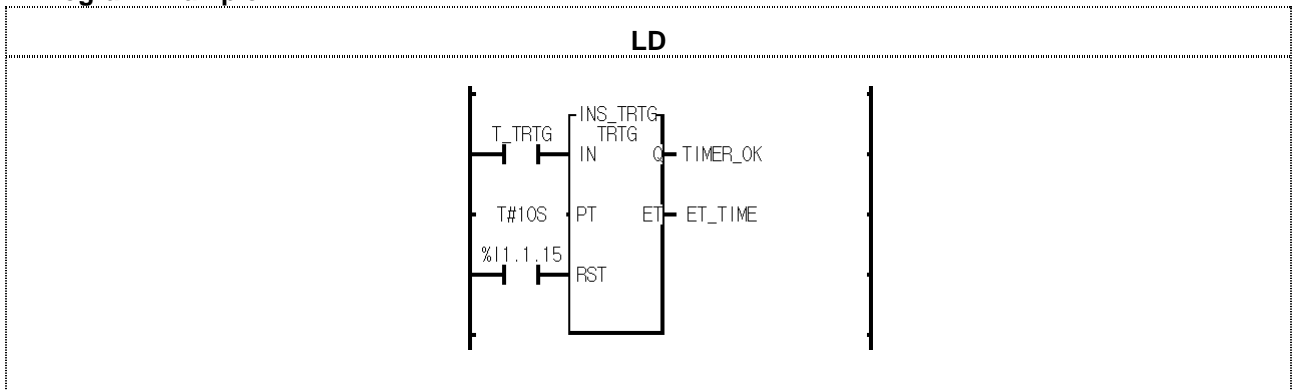
- If IN is 1, Q is 1. And if elapsed time reaches preset time, timer output Q is 0.
- If IN turns on again before elapsed time reaches preset time, then elapsed time is set as 0 and increased again. And if it reaches PT, Q is 0.
- If RST is 1, timer output Q and elapsed time ET are 0.

#### ■ Timing Chart





■ Program Example



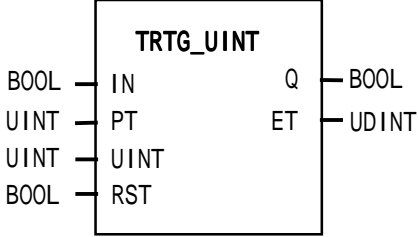
- (1) TIMER\_OK is 1 during 10 seconds after input variable T\_TRTG becomes 1 from 0. If T\_TRTG becomes 1 from 0 after timer is executed, ET\_TIME is set as 0 and increased again.
- (2) TIMER\_OK is 1 during 10 seconds even when T\_TRTG becomes 0 from 1.
- (3) ET\_TIME value increases and stops at T#10S. And it is 0 when T\_TRTG is 0.
- (4) If input contact %I1.1.15 is 1, TIMER\_OK and ET\_TIME are all cleared.

## 8. Basic Function/Function Block Library

### TRTG\_UINT

TRTG with integer setting

Model	GMR	GM1	GM2	GM3	GM4	GM6	GM7
Application							

Function Block	Description
	<p><b>Input</b></p> <p>IN: operation condition for Timer            PT: preset time            UNIT: time unit of setting time            RST: reset</p> <p><b>Output</b></p> <p>Q: timer output            ET: elapsed time</p>

#### ■ Function

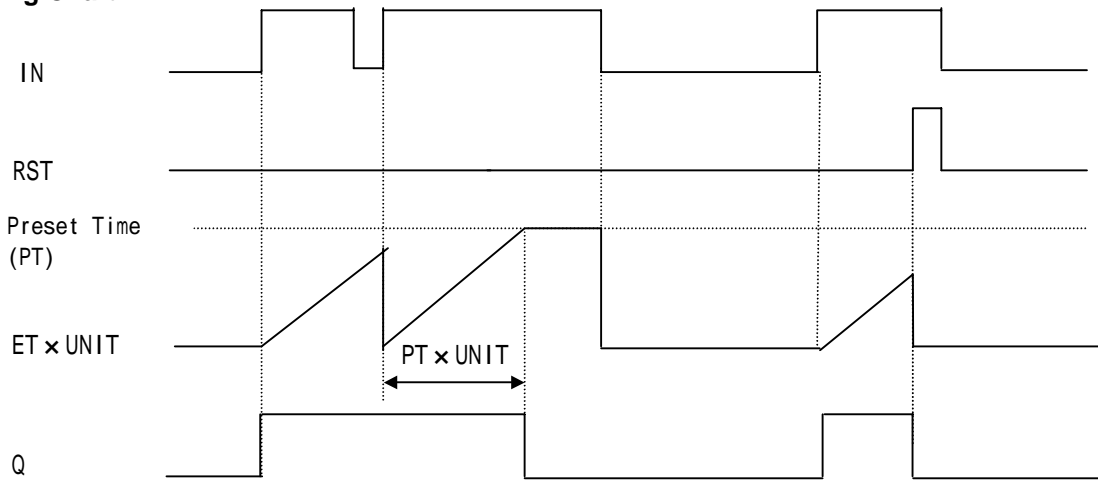
If IN is 1, Q is 1. And if elapsed time reaches preset time, timer output Q is 0.

If IN turns on again before elapsed time reaches preset time, then elapsed time is set as 0 and increased again. And if it reaches PT, Q is 0.

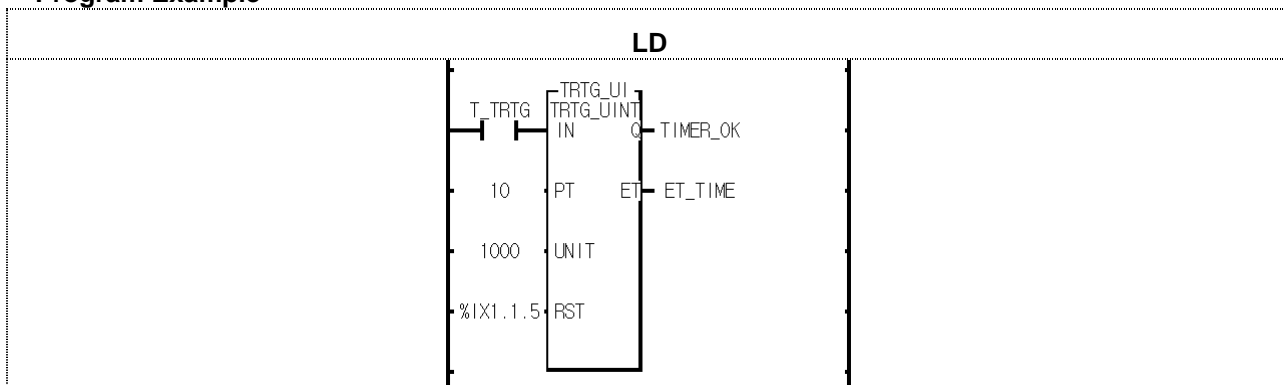
If RST is 1, timer output Q and elapsed time ET are 0.

Preset time is  $PT \times UNIT[ms]$ .

#### ■ Timing Chart



■ Program Example



- (1) Preset time is  $PT \times UNIT[ms] = 10 \times 1000[ms] = 10[s]$ .
- (2) TIMER\_OK is 1 during 10 seconds after input variable T\_TRTG becomes 1 from 0. If T\_TRTG becomes 1 from 0 after timer is executed, ET\_TIME is set as 0 and increased again.
- (3) TIMER\_OK is 1 during 10 seconds even when T\_TRTG becomes 0 from 1.
- (4) ET\_TIME value increases and stops at 10000. And it is 0 when T\_TRTG is 0.
- (5) If input contact %IX1.1.5 is 1, TIMER\_OK and ET\_TIME are all cleared.

