Content

Chapter 1 Introduction

Chapter 2 Software structure

Chapter 3 Common element

Chapter 4 SFC(Sequential Function Chart)

Chapter 5 IL(Instruction List)

Chapter 6 LD(Ladder Diagram)

Chapter 7 Functions and function blocks

Chapter 8 Function/Function block libraries

Chapter 10 Communication function block libraries

Chapter 1 Introduction

1. Introduction

This manual describes the language for the G Series range of PLC.

GLOFA PLC is based on the international standard language defined in IEC (International Electrotechnical Commission) 1131-3.

1.1. Characteristics of IEC 1131-3 language

Main characteristics introduced to IEC language are as below.

- Support to various and strong data type.
- Top-down or bottom-up design is available by adapting the program configuration element such as function, function block, and the PLC program can be prepared structurally.
- The program prepared by the user can be librarized to be used for other applications so that the software is reused.
- The user can select the most apprpriate language for his application, since various languages are supported.

1.2. Language type

PLC language standardized by IEC consists of two graphical languages, two textual languages and SFC.

- Graphical language
  a) LD(Ladder Diagram) : Relay logic type language
  b) FBD(Function Block Diagram) : Language expressing the program linked the block function
- Textual language
  a) IL(Instruction List) : Language of the assembly language type
  b) ST(Structured Text) : High level language of the Pascal type
- SFC(Sequential Function Chart)

G Series PLC supports the language of IL, LD, and SFC.

MEMO

Chapter 2 Software structure

2. Software structure

2.1. Overview

Total PLC system shall be configured before preparing PLC application program. Total PLC system is defined as one project in GLOFA PLC. All items required for one PLC system is defined hierarchically in this project.

- Project
  - Configuration
    - Resource
      - Program
      - Resource global variable
      - Task
    - Configuration global variable
    - Access variable
  - Parameter
    - Basic parameter
    - I/O parameter
    - Link parameter

2.2. Project

- The project shall be configured to prepare G Series PLC program. The configuration of one project means that all configuration elements are prepared for one PLC system. The basic scan program(general PLC program), basic parameter, task definition, I/O parameter etc. are prepared.
- The project consists of two parts, the configuration section and the parameter section. The configuration section defines the global variable, program, task etc. and controls the related settings. The parameter section prepare various information for the PLC system operation. Please refer to GMWIN user manual for the parameter section.

2.3. Configuration

- The configuration means one PLC system. One PLC system consists of the base, CPU module, I/O module, special module etc. Generally, one PLC system contains one CPU module
- PLC system has its designation, named as the configuration designation. This designation is a unique name for each PLC when the communication is performed between PLCs. The configuration can be named up to 8 digits according to clause 3.1.1. Identification.
- The configuration includes the configuration global variable and access variable.

2.3.1. Resource

- The resource means one CPU module. Only one can be defined for G4 ~ G6. The resource has its own name. The resource can be named up to 8 digits according to clause 3.1.1. Identification.
- The resource includes the program, resource global variable and task definition.

2.3.1.1. Program

- Application program executing in PLC. In G Series PLC, several application programs can be prepared at one resource and the executing condition can be defined. For example, "A" can be defined as general scan program, "B" as the program executing by every second and "C" as the program executing by certain input. These executing conditions are called "Task". The user should prepare the application program and define executing condition(task). If the task is not defined, that program is regarded as a singal scan program.

Reference

Scan program: An Application program that execute the program from start to end after reading input data at the input module and, then, writes the result to output module. (it’s rotating)

- The program has instance name. The data to be handled in the program is stored in the instance.

Reference

Please, refer to clause 3.5.2 Function block for the instance.

2.3.1.2. Resource global variable

- The variable, defined at the resource global variable, can be used for any program in the resource. The data shared between programs are defined in the resource global variable.
- The variable type shall be declared as VAR_EXTERNAL in order to use the resource global variable in the program.

Reference

Please refer to clause 3.3.2 Variable declaration for the variable type.

2.3.1.3. Task

- The condition to execute the program is defined as task. The task definition is classified by the program execution condition and priority.
- The program execution condition is divided by 3 types.
  1) Single: Executed only once if the condition is satisfied. The condition is set by BOOL variable name.
  2) Interval: Executed periodically by the specified interval. The condition is selected by the interval time. Please refer to Clause 3.1.3.1 for the interval time.
  3) Interrupt: Executed once when the contact point of interrupt card is on. The selection is set by the contact point number.

Operation condition	Setting	Description
Single	%IX0.0.1	Executed once when input contact point, %IX0.0.1, is on.
Interval	T#1S	Executed every second.
Interrupt	4	Executed once when input contact point 4 is on.

- The priority is set from 0 to 7. Priority 0 is the highest one. The highest priority executes high task during scheduling and, if the priority is same, it is executed in the order of the generation of condition.
- _ERR_SYS, _H_INIT and _INIT tasks are the task which is subscribed by the system.

  _ERR_SYS:Systemerror task (supportedonlyatGM1,2)

  _H_INIT: Hot restart task

  _INIT: Cold/Warm restart task

2.3.2. Configuration global variable

- The variable, defined at the configuration global variable, can be used for any program in the resource. The data shared between resources are defined in the configuration global variable.
- The variable type shall be declared as VAR_EXTERNAL in order to use the configuration global variable in the program.

Reference

Please refer to clause 3.3.2 Variable declaration for the variable type.

- The configuration global variable can be defined at GM1 when several resources exist.

2.3.3. Access variable

The variable, defined at the access variable, can be used for other PLC system.

Reference

Please refer to the user's manual(Communication section) for the access variable.

Chapter 3 Common element

3. Common element

Program configuration element of G series PLC(program block, function, and function block) can be prepared by different languages such as IL, LD, SFC and etc. But, these languages contain common structure elements.

3.1. Expression

3.1.1. Identifiers

- The combination of English alphabet or all literal starting with underline character(_), number, underline character can be the identifier.
- Identifier is used for the variable name.
- Identifier shall not include the space.
- Identifier is 16 characters of English literal in case of normal variable and 8 characters of English literal in case of I/O variable and instance name.
- All English literal are acknowledged as the capital letter.

Feature description	Examples
Upper case and numbers	IW210, IW215Z, QX75, IDENT
Upper case, numbers, embedded underlines	LIM_SW_2, LIMSW5, ABCD, AB_CD
Upper case, numbers, leading or embedded underlines	_MAIN, _12V7, _ABCD

3.1.2. Data expression

Numeric Literal, Character String, Time Literal and etc. are used for the data in G SERIES PLC.

Feature description	Examples
Integer literals	-12, 0, 123_456, +986
Real literals	-12.0, 0.0, 0.456, 3.14159_26
Real literals with exponents	-1.34E-12, 1.0E+6, 1.234E6
Base 2 literals	2#1111_1111, 2#1110_0000
Base 8 literals	8#377(Decimal 255) 8#340(Decimal 224)
Base 16 literals	16#FF(Decimal 255) 16#E0(Decimal 224)
Boolean zero (false) and one (true)	0, 1, TRUE, FALSE

3.1.2.1. Numeric literals

- There are two classes of numeric literals : integer literals and real literals.
- Single underline characters(_) inserted between the digits of a numeric literal shall not be significant.
- Decimal is expressed by general decimal number and, if it has decimal point, it is regarded as real literals.
- Symbol of + and - can be used for the exponent. 'E' identifying the exponent does not have any difference between capital letter and small letter.
- Real literals with exponent shall be described as below.

  Ex) 12E-5 (x) 12.0E-5 (.)
- 2,8,10, 16 can be used for the integer and binary # is marked before the numeric literals. If binary # is not marked, it is regarded as decimal.
- 0 - 9, A - F are used for hexadecimal and a - f can be used also.
- Symbol (+,-) shall not be used for the binary expression.
- Boolean Data can be expressed by integer 0 and 1.

3.1.2.2. Character string literals

- All characters in the single quote character(') are the character string literals.
- The length is restricted within 16 characters for character string constant and 30 characters for initialization.

Example

'CONVEYER'

3.1.2.3. Time literals

- Time literals is classified by Duration data for measuring or controlling the elapsed time of a control event time and Time of day and date data for synchronizing the beginning or end of a control event to an absolute time reference.

3.1.2.3.1. Duration

- Duration data shall be delimited on the lelf, by the keyword T# or t#.
- The data shall be described in order of days(d), hours(h), minutes(m), seconds(s) and milli-seconds(ms) and can be start from any unit and ms do not need to be used but intermediate unit can not be omitted.
- Underline literal(_) is not used.
- The overflow is allowed at maximum unit and the unit can be described down to the decimal point except ms. However, the maximum can not exceed T#49d17h2m47s295ms. (32 bits of ms unit)
- The place of decimal point is restricted to three points at the second unit(s).
- The decimal point can not be used at ms unit.
- Capital · small letter are available for the unit letter.

Feature description	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

- The date and time are expressed by three types of date, hour and date/hour as below.

Feature description	Prefix keyword
Date prefix	D#
Time of day prefix	TOD#
Date and time prefix	DT#

- The date start with Jan. 1, 1984.
- The expression of hour and date/hour is restricted and ms unit is available for three places down to decimal point. (1ms unit)
- The overflow is not allowed at all units for hour and date/hour expression.

Feature description	Examples
Date literals	D#1984-06-25 d#1984-06-25
Time of day literals	TOD#15:36:55.36 tod#15:36:55.369
Date and time literals	DT#1984-06-25-15:36:55.36 dt#1984-06-25-15:36:55.369

3.2. Data types

Data has the data type expressing unique property.

3.2.1. Elementary data types

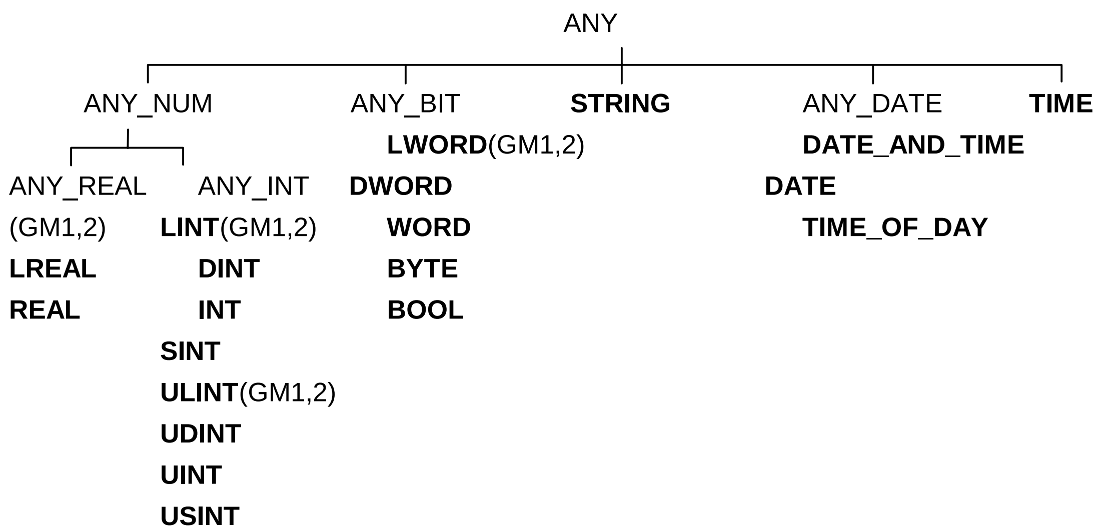
G SERIES PLC supports the basic data type as below.

No.	Keyword	Data type	Size (Bit)	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} ~ 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 ~ 2^{64}-1
9	REAL	Real numbers	32	-3.402823E38 ~ -1.401298E-45 1.401298E-45 ~ 3.402823E38
10	LREAL	Long reals	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 of day	64	DT#1984-01-01-00:00:00 ~ DT#2163-12-31-23:59:59.999
15	STRING	Character string	30*8	-
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

Note LINT, ULINT, REAL, LREAL and LWORD is **NOT** supported in the G6 and G4

3.2.2. Data type hierarchy

Below data type is used in G SERIES PLC.



- ANY_REAL(LREAL, REAL), LINT, ULINT and LWORD are applied in GM1 and GM2.
- If ANY_NUM is displayed in the data type, LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT and USINT are included as the hierarchy.
- For example, if the type is expressed as ANY_BIT in GM3, one of DWORD, WORD, BYTE and BOOL can be used.

3.2.3. Initial value

If initial value of data is not assigned, the data will be assigned automatically as below.

Data type(s)	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)

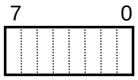
3.2.4. Data type structure

Bit String

BOOL

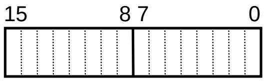
1 Bit, Range: 0 , 1

BYTE



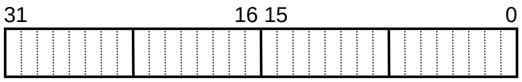
8 Bit, Range: 2#0000_0000 ~ 2#1111_1111, 16#00 ~ 16#FF

WORD



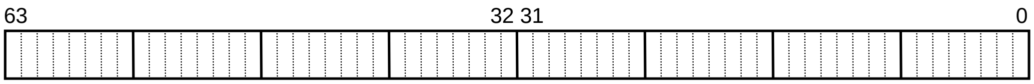
16 Bit, Range: 2#0000_0000 _0000_0000 ~ 2#1111_1111_1111_1111
16#0000 ~ 16#FFFF

DWORD



32 Bit, Range: 2#0000_...000 ~ 2#1111_...111
16#00000000 ~ 16#FFFFFFFF

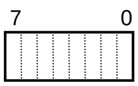
LWORD



64 Bit, Range: 2#0000_...000 ~ 2#1111_...111, 16#0000000000000000 ~ 16#FFFFFFFFFFFFFFFF

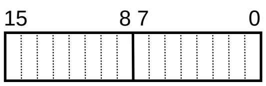
Unsigned Integer

USINT



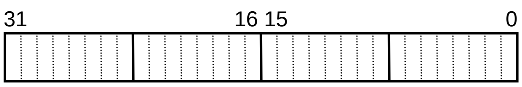
8 Bit, Range: 0 ~ 255

UINT



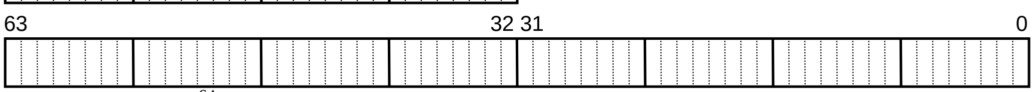
16 Bit, Range: 0 ~ 65,535

UDINT



32 Bit, Range: 0 ~ 4,294,967,295

ULINT



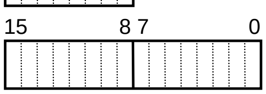
64 Bit, Range: 0 ~ 2^{64}-1

Integer (Negative is expressed by 2' Complement)

SINT

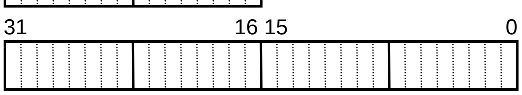
8 Bit, Range: -128 ~ 127

INT



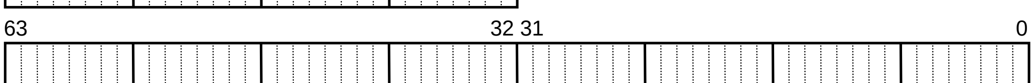
16 Bit, Range: -32,768 ~ 32,767

DINT



32 Bit, Range: -2,147,483,648 ~ 2,147,483,647

LINT



64 Bit, Range: -2^{63} ~ 2^{63}-1

Real (Based on IEEE Standard 754-1984)

REAL

31	30 … 23	22 … 0
S	Exponent	Fraction

32 Bit, Range: ±1.401298E-45 ~ ±3.402823E38

LREAL

63	62 … 52	51 … 0
S	Exponent	Fraction

64 Bit, Range: ±4.9406564E-324 ~ ±1.7976931E308

- S: symbol mark (0: Positive, 1: Negative)
- Exponent: 2' exponent (2^{e-127}: $e=b_{30}b_{29}...b_{23}$, $e=b_{62}b_{61}...b_{52}$)
- Fraction: value down to the decimal point (Fraction: $f=b_{22}b_{21}...b_0$, $e=b_{51}b_{52}...b_0$)

Time

TIME (bits 31 … 0)

32 Bit, Range: 0 ~ 4,294,967,295ms

T#49d17h2m47s295ms

Date

DT

63 … 48	47 … 32	31 … 0
0000000000000000	DATE	TOD

64 Bit, Range: DT#1984-01-01-00:00:00 ~ DT#2163-12-31-23:59:59.999

DATE (bits 15 … 0)

16 Bit, Range: D#1984-01-01 ~ D#2163-6-6

TOD (bits 31 … 0)

32 Bit, Range: TOD#00:00:00~TOD#23:59:59.999

#BCD

(BYTE)

7 … 4	3 … 0
10^1	10^0

8 Bit, Range: 0 ~ 99

(WORD)

15		8 7	0
10^3	10^2	10^1	10^0

16 Bit, Range: 0 ~ 9999

(DWORD)

31		24 23		16 15		8 7	0
10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0

32 Bit, Range: 0 ~ 99,999,999

(LWORD)

63				48 47				32 31				16 15			0
10^{15}	10^{14}	10^{13}	10^{12}	10^{11}	10^{10}	10^9	10^8	10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0

64 Bit, Range: 0 ~ 9,999,999,999,999,999

3.3. Variable

The variable includes the data value in the program. The variable indicates PLC I/O, internal memory and etc.

3.3.1. Representation

- The variable is expressed by two types; one is by assigning name with identifier and the other is by expressing PLC I/O or physical or logical place in the memory in the data element directly.
- The variable by identifier shall be unique in the scope(program configuration region declared as variable) to identify other variable.
- Direct variable can be expressed by the order of prefix starting with % letter and data. The prefix is described as below.

Location prefix

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

Size prefix

No.	Prefix	Meaning
1	X	Single bit size
2	None	Single bit size
3	B	Byte(8 Bits) size
4	W	Word(16 bits) size
5	D	Double Word(32 Bits) size
6	L	Long Word(64 Bits) size

Expression type

%[Location prefix][Size prefix] n1.n2.n3

No.	I, Q	M
n1	Base number(start from 0)	n1 data according to [Size prefix] (start from 0)
n2	Slot number(start from 0)	n2 bit on n1 data (start from 0) : can be omitted.
n3	n3 data according to [Size prefix] (start from 0)	Not used.

Ex)

%QX3.1.4 or %Q3.1.4	No. 4 output (1 Bit) of No. 1 slot of No. 3 base
%IW2.4.1	No.1 input by the word unit of No. 4 slot of No. 2 base (16 Bits)
%MD48	Memory of double word unit located at 48
%MW40.3	No.3 bit in memory on word unit located at 40 (There is no concept of base, slot and etc. internal memory.)

- Small letter can not be used for the prefix.
- If size prefix is not used, the variable is processed as 1 bit.
- Direct variable can be used without declaration.

3.3.2. Variable declaration

- Program configuration element(i.e. programn block, function, function block) has the declaration, which can declare the variable in configuration element.
- The variable shall be declared in order to use it in the program configuration element.
- Bellows shall be set for the variable declaration.

1) Variable type : set how to declare the variable.

Variable type	Description
VAR	General variable readable and writable
VAR_RETAIN	Retentive variable
VAR_CONSTANT	Variable readable only
VAR_EXTERNAL	Declaration to use VAR_GLOBAL variable

Reference

Resource global variable and configuration global variable are declared by VAR_GLOBAL, VAR_GLOBAL_RETAIN, VAR_GLOBAL_CONSTANT and VAR_EXTERNAL can not be declared.

2) Data type : Assign the data type of variable.
3) Memory allocation : Allocate the memory for variable.

Automatic	----	The compiler assigns the variable location automatically (Automatic allocation variable).
User definition(AT)	----	The user assigns the location by direct variable. (Direct variable)

Reference

The location of automatic allocation variable is not fixed. For example, if VAL1 variable is declared as BOOL data type, the variable is not fixed in a specific location in the data region. The compiler and linker define the location after programming. In case of compiling again after correcting the program, the location can be changed.

The benefit of automatic allocation of a variable is that the user does not need to check the location of the internal variable. A variable declared by a different name is not duplicated in the data memory.

If the variable location is set for direct variable, the memory is allocated implicitly by %I, %Q and %M. It is possible to duplicate the memory allocated so care must be taken

. Initial Value allocation : Allocate the initial value of variable. If no, the value of clause 3.2.3. is allocated as initial value.

Reference

The initial value can not be allocated if VAR_EXTERNAL is declared.

The initial value can not be allocated if %I %Q, and %Mare allocated.

. After PLC is switched off, the variable, which requires the data, can be declared by VAR_RETAIN supplying retention function under below regulation.

1) Retention variable is retained at the warm restart.
2) During cold restart of system, it is initialized to initial value defined by user or basic initial value.

. Variable not declared by VAR_RETAIN is initialized to initial value defined by user or basic initial value during cold restart or warm restart.

Reference

The variable allocated to %I,%Q and %M can not be declared to VAR_RETAIN, VAR_CONSTANT.

. The variable can be used by declaring basic data type to array. To declare array variable, the type and array size to be used for parameter shall be declared.
However, string data type in basic data type can not be set to parameter.

. Scope of variable declaration, i.e., region to use variable, is restricted to the program configuration element that the variable is declared. Therefore, the variable declared in other program configuration element can not be used. The variable declared as global variable can be accessed in all location by declaring VAR_EXTERNAL.
The configuration global variable can be used all program configuration element of all resources and the resource global variable can be used in all program configuration element.

Example of variable declaration

Name	Type	Data type	Initial value	Memory allocation
I_VAL	VAR	INT	1234	Automatic
BIPOLAR	VAR_RETAIN	REAL		Automatic
LIMIT_SW	VAR	BOOL		%IX1.0.2
GLO_SW	VAR_EXTERNAL	DWORD		Automatic
READ_BUF	VAR	ARRAY OF INT[10]		Automatic

3.3.3. Reserve variable

- The reserved variable is declared in the system in advance. These variables are used for special purpose and the user can not declare them as variable name.
- Use the reserved variable without declaration.
- Refer to 'CPU user's manual' for details.

1) User flag

Reserve variable	Data type	Description
_ERR	BOOL	Operation error contact
_LER	BOOL	Operation error latch contact
_T20MS	BOOL	20 ms Clock contact
_T100MS	BOOL	100 ms Clock contact
_T200MS	BOOL	200 ms 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

Reserve variable	Data type	Content
_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
_FUSE_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

Reserve variable	Data type	Description
_CNF_ER_M	BYTE	System error(heavy trouble) release
_IO_DEER_M	BOOL	Module installation error release
_FUSE_ER_M	BOOL	Fuse shortage error release
_IO_RWER_M	BOOL	I/O module read/write error release
_SP_IFER_M	BOOL	Special/Communication module interface error release
_ANNUN_ER_M	BOOL	Heavy trouble detection error release of external device

4) System alarm flag

Reserve variable	Data type	Description
_CNF_WAR	WORD	System alarm(Alarm message)
_RTC_ERR	BOOL	RTC data error
_D_BCK_ER	BOOL	Data back-up 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

Reserve variable	Data type	Description
_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

5) System error details flag

Reserve 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
_IP_IFER_N	UINT	Special/Link module interface error slot number
_IP_IFERR	ARRAY OF BYTE	Special/Link module interface 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 BIT	Light trouble detection bit map of external device
_TC_BMAP	ARRAY OF BYTE	Task conflict mark
_TC_CNT	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 power failure history

6) System operation status

Reserve variable	Data type	Description
_CPU_TYPE	UINT	System type
_VER_NUM	UINT	PLC O/S Ver. No.
_MEM_TYPE	UINT	Memory module type
_SYS_STATE	WORD	PLC mode and status
_GMWIN_CNF	BYTE	PADT connection 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 corresponds to the slot number which the communication module is installed(n = 0 - 7)]

Reserve variable	Data type	Description
_CnVERNO	UINT	Communication module Ver. No.
_CnSTNOH _CnSTNOL	UINT	Communication module station 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	Min. 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)

8) Remote I/O control flag

[m correspond to the slot number which the communication module is installed(m = 0 - 7)].

Reserve variable	Data type	Description
_FSMm_reset	BOOL(Writable)	Remote I/O station reset control(reset=1)
_FSMm_io_reset	BOOL(Writable)	Output reset control of remote I/O station (reset=1)
_FSMm_st_no	USINT(Writable)	Station number of corresponding remote I/O station

9) HS(High-speed) link information detail flag

[m corresponds to the high-speed link parameter number(m = 1,2,3,4)]

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

3.4. Keywords

The keywords are the words defined in advance for the system. Therefore, the identifier can not be used for the keywords.

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

3.5. Program type

- The program is divided by function, function block and program.
- The program can not call its program itself.(Recursive calling prohibit)

3.5.1. Functions

- Function has one output.

Example

If A function adds input IN1 and IN2 and then adds the result with 100, i.e., output 1 <= IN1 + IN2 + 100, this function is correct. But, if output 2 <= IN1 + IN2 * 100 at another output, this can not be the function since the output is two, output 1 and output 2.

- Functions shall contain no internal state information, i.e., invocation of a function with the same arguments (input parameters) shall always yield the same value (output).

Example

If B function is

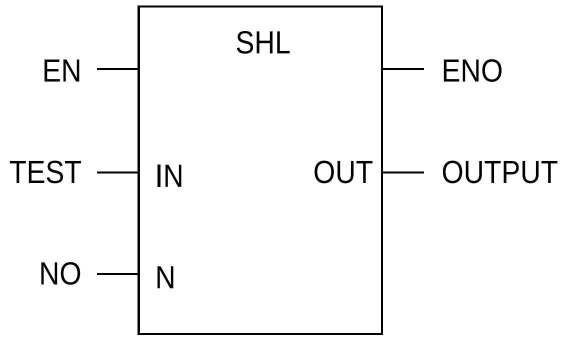
Output 1 <= IN1 + IN2 + Val

Val <= Output 1 (Val is internal variable),

This can not be function since internal variable Val exists. Internal variable causes different output though the input is same. In above sample, output 1 can be changed due to Val variable though IN1 and IN2 are fixed. Comparing A function, if IN1 is 20 and IN2 is 30 in A function, output 1 is always 150. If the input is fixed, the output is also fixed.

- Internal variable of function can not get initial value.
- The variable can not be declared as VAR_EXTERNAL.
- Direct variable shall not be used in the function.
- The function is called from the program configuration element.
- In the configuration element of program calling function, the data transfer from function is executed through the input.

Example



SHL function is elementary function to output the result after left-shifting data if IN input to N times. The program configuration element calling SHL function assigns and calls TEST variable to IN input and N input to NO variable. The function result is stored to OUTPUT variable.

- The function is used in the library.
- The function block or program can not be called in the function.
- The function is same to the function name and has the variable that data type is same to the function result. This variable is automatically generated during creating the function and outputs the result in this variable.

Example

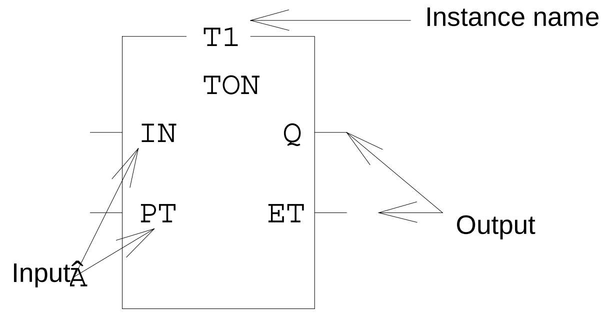
If the function name is WEIGH and the result is WORD data type, internal function name becomes WEIGH and variable of WORD data type is automatically generated. The user inputs the result to WEIGH variable and outputs it.

ST WEIGH (Example at IL)

3.5.2. Function blocks

- The function block output can be several.
- The function block can have internal data. The function block shall declare the instance as if the variable is declared before using. The instance is the assembly of variable used in function block. The function block, as instance, shall have data memory since it stores the variable and output values. The program can be kind of function block and shall declare the instance. But, the program can not be used in the program or function block.
- The output of function block puts the period(.) behind the instance name and then, write output name.

Example



General type of function block is the timer and counter. As on-delay timer function block is TON, declaring T1 and T2 as instance and calling them in the program operates on-delay time. The timer's output contact or elapsed time is used by putting the period(.) between instance name and output name. For the timer function block, as the output contact name is Q and the output contact of each instance is T1.Q, T2.Q and elapsed time name is ET, T1.ET and T2.ET describe elapsed time. Comparing the function output, the output is the return value calling function and the function block output is defined at each instance.

- Direct variable can not be used in the function block. But the direct variable declared as global variable and allocated to user definition(AT) can be used as declaration of VAR_EXTERNAL.
- Function block is used by the library.
- The program can not be called in the function block.

3.5.3. Program blocks

- The program is declared as instance like function block.
- Direct variable can be used in the program.
- Input/Output variable is not used in the program.

The program call is defined in the resource.

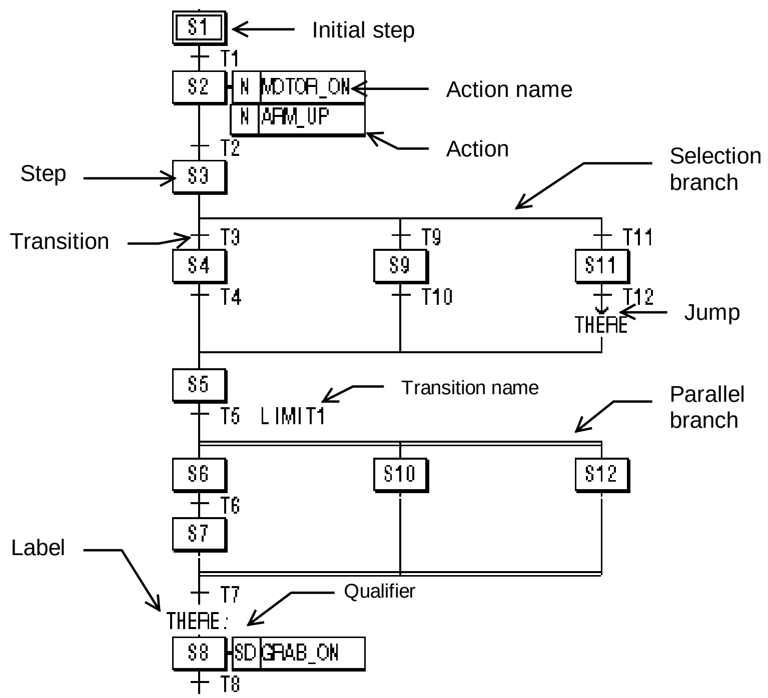
MEMO

Chapter 4 SFC (Sequential Function Chart)

4. SFC(Sequential Function Chart)

4.1 Overview

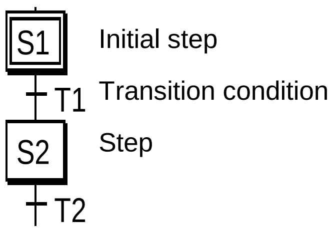
- SFC is a graphical language for depicting sequential behaviur of a control system.It is used for defining control sequences that are time and event driven.It is not a full language as it requires instructions taken from other languages.
- SFC. Provides the means of dividing the program into a set of steps and transitions interconnected by direct links. Associated with each step is a set of actions. The actions and transition are achived by the use of any of the other languages i.e Ladder, Instruction List
- Type



4.2. SFC structure

4.2.1. Steps

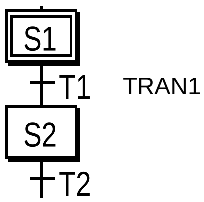
- Step indicates the unit of the sequence control by connecting the action.
- If the step is active, the attached action is executed.
- Initial step is initially active step.



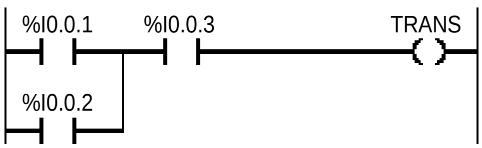
- If the transition condition is established after the initial step, current active step(S1) is changed to the inactive status and next step(S2) is active.

4.2.2 Transitions

- The transition indicates the execution process transition condition between steps.
- The transition condition shall be prepared by IL or LD of PLC language.
- If the result of transition is 1, current step will be inactive and next step will be active.
- The transition shall be arranged between steps.



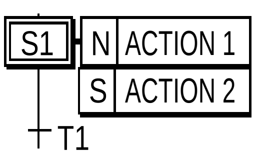
Description of TRAN1



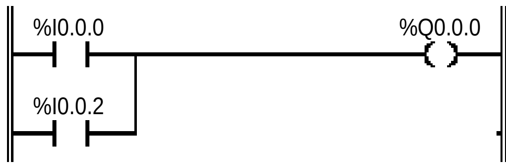
If TRANS is on, S1 will be inactive and S2 will be active.
TRANS variable is the declared one internally.
The transition condition shall be output by TRANS variable at all transition.

4.2.3. Actions

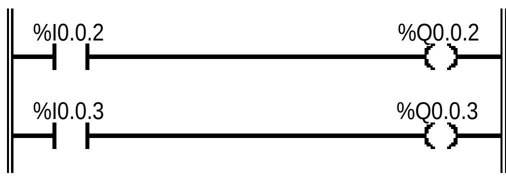
- Two actions can be connected to each step.
- The step without action is regarded as the standby untill next transition condition is 1.
- The action consists of IL or LD of PLC language and the action is executed while the step is active.
- The gualifier is used to control the action.
- When the action is changed from active to inactive status, the contact output executed at the action will be 0. However, S, R, function, function block output remains the status before inactive status.



Content of ACTION1



Content of ACTION2



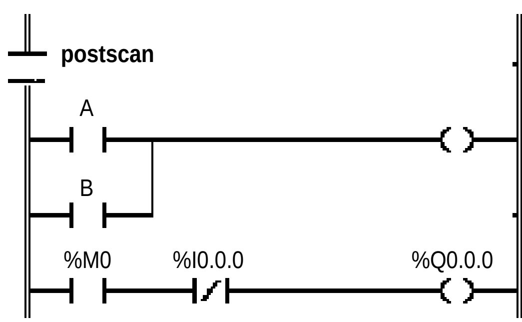
- ACTION1 is executed only when S1 is active.
- ACTION2 is executed till meets R qualifier after S1 is active. It will be continuously executed though S1 is inactive.
- At the moment of disabling of action, this action is post scanned and switched over next step.

Reference

Post scan

The action is scanned again when the action becomes inactive. The program output of contact program will be 0 because it is scanned regarding certain contact point(value = 0) is existed at the initial of action program.

The function, function block, S, R output and etc. are not included.



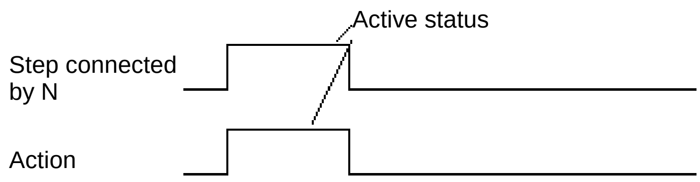
In above figure, C and %Q0.0.0 will be 0 since postscan's contact point is 0.

4.2.4. Action qualifiers

- Action qualifier is used whenever the action is used.
- The execution point/time of action relating to the step is defined according to the selected qualifier.
- The action qualifier is classified as follows;

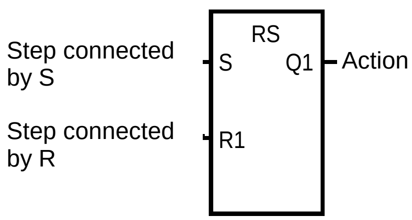
1) N(Non-Stored)

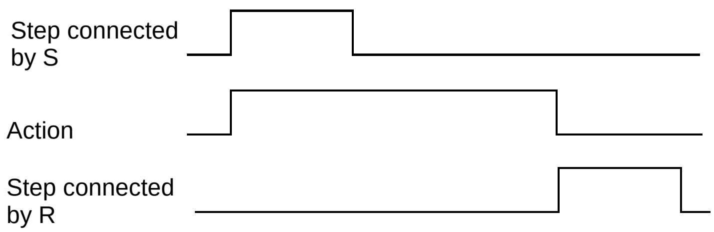
The action is executed as long as the step is active.



2) (S)Set

The action is executed, as soon as the step is active, till R qualifier is executed.



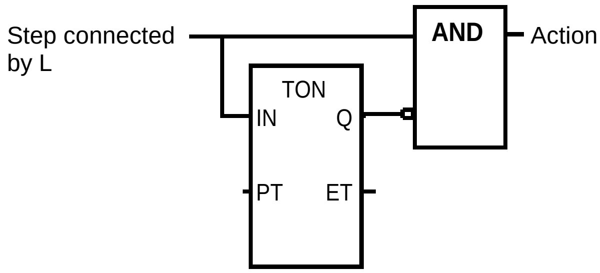


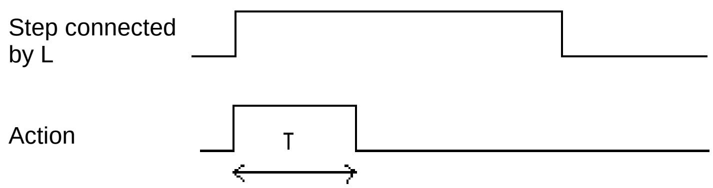
3) R(Overriding Reset)

The action executed by S, SD, DS and SL qualifier is aborted.

4) L(Time Limited)

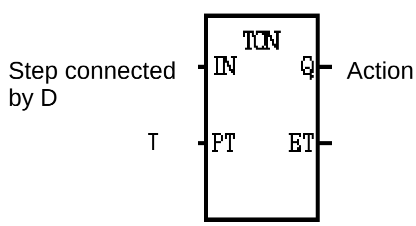
The action is executed for a preset length of time T, as long as the step is active.

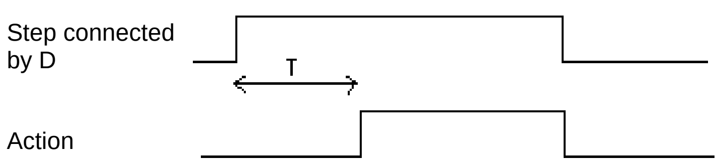




5) D(Time Delayed)

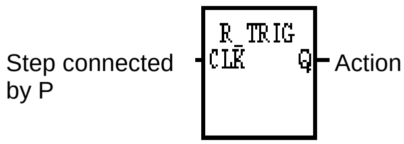
The action is executed after a preset time T has elapsed and remains executed for as long as the step is active.

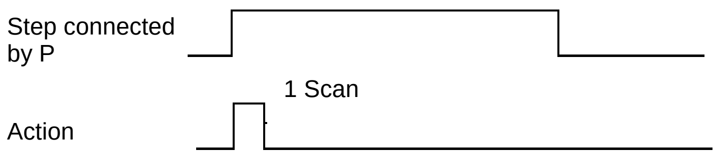




6) P(Pulse)

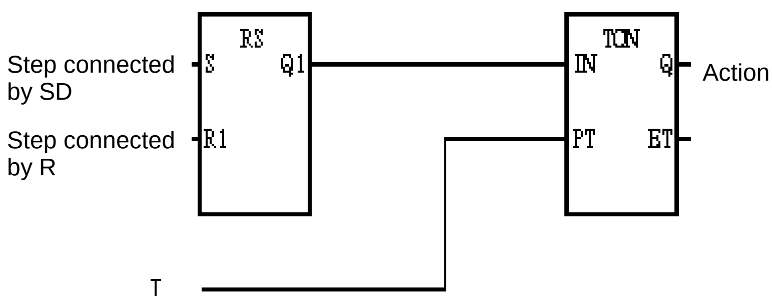
As soon as the step is active, the action is executed for one cycle.

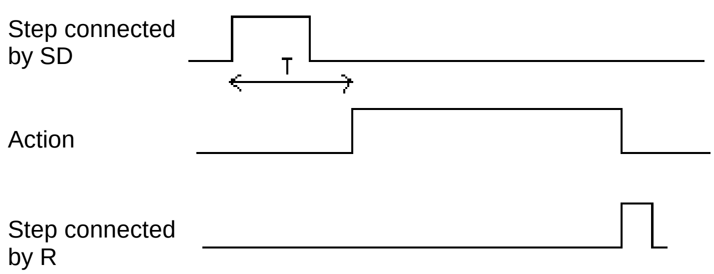




7) SD(Stored & Time Delayed)

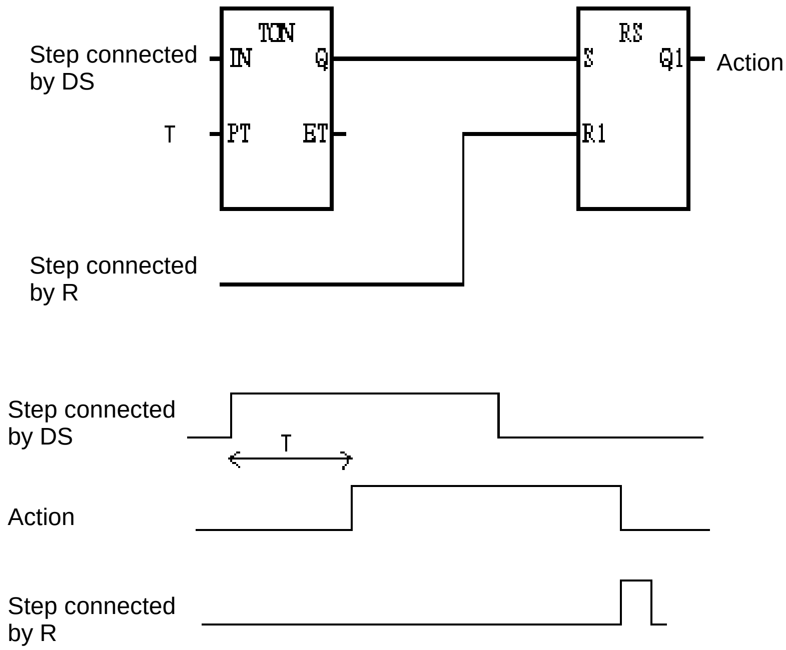
The action is executed when a preset period of time T has elapsed after step activation, even if the step becomes inactive. This condition persists until the R qualifier is executed.





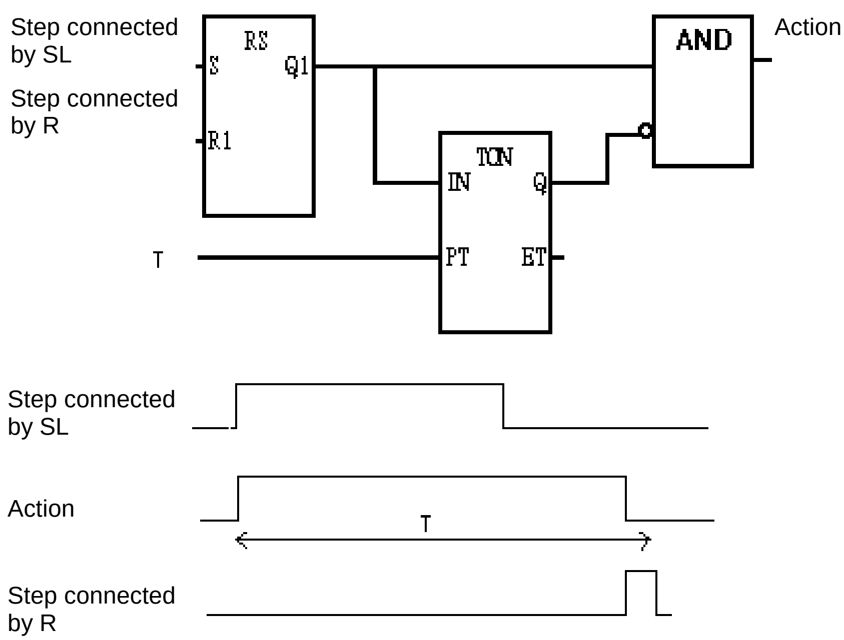
8) DS(Delayed & Stored)

As for the SD qualifier, the action is also executed with a time delay. It differs from the SP qualifier in that the step must remain active during the time delay.



9) SL(Stored & Time Limited)

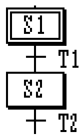
The action is executed for a preset length of time T as soon as the step is active until the R qualifier is executed.

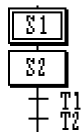


4.3. Rules of evolution

4.3.1. Serial connection

- Two steps are not connected directly and are always divided by the transition .
- Two transitions are not connected directly and are always divided by the step





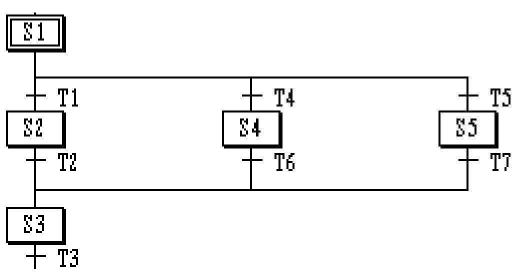
[Right example]   [Wrong example]

- Regarding the transition between steps of serial connection, lower step will be active when next transition condition becomes 1 under the activation of upper step.

4.3.2. Selection branch

- The step next to the transition condition of 1 will be active under the activation of upper step in case of selection branch. The others are same to the serial connection.

Example



* When the transition condition of T1 is 1,
  The selection branch will be active in order of S1 → S2 → S3.

* When the transition condition of T4 is 1,
  The selection branch will be active in order of S1 → S4 → S3.

* When the transition condition of T5 is 1,
  The selection branch will be active in order of S1 → S5 → S3.
  When the transition condition is 1 simultaneously, left transition will be prior to the right transition.

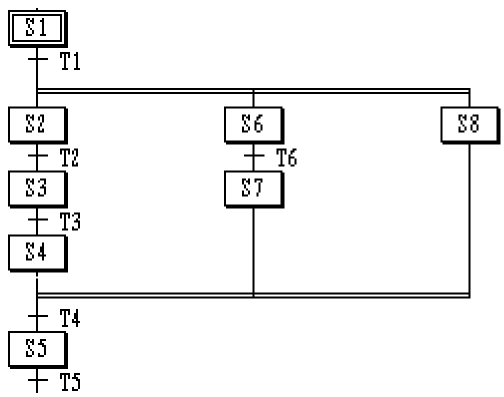
* When the transition condition of T1 and T4 is simultaneously,
  The selection branch will be active in order of S1 → S2 → S3.

* When the transition of T1 and T5 is 1 simultaneously,
  The selection branch will be active in order of S1 → S4 → S3.

4.3.3. Parallel branch

- All steps which are connected below this transition will be active if the transition condition of next transition is 1 under the activation of upper step in case of parallel branch. At this time, the active step will be as many as the number of branch.
- If parallel branches are joined, next step will be active when last step of each branch is active and transition condition is 1.

Example



* When S1 is active and transition condition T1 is 1, S2, S6 and S8 will be active and S1 will be inactive.

* When S4, S7 and S8 are active and transition condition T4 is 1, S5 will be active and S4, S7 and S8 will be inactive.

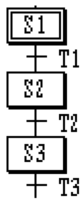
* Active order

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

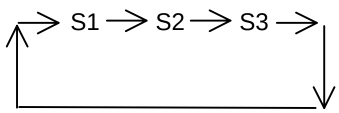
4.3.4. Jump

- SFC initial step will be active if next transition condition of the last step is 1 after SFC last step is active.

Example



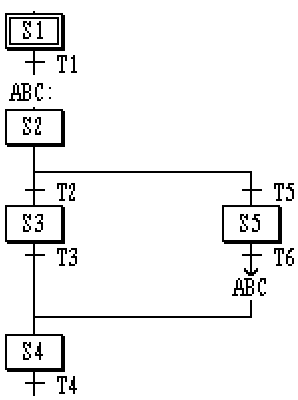
* Active order



- The flow can be controled to the aimed position using the jump.
- The jump can be extended only the end of SFC program or selection branch. The jump can not be done toward or outward parallel branch. But, the jump is possible within the parallel branch.

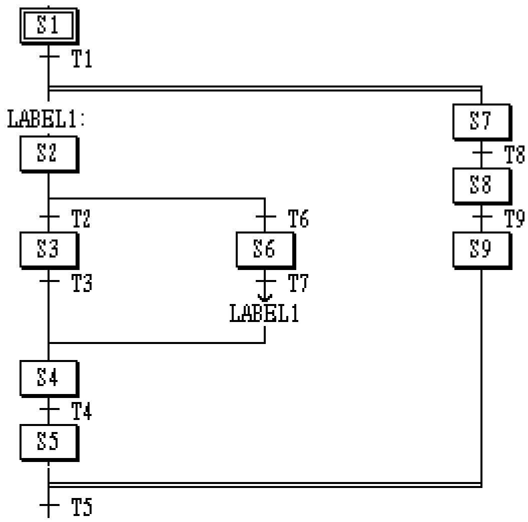
Example

1) Jump at the end of selection jump.

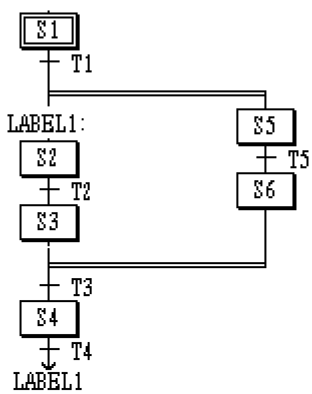


- S2 is active after S5.

2) Jump in parallel branch



3) The jump can not be done into parallel branch.



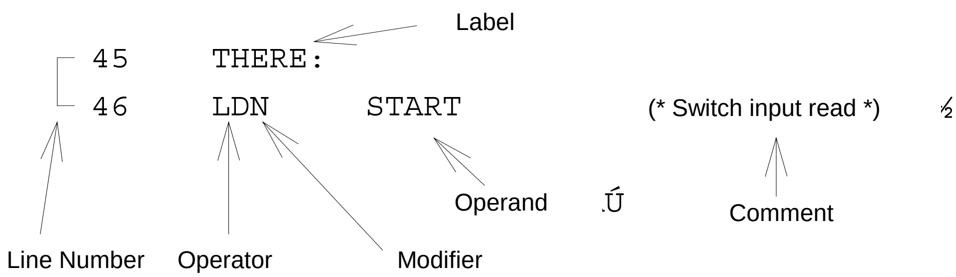
MEMO

Chapter 5 IL(Instruction List)

5. IL(Instruction List)

5.1 Overview

- IL is the type of assembly language.
- IL can be applied to simple PLC system.
- Type



5.2. Current Result(CR)

- Current Result(CR) is the registor that has the operation result at any time.
- Only one CR exists in IL.
- CR can be any data type.
- LD(Load) is the operator that defines the data type of CR by inputting any value in CR.

Example

Variable %IX0.0.0 value is input for LD %IX0.0.0 in CR. The data type of CR will be BOOL since the data type of variable expressed by X is BOOL. If VAL variable is declared as INT and used as LD VAL, the variable value of VAL will be input to CR and data type of CR will be INT.

- ST(Store) is the operator that stores CR to certain variable.

Example

If VAL variable is declared as INT and used as ST VAL, it means that CR is input to the variable of VAL. At this moment, CR shall be the data type of INT. If CR is other data type except INT, the error will occur during compiling.

When

LD %IX0.0.0

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

is set, as CR is defined as BOOL in first line and CR is used as INT, the error occurs during compiling.

LD %IX0.0.0

ST START

LD 20

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

As the data type is same in above example when selected CR is stored, this is normal program.

5.3. Instructions

- IL consists of sequential command.
- Each command consists of the operator, which contains the modifier, and operand.

5.3.1. Label

- Label is marked at the region of operator putting colon(:) behind the label name.
- Label is used for the destination of jump command.

5.3.2. Modifiers

- Modifier is put behind the operator and executed by modifying original operation function. The type of modifier is N, (and C.
- Modifier "N" indicates Boolean Negation of operand.

Example

ANDN %IX2.0.0 is translated as below.

CR <= CR AND NOT %IX2.0.0

When N is used for JMP, CAL or RET, it means that the command is executed if CR is BOOL 0.

- Modifier "(" is the operation of operator")" is delayed till it meets the operator.

Operates other operation since one CR exists and IL stores CR for a short time and the result and the delay operation which stored CR value can be executed.

Type	Item	Description
(	Modifier	Indicate the delay of operation.
)	Operator	Executes delayed operation.

Example

AND(%IX1.0.0

OR %IX2.0.0

)

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

Therefore, AND execution is delayed till) appears. The operation %IX1.0.0 OR %IX2.0.0 in parentheses is first executed and the result is operated lately.

- Modifier " C " means that the selected command is executed only when current operated CR is BOOL 1.

Example

JMPC THERE

If CR is BOOL1, jump to THERE

5.3.3. Operators

- Basic operator is described as below.

No.	Operator	Modifiers	Operand	Description
1	LD	N	Data	Operand is input to Current Result(CR).
2	ST	N	Data	Current Result(CR) is stored at the operand.
3	S		BOOL	If CR is BOOL1, the operand will be 1.
	R		BOOL	If CR is BOOL1, the operand will be 0.
4	AND	N,(	Data	Logic AND operation
5	OR	N,(	Data	Logic OR operation
6	XOR	N,(	Data	Logic XOR operation
7	ADD	(	Data	Arithmetic + operation
8	SUB	(	Data	Arithmetic - operation
9	MUL	(	Data	Arithmetic * operation
10	DIV	(	Data	Arithmetic / operation
11	GT	(	Data	Comparison operation: >(greater than)
12	GE	(	Data	Comparison operation: >=(greater than or equal)
13	EQ	(	Data	Comparison operation: =(equal)
14	NE	(	Data	Comparison operation: <>(not equal)
15	LE	(	Data	Comparison operation: <=(less than or equal)
16	LT	(	Data	Comparison operation: < (less than)
17	JMP	C,N	Label	Jump to label
18	CAL	C,N	Name	Call function block
19	RET	C,N		Return at the function or function block
20	)			Executes delayed operation with '(' modifier.

- The operator from 4 to 16 executes below function.

CR **<=** ***CR*** **operand**

CR and operand's values are operated and, then, the result is stored to CR again.

Example

AND %IX1.0.0 is translated as below.

CR <= CR AND %IX1.0.0

- The comparison operator operates left CR and right operand and stores the BOOL result in CR.

Example

For GT %MW10, CR will be BOOL 1 if CR is larger than memory word 10's value and will be 0 if not.

- Most of operation command does not change CR data type after operation. But, the data type of CR is changed for the comparison command.

Example

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

(Assuïe that START variable is declared as BOOL and VAL and CR variables are declared as INT.)

Input VAL of INT value to CR in (a) line. Input BOOL 1 value in CR if CR and INT values called GROSS are same in (b) line and input BOOL 0 in CR if not. The data type of CR will be changed from INT to BOOL. Therefore, the compile error does not occur normally in case of using command in (c) and (d) lines.

5.3.3.1. Details of operator

(1) LD

Description	Input operand to CR. At this moment, CR's data type is changed to the operand's one.	
Modifier	N: The operand value is reversed and input to CR if the operand is BOOL.	
Operand	All data type is available. The constant is available.	
Example	LD TRUE	Input BOOL 1 value in CR. (BOOL data type)
	LD INT_VALUE	Input INT_VALUE as INT variable in CR. (INT data type)
	LD T#1S	Input T#1S as duration constant in CR. (TIME data type)
	LDN B_VALUE	B_VALUE as BOOL variable is reversed and be put in CR. (BOOL data type)

(2) ST

Description	Input CR to operand At this moment, CR's data type shall be same to the operand's one. CR value is not changed.	
Modifier	N: In case the CR data type is BOOL, input CR to certain variable value in the operand after reversing. CR value is not changed.	
Operand	All data type is available. The constant shall not input. Shall be same to CR's data type.	
Example	LD FALSE	Input BOOL 0 to CR. CR's data type is BOOL.
	ST B_VALUE1	Input CR value 0 to B_VALUE1 variable of BOOL data type.
	STN B_VALUE2	Convert CR value(1) and input it to B_VALUE2 variable of BOOL data type.
	LD INT_VALUE	Input INT_VALUE of INT variable to CR. CR's data type is INT.
	ST I_VALUE1	Input CR value to I_VALUE1 variable of INT data type.
	LD D#1995-12-25	Input D#1995-12-25 of date constant to CR. CR's data type is DATE.
	ST D_VALUE1	Input CR value to D-VALUE1 variable of DATE data type.

(3) S(Set)

Description	If CR value is BOOL1, operand value of BOOL data type will be 1. If CR value is BOOL0, no operation is executed. CR value is not changed.	
Modifier	None	
Operand	BOOL data type is available only. The constant shall not input.	
Example	LD FALSE	Input BOOL 0 to CR. CR's data type is BOOL.
	S B_VALUE1	No operation is executed since CR is 0. B_VALUE1 variable will not be changed.
	LD TRUE	Input BOOL1 value to CR. CR's data type is BOOL.
	S B_VALUE2	B_VALUE2 variable of BOOL data type will be 1 since CR is 1.

(4) R(Reset)

Description	If CR value is BOOL1, operand value of BOOL data type will be 0. If CR value is BOOL0, no operation is executed. CR value is not changed.	
Modifier	None	
Operand	BOOL data type is available only. The constant shall not input.	
Example	LD FALSE	Input BOOL 0 to CR. CR's data type is BOOL.
	R B_VALUE1	No operation is executed since CR is 0. B_VALUE1 variable will not be changed.
	LD TRUE	Input BOOL1 value to CR. CR's data type is BOOL.
	R B_VALUE2	Because CR value is 1, B_VALUE 2 variable of BOOL data type will be 0.
	ST B_VALUE3	Input CR 1 to B_VALUE3 variable of BOOL data type.

(5) AND

Description	Executes logic AND operation of CR value and operand value and input the result to CR. The CR's data type shall be same to the operand's one. The operand value is not changed.		
Modifier	N: If the operand is BOOL, reverse the operand value and calculate with CR value. (: If operand's data type is BOOL. Current CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)		
Operand	BOOL, BYTE, WORD, DWORD and LWORD data types are available. The constant is available also.		
Example	LD	B_VALUE1	Input B_VALUE1 of BOOL data type to CR.
	AND	B_VALUE2	Execute AND operation of CR and B_VALUE2 of BOOL data type and input the result to CR.
	ANDN	B_VALUE3	Convert CR and B_VALUE3 of BOOL data type and execute AND operation and input the result to CR.
	ST	B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <= B_VALUE1 AND B_VALUE2 AND NOT(B_VALUE3)
	LD	W_VALUE1	Input W_VALUE1 of WORD variable to CR. CR's data type is WORD.
	AND	W_VALUE2	Execute AND operation of CR and W_VALUE2 of WORD data type and input the result to CR.
	ST	W_VALUE3	Input CR to W_VALUE3 of WORD data type. W_VALUE3 <== W_VALUE1 AND W_VALUE2
	LD	B_VALUE1	Input B_VALUE1 of BOOL data type to CR. CR's data type is BOOL.
	AND(	B_VALUE2	Store CR value to other position and input B_VALUE2 of BOOL data type to CR.
	OR	B_VALUE3	Execute OR operation of CR value and B_VALUE3 value of BOOL data type and input the result to CR.
	)		Execute AND operation of current CR and other CR values and input the result to CR.
	ST	B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <== B_VALUE1 AND (B_VALUE2 OR B_VALUE3)

(6) OR

Description	Executes logic OR operation of CR value and operand value and input the result to CR. The CR's data type shall be same to the operand's one. The operand value is not changed.	
Modifier	N: Reverse the operand value and calculate with CR value if the operand is BOOL data type. (: If operand's data type is BOOL. Current CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)	
Operand	BOOL, BYTE, WORD, DWORD and LWORD data types are available. The constant is available also.	
Example	LD B_VALUE1	Input B_VALUE1 of BOOL data type to CR. CR's data type is BOOL.
	OR B_VALUE2	Execute OR operation of CR and B_VALUE2 of BOOL data type and input the result to CR.
	ORN B_VALUE3	Convert CR and B_VALUE3 of BOOL data type and execute OR operation and input the result to CR.
	ST B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <= B_VALUE1 OR B_VALUE2 OR NOT(B_VALUE3)
	LD W_VALUE1	Input W_VALUE1 of WORD variable to CR. CR's data type is WORD.
	OR W_VALUE2	Execute OR operation of CR and W_VALUE2 of WORD data type and input the result to CR.
	ST W_VALUE3	Input CR to W_VALUE3 of WORD data type. W_VALUE3 <== W_VALUE1 AND W_VALUE2
	LD B_VALUE1	Input B_VALUE1 of BOOL data type to CR. CR's data type is BOOL.
	OR(B_VALUE2	Store CR value to other position and input B_VALUE2 of BOOL data type to CR.
	AND B_VALUE3	Execute AND operation of CR value and B_VALUE3 value of BOOL data type and input the result to CR.
	)	Execute OR operation of current CR and other CR values and input the result to CR.
	ST B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <== B_VALUE1 OR (B_VALUE2 AND B_VALUE3)

(7) XOR

Description	Executes logic XOR operation of CR value and operand value and input the result to CR. The CR's data type shall be same to the operand's one. The operand value is not changed.	
Modifier	N: In case the operand is BOOL data type, reverse the operand value and calculate with CR value. (: If operand's data type is BOOL. Current CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)	
Operand	BOOL, BYTE, WORD, DWORD and LWORD data types are available. The constant is available also.	
Example	LD B_VALUE1	Input B_VALUE1 of BOOL data type to CR. CR's data type is BOOL.
	XOR B_VALUE2	Execute XOR operation of CR and B_VALUE2 of BOOL data type and input the result to CR.
	XORN B_VALUE3	Convert CR and B_VALUE3 of BOOL data type and execute XOR operation and input the result to CR.
	ST B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <= B_VALUE1 XOR B_VALUE2 OR NOT(B_VALUE3)
	LD W_VALUE1	Input W_VALUE1 of WORD variable to CR. CR's data type is WORD.
	XOR W_VALUE2	Execute XOR operation of CR and W_VALUE2 of WORD data type and input the result to CR.
	ST W_VALUE3	Input CR to W_VALUE3 of WORD data type. W_VALUE3 <== W_VALUE1 AND W_VALUE2
	LD B_VALUE1	Input B_VALUE1 of BOOL data type to CR. CR's data type is BOOL.
	XOR(B_VALUE2	Store CR value to other position and input B_VALUE2 of BOOL data type to CR.
	AND B_VALUE3	Execute AND operation of CR value and B_VALUE3 value of BOOL data type and input the result to CR.
	)	Execute XOR operation of current CR and other CR values and input the result to CR.
	ST B_VALUE4	Input CR to B_VALUE4 of BOOL data type. B_VALUE4 <== B_VALUE1 XOR (B_VALUE2 AND B_VALUE3)

(8) ADD

Description	Executes arithmetic ADD operation of CR value and operand value and input the result to CR. The CR’s data type shall be same to the operand’s one. The operand value is not changed.	
Modifier	(:CR is stored at other position temporarily and operand’s value will be input to CR. (delay operation)	
Operand	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL and LREAL data types are available. The constant is available also.	
Example	LD I_VALUE1	Input I_VALUE1 of INT data type to CR. CR’s data type is INT.
	ADD I_VALUE2	Execute + operation of CR and I_VALUE2 of INT data type and input the result to CR.
	ST I_VALUE3	Input CR to I_VALUE3 of INT data type. I_VALUE3 <= I_VALUE1 + I_VALUE2
	LD D_VALUE1	Input D_VALUE1 of DINT data type to CR. CR’s data type is DINT.
	ADD(D_VALUE2	Store CR value to other position and input D_VALUE2 of DINT data type to CR.
	DIV D_VALUE3	Execute arithmetic / operation of CR and D_VALUE3 of DINT data type and input the result to CR.
	)	Execute + operation of current CR and other CR values and input the result to CR.
	ST D_VALUE4	Input CR to D_VALUE4 of DINT data type. D_VALUE4 <== D_VALUE1 + (D_VALUE2 / D_VALUE3)

(9) SUB

Description	Executes arithmetic - operation of CR value and operand value and input the result to CR. The CR’s data type shall be same to the operand’s one. The operand value is not changed.		
Modifier	(:CR is stored at other position temporarily and operand’s value will be input to CR. (delay operation)		
Operand	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL and LREAL data types are available. The constant is available also.		
Example	LD	I_VALUE1	Input I_VALUE1 of INT data type to CR. CR’s data type is INT.
	SUB	I_VALUE2	Execute - operation of CR and I_VALUE2 of INT data type and input the result to CR.
	ST	I_VALUE3	Input CR to I_VALUE3 of INT data type. I_VALUE3 <= I_VALUE1 - I_VALUE2
	LD	D_VALUE1	Input D_VALUE1 of DINT data type to CR. CR’s data type is DINT.
	SUB(	D_VALUE2	Store CR value to other position and input D_VALUE2 of DINT data type to CR.
	MUL	D_VALUE3	Execute arithmetic * operation of CR and D_VALUE3 of DINT data type and input the result to CR.
	)		Execute - operation of current CR and other CR values and input the result to CR.
	ST	D_VALUE4	Input CR to D_VALUE4 of DINT data type. D_VALUE4 <== D_VALUE1 - (D_VALUE2 * D_VALUE3)

(10) MUL

Description	Executes arithmetic * operation of CR value and operand value and input the result to CR. The CR’s data type shall be same to the operand’s one. The operand value is not changed.		
Modifier	(:CR is stored at other position temporarily and operand’s value will be input to CR. (delay operation)		
Operand	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL and LREAL data types are available. The constant is available also.		
Example	LD	I_VALUE1	Input I_VALUE1 of INT data type to CR. CR’s data type is INT.
	MUL	I_VALUE2	Execute * operation of CR and I_VALUE2 of INT data type and input the result to CR.
	ST	I_VALUE3	Input CR to I_VALUE3 of INT data type. I_VALUE3 <= I_VALUE1 * I_VALUE2
	LD	D_VALUE1	Input D_VALUE1 of DINT data type to CR. CR’s data type is DINT.
	MUL(	D_VALUE2	Store CR value to other position and input D_VALUE2 of DINT data type to CR.
	SUB	D_VALUE3	Execute arithmetic - operation of CR and D_VALUE3 of DINT data type and input the result to CR.
	)		Execute MUL operation of current CR and other CR values and input the result to CR.
	ST	D_VALUE4	Input CR to D_VALUE4 of DINT data type. D_VALUE4 <== D_VALUE1 * (D_VALUE2 - D_VALUE3)

(11) DIV

Description	Executes arithmetic / operation of CR value and operand value and input the result to CR. The CR's data type shall be same to the operand's one. The operand value is not changed.		
Modifier	(:CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)		
Operand	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL and LREAL data types are available. The constant is available also.		
Example	LD	I_VALUE1	Input I_VALUE1 of INT data type to CR. CR's data type is INT.
	DIV	I_VALUE2	Execute arithmetic / operation of CR and I_VALUE2 of INT data type and input the result to CR.
	ST	I_VALUE3	Input CR to I_VALUE3 of INT data type. I_VALUE3 <= I_VALUE1 / I_VALUE2
	LD	D_VALUE1	Input D_VALUE1 of DINT data type to CR. CR's data type is DINT.
	DIV(	D_VALUE2	Store CR value to other position and input D_VALUE2 of DINT data type to CR.
	ADD	D_VALUE3	Execute arithmetic + operation of CR and D_VALUE3 of DINT data type and input the result to CR.
	)		Execute arithmetic / operation of current CR and other CR values and input the result to CR.
	ST	D_VALUE4	Input CR to D_VALUE4 of DINT data type. D_VALUE4 <== D_VALUE1 / (D_VALUE2 + D_VALUE3)

(12) GT

Description	CR and operand values are compared and BOOL result is input to CR. If CR is greater than the operand, CR will be 1. Otherwise CR will be 0. The CR's data type shall be same to the operand's one. The operand value is not changed. After operation, CR's data type will be BOOL regardless of data type of operand.	
Modifier	(:CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)	
Operand	All data types excluding ARRAY are available. The constant is available also.	
Example		Ex) If I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 70
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	GT I_VAL2	Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 0 since I_VAL1 < I_VAL2)
	ST B_VAL1	Input CR to B_VAL1 variable of BOOL data type. B_VAL1 <== FALSE
	LD I_VAL2	Input I_VAL2 of INT data type to CR.
	GT I_VAL1	Compare CR and I_VAL1 of INT data type and input the result to CR.(CR is 1 since I_VAL1 < I_VAL2)
	ST B_VAL2	Input CR to B_VAL2 variable of BOOL data type. B_VAL2 <== TRUE
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	GT(I_VAL2	Store CR to other position and input I_VAL2 of INT data type to CR.
	SUB I_VAL3	Execute arithmetic / operation of CR and I_VAL3 of INT data type and input the result to CR.
	)	Compare CR stored in other position and current CR values and input the result to CR.(CR is 1 since stored CR > Current CR)
	ST B_VAL3	Input CR to B_VAL3 variable of BOOL data type. B_VAL3 <== TRUE

(13) GE

Description	CR and operand values are compared and BOOL result is input to CR. If CR is greater than or equal the operand, CR will be 1. Otherwise CR will be 0. The CR’s data type shall be same to the operand’s one. The operand value is not changed. After operation, CR’s data type will be BOOL regardless of data type of operand.		
Modifier	(:CR is stored at other position temporarily and operand’s value will be input to CR. (delay operation)		
Operand	All data types excluding ARRAY are available. The constant is available also.		
Example			Ex) If I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 70
	LD	I_VAL1	Input I_VAL1 of INT data type to CR.
	GE	I_VAL2	Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 0 since I_VAL1 < I_VAL2)
	ST	B_VAL1	Input CR to B_VAL1 variable of BOOL data type. B_VAL1 <== FALSE
	LD	I_VAL2	Input I_VAL2 of INT data type to CR.
	GE	I_VAL1	Compare CR and I_VAL1 of INT data type and input the result to CR.(CR is 1 since I_VAL1 < I_VAL2)
	ST	B_VAL2	Input CR to B_VAL2 variable of BOOL data type. B_VAL2 <== TRUE
	LD	I_VAL1	Input I_VAL1 of INT data type to CR.
	GE(	I_VAL2	Store CR to other position and input I_VAL2 of INT data type to CR.
	SUB	I_VAL3	Execute arithmetic - operation of CR and I_VAL3 of INT data type and input the result to CR.
	)		Compare CR stored in other position and current CR values and input the result to CR.(CR is 1 since stored CR > Current CR)
	ST	B_VAL3	Input CR to B_VAL3 variable of BOOL data type. B_VAL3 <== TRUE

(14) EQ

Description	CR and operand values are compared and BOOL result is input to CR. If CR is equal the operand, CR will be 1. Otherwise CR will be 0. The CR's data type shall be same to the operand's one. The operand value is not changed. After operation, CR's data type will be BOOL regardless of data type of operand.	
Modifier	(:CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)	
Operand	All data types excluding ARRAY are available. The constant is available also.	
Example		Ex) If I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 50
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	EQ I_VAL2	Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 0 since I_VAL1 < I_VAL2)
	ST B_VAL1	Input CR to B_VAL1 variable of BOOL data type. B_VAL1 <== FALSE
	LD I_VAL1	Input I_VAL2 of INT data type to CR.
	EQ I_VAL3	Compare CR and I_VAL1 of INT data type and input the result to CR.(CR is 1 since I_VAL1 < I_VAL2)
	ST B_VAL2	Input CR to B_VAL2 variable of BOOL data type. B_VAL2 <== TRUE
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	EQ(I_VAL2	Store CR to other position and input I_VAL2 of INT data type to CR.
	SUB I_VAL3	Execute arithmetic - operation of CR and I_VAL3 of INT data type and input the result to CR.
	)	Compare CR stored in other position and current CR values and input the result to CR.(CR is 1 since stored CR = Current CR)
	ST B_VAL3	Input CR to B_VAL3 variable of BOOL data type. B_VAL3 <== TRUE

(15) NE

<table>
<tr><td>Description</td><td colspan="2">CR and operand values are compared and BOOL result is input to CR.
If CR is not equal the operand, CR will be 1.
Otherwise CR will be 0.
The CR’s data type shall be same to the operand’s one.
The operand value is not changed.
After operation, CR’s data type will be BOOL regardless of data type of operand.</td></tr>
<tr><td>Modifier</td><td colspan="2">(:CR is stored at other position temporarily and operand’s value will be input to CR.
(delay operation)</td></tr>
<tr><td>Operand</td><td colspan="2">All data types excluding ARRAY are available.
The constant is available also.</td></tr>
<tr><td>Example</td><td></td><td>Ex) If I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 50</td></tr>
<tr><td></td><td>LD I_VAL1</td><td>Input I_VAL1 of INT data type to CR.</td></tr>
<tr><td></td><td>NE I_VAL3</td><td>Compare CR and I_VAL3 of INT data type and input the result to CR.(CR is 0 since I_VAL1 = I_VAL3)</td></tr>
<tr><td></td><td>ST B_VAL1</td><td>Input CR to B_VAL1 variable of BOOL data type.
B_VAL1 <== FALSE</td></tr>
<tr><td></td><td>LD I_VAL1</td><td>Input I_VAL1 of INT data type to CR.</td></tr>
<tr><td></td><td>NE I_VAL2</td><td>Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 1 since I_VAL1 <> I_VAL2)</td></tr>
<tr><td></td><td>ST B_VAL2</td><td>Input CR to B_VAL2 variable of BOOL data type.
B_VAL2 <== TRUE</td></tr>
<tr><td></td><td>LD I_VAL1</td><td>Input I_VAL1 of INT data type to CR.</td></tr>
<tr><td></td><td>NE(I_VAL2</td><td>Store CR to other position and input I_VAL2 of INT data type to CR.</td></tr>
<tr><td></td><td>SUB I_VAL3</td><td>Execute arithmetic - operation of CR and I_VAL3 of INT data type and input the result to CR.</td></tr>
<tr><td></td><td>)</td><td>Compare CR stored in other position and current CR values and input the result to CR.(CR is 0 since stored CR = Current CR)</td></tr>
<tr><td></td><td>ST B_VA3</td><td>Input CR to B_VAL3 variable of BOOL data type.
B_VAL2 <== FALSE</td></tr>
</table>

(16) LE

Description	CR and operand values are compared and BOOL result is input to CR. The operand is less than or equal the CR, CR will be 1. Otherwise CR will be 0. The CR's data type shall be same to the operand's one. The operand value is not changed. After operation, CR's data type will be BOOL regardless of data type of operand.	
Modifier	(:CR is stored at other position temporarily and operand's value will be input to CR. (delay operation)	
Operand	All data types excluding ARRAY are available. The constant is available also.	
Example		Ex) I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 70
	LD I_VAL2	Input I_VAL2 of INT data type to CR.
	LE I_VAL1	Compare CR and I_VAL1 of INT data type and input the result to CR.(CR is 0 since I_VAL1 < I_VAL2)
	ST B_VAL1	Input CR to B_VAL1 variable of BOOL data type. B_VAL1 <== FALSE
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	LE I_VAL2	Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 1 since I_VAL1 < I_VAL2)
	ST B_VAL2	Input CR to B_VAL2 variable of BOOL data type. B_VAL2 <== TRUE
	LD I_VAL1	Input I_VAL1 of INT data type to CR.
	LE(I_VAL2	Store CR to other position and input I_VAL2 of INT data type to CR.
	SUB I_VAL3	Execute arithmetic - operation of CR and I_VAL3 of INT data type and input the result to CR.
	)	Compare CR stored in other position and current CR values and input the result to CR.(CR is 0 since stored CR > Current CR)
	ST B_VA3	Input CR to B_VAL3 variable of BOOL data type. B_VAL2 <== FALSE

(17) LT

Description	CR and operand values are compared and BOOL result is input to CR. If CR is less than the operand, CR will be 1. Otherwise CR will be 0. The CR’s data type shall be same to the operand’s one. The operand value is not changed. After operation, CR’s data type will be BOOL regardless of data type of operand.		
Modifier	(:CR is stored at other position temporarily and operand’s value will be input to CR. (delay operation)		
Operand	All data types excluding ARRAY are available. The constant is available also.		
Example			Ex) If I_VAL1 = 50, I_VAL2 = 100 IVAL_3 = 70
	LD	I_VAL2	Input I_VAL2 of INT data type to CR.
	LT	I_VAL1	Compare CR and I_VAL1 of INT data type and input the result to CR.(CR is 0 since I_VAL1 < I_VAL2)
	ST	B_VAL1	Input CR to B_VAL1 variable of BOOL data type. B_VAL1 <== FALSE
	LD	I_VAL1	Input I_VAL1 of INT data type to CR.
	LT	I_VAL2	Compare CR and I_VAL2 of INT data type and input the result to CR.(CR is 1 since I_VAL1 < I_VAL2)
	ST	B_VAL2	Input CR to B_VAL2 variable of BOOL data type. B_VAL2 <== TRUE
	LD	I_VAL1	Input I_VAL1 of INT data type to CR.
	LT(	I_VAL2	Store CR to other position and input I_VAL2 of INT data type to CR.
	SUB	I_VAL3	Execute arithmetic - operation of CR and I_VAL3 of INT data type and input the result to CR.
	)		Compare CR stored in other position and current CR values and input the result to CR.(CR is 0 since stored CR > Current CR)
	ST	B_VA3	Input CR to B_VAL3 variable of BOOL data type. B_VAL2 <== FALSE

(18) JMP

Description	Move the execution flow with the label described in the operand.		
Modifier	C : Move to label if CR of BOOL data type is TRUE(1). Next command is executed without moving if CR of BOOL data type is FALSE(0). N : Move to label if CR of BOOL data type is FALSE(0). Next command is executed without moving if CR of BOOL data type is TRUE(1). Move to the label regardless of CR value if there is no modifier.		
Operand	Label name		
Example			Program that input I_VAL1 or I_VAL2 to I_VAL3 according to B_VAL1 of BOOL data type.
	LD	B_VAL1	Input B_VAL1 of BOOL data type to CR.
	JMPC	THERE1	If CR is 1, move to THERE1 label and if CR is 0, execute next sentence.
	LD	I_VAL1	CR <== I_VAL1
	JMP	THERE2	Move to THERE2 label.
	THERE1:		THERE1 label
	LD	I_VAL2	CR <== I_VAL2
	THERE2:		THERE2 label
	ST	I_VAL3	I_VAL3 <== CR
			Program that executes SEL function if B_VAL2 of BOOL data type is 1.
	LD	B_VAL2	CR <== B_VAL2
	JMPN	THERE3	If CR is 0(FALSE), move to THERE3 label.
	LD	B_VALUE	CR <== B_VALUE
	SEL		Call SEL function.
	G:=	CURRENT RESULT	
	IN1:=	I_VAL1	
	IN2:=	I_VAL2	
	ST	I_VAL3	I_VAL3 <== CR
	THERE3:		THERE3 label

(19) CAL

Description	Call the function block named in the operand.	
Modifier	C : The function block is called if CR of BOOL data type is TRUE(1). The function block is not called if CR of BOOL data type is FALSE(0). N : The function block is called if CR of BOOL data type is FALSE(0). The function block is not called if CR of BOOL data type is TRUE(1). Call the function block regardless of CR value if there is no modifier.	
Operand	Function block name	
Example		Program that calls TON, On Delay Timer, if B_VAL1 of BOOL data type is 1(TRUE).
	LD B_VAL1	Input B_VAL1 of BOOL data type to CR.
	CALC TON TIMER1	If CR is 1, the instance calls On delay timer TON of TIMER1.
	IN:= T_INPUT	
	PT:= PRE_TIME	
		Program that calls CTU of up-counter if B_VAL2 of BOOL data type is 0(FALSE).
	LD B_VAL2	Input B_VAL2 of BOOL data type to CR.
	CALN CTU COUNT1	If CR is 0, the instance calls Up-counter CTU of COUNTER1.
	CU:= B_UP	
	R:= B_RESET	
	PV:= 100	
		Program that calls Down-counter CTD regardless CR value.
	CAL CTD COUNT2	The instance calls Down-counter CTD of COUNTER2.
	CD:= B_DOWN	
	LD:= B_LDV	
	PV:= 300	

(20) RET

Description	Returned from the function or function block.	
Modifier	C : Return if CR of BOOL data type is TRUE(1). Not returned if CR of BOOL data type is FALSE(0). N : Return if CR of BOOL data type is FALSE(0). Not returned if CR of BOOL data type is TRUE(1). Returned regardless of CR if there is no modifier.	
Operand	None	
Example		Function that executes MUL operation of I_VAL1 and I_VAL2 of INT data types and inputs the result to I_VAL3. If arithmetic * operation error occurs, input 0 to I_VAL3 and return.
	LD I_VAL1	
	MUL I_VAL2	
	ST I_VAL3	
	LD _ERR	CR <== System error flag
	RETN	The instance is returned if CR is 0.
	LD 0	
	ST I_VAL3	I_VAL3 <== 0
	RET	Returned.

(21))

Description	Execute delayed operation using (.	
Modifier	None	
Operand	None	
Example	LD I_VAL1	I_VAL4 <== (I_VAL1 + IVAL2) * I_VAL3
	ADD I_VAL2	
	MUL I_VAL3	
	ST I_VAL4	
	LD I_VAL1	I_VAL4 <== I_VAL1 + (IVAL2 * I_VAL3)
	ADD(I_VAL2	
	MUL I_VAL3	
	)	
	ST I_VAL4	
	LD L_VAL1	L_VAL7 <== (L_VAL1 + (L_VAL2 * (L_VAL3 - L_VAL4) +
	ADD(L_VAL2	L_VAL5)) / L_VAL6
	MUL(L_VAL3	
	SUB L_VAL4	
	)	
	ADD L_VAL5	
	)	
	DIV L_VAL6	
	ST L_VAL7	

5.4. Calling functions and function blocks

- Call the function using the function name as operator.
- During calling the function, CR is input first to the function.
- If the function input is more than one, the other values is selected and the function is called.
- The output of function is input to CR.
- CR's data type will be the function output's one.

Example

```
LD      VAL
SIN
ST      RESULT          (Consider that VAL and RESULT is REAL data type)
```

If VAL variable is input to CR in first line and SIN function is called in second line, the CR will be input first input to SIN function. More than one input is impossible for SIN function and the output will be inserted to CR after SIN function execution. CR is stored to RESULT variable in third line.

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

This is the example of several input. In first line, CR is set and input as first input of SEL function. For other input, define each value and input the result to CR when calls SEL function and CR is stored to VAL3 variable.

- Use JMP(JMPN, JMPC) command to call the function optionally.

Example

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

Input %IX0.0.0 of BOOL data type to CR in first line and check the value is 1 or 0 in second line and move to THERE if %IX0.0.0 is 0 with label. If %IX0.0.0 is 1, JMP command is not executed and the function in next line is executed.

- CAL operator is used for the function block call and instance name of function block declared previously is used for the operand.
- CAL INSTANCE /* Call the function block as default. */
  CALN INSTANCE /* Call the function block if CR is BOOL 0. */
  CALC INSTANCE /* Call the function block if CR is BOOL 1. */

At this time, INSTANCE shall be declared as the instance for function block in advance.

- CR is not input for the function block. Thus, all input required for the function block shall be set. The output value can not be output with 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 the instance of TON)

The input of On-Delay Timer is two and each input is set and each function block is called. The result is stored in TIMER0.Q and TIMER0.ET and, if the value is required, use it like fifth line.

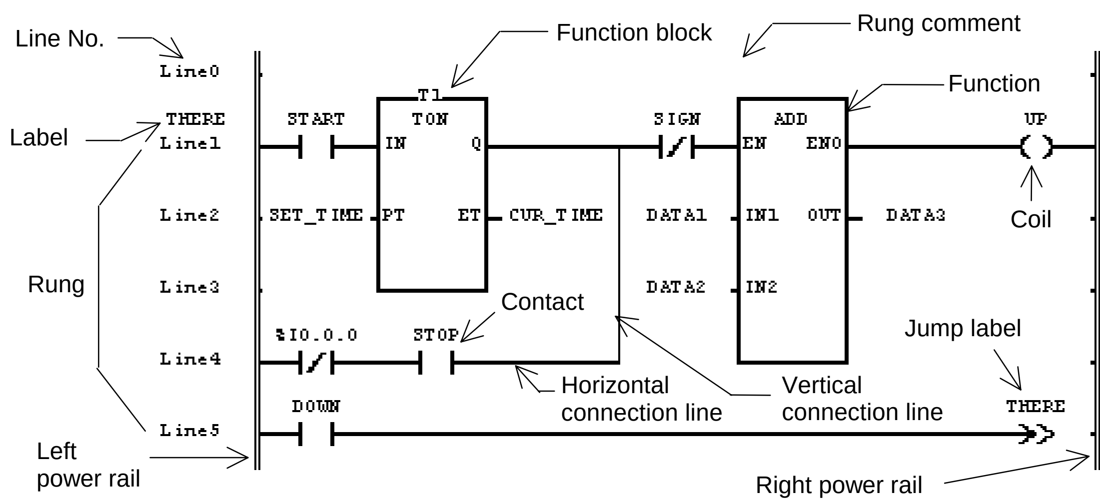
MEMO

Chapter 6 LD(Ladder Diagram)

6. LD(Ladder Diagram)

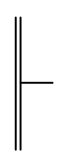
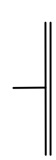
6.1. Overview

- LD program expresses the PLC program through graphic symbols such as coil or contact normally as used in relay logic diagram.
- Type



6.2. Power rails

- Base line of power line concept is laid in the left and right end of LD graphic diagram.

No.	Symbol	Description
1		Left power rail (with attached horizontal connection line) Has BOOL 1 value always.
2		Right power rail (with attached horizontal connection line) Value is not defined.

6.3. Connection line

- BOOL 1 value of left power rail is transferred to the right according to the connection. The line having transferred value is called as power flow line or connection line that is connected to the contact or coil. Power flow line always has BOOL value and only one exists in one rung. The rung means the line without the down-directed line from LD start.
- The connection line consists of horizontal and vertical connection line to connect each LD element.

No.	Symbol	Description
1	————	Horizontal connection line Transfer left value to right.
2	|	Vertical connection line Logical sum of horizontal connection lines at left side

6.4. Contacts

The contact transfers the horizontal connection line status, BOOL input/output, or Boolean AND of memory variables to the horizontal connection line at right side. The contact does not change the variable value relating to contact. The standard contact symbols are as below.

Static contacts		
No.	Symbol	Description
1	*** ─┤ ├─	Normally Open Contact When BOOL variable(marked by "***") is ON, left connection line is copied to the right. Otherwise, right connection is OFF.
2	*** ─┤ / ├─	Normally Closed Contact When BOOL variable (marked by "***") is OFF, left connection line is copied to the right. Otherwise, right connection is OFF.
Transition sensing contacts		
3	*** ─┤ P ├─	Positive transition-sensing contact The state of the right connection is ON from one evaluation of this element to the next when a transition of the associated variable from OFF to ON is sensed at the same time that the state of the left connection is ON. The state of the right connection shall be OFF at all other times.
4	*** ─┤ N ├─	Negative transition-sensing contact The state of the right connection is ON from one evaluation of this element to the next when a transition of the associated variable from ON to OFF is sensed at the same time that the state of the left connection is ON. The state of the right connection shall be OFF at all other times.

6.5. Coils

- Coil stores the result of left connection status or status transition to related BOOL variable. The standard coil symbol is as below.

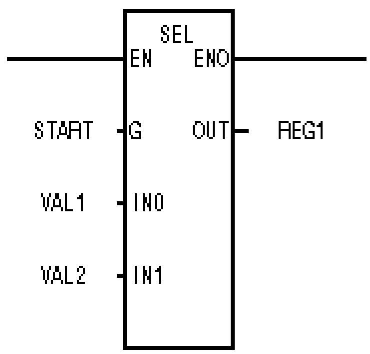
Momentary Coils		
No.	Symbol	Description
1	*** — ( ) —	Coil The state of the left connection is copied to the associated BOOL variable (marked by “***”)
2	*** — (/) —	Negated Coil Input negated left connection line status to the associated BOOL variable (marked by "***"). If left connection line is OFF, it switches relating variable ON and if left connection line ON, it switches relating variable OFF.
Latched Coils		
3	*** — (S) —	Set (Latch) Coil When left connection is ON, associated BOOL variable (marked by "***") is ON till reset by a reset coil.
4	*** — (R) —	Reset(Unlatch) Coil When left connection is ON, associated BOOL variable(marked by "***") is OFF till set by a set coil.
Transition-Sensing Coils		
5	*** — (P) —	Positive Transition-Sensing Coil When the left connection(marked by "***"), which is OFF during previous scanning, is on during current scanning, value of the associated BOOL variable (marked by "***") is ON only during current scanning.
6	*** — (N) —	Negative Transition-Sensing Coil When the left connection(marked by "***"), which is ON during previous scanning, is OFF during current scanning, value of the associated BOOL variable (marked by "***") is ON only during current scanning.

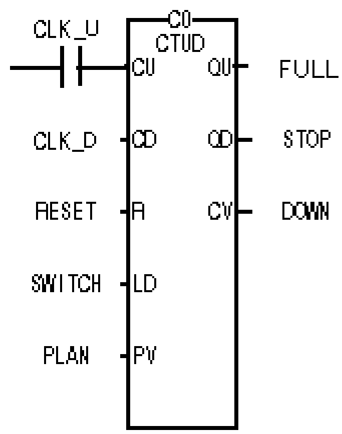
- Coil can be located only at right side of LD. There’s only right power rail at the right of coil.

6.6. Calling functions and function blocks

- Actual Input/Outptu connection for the function or function block is arranged by describing proper data or variable to the outside of function or function block body.

Example

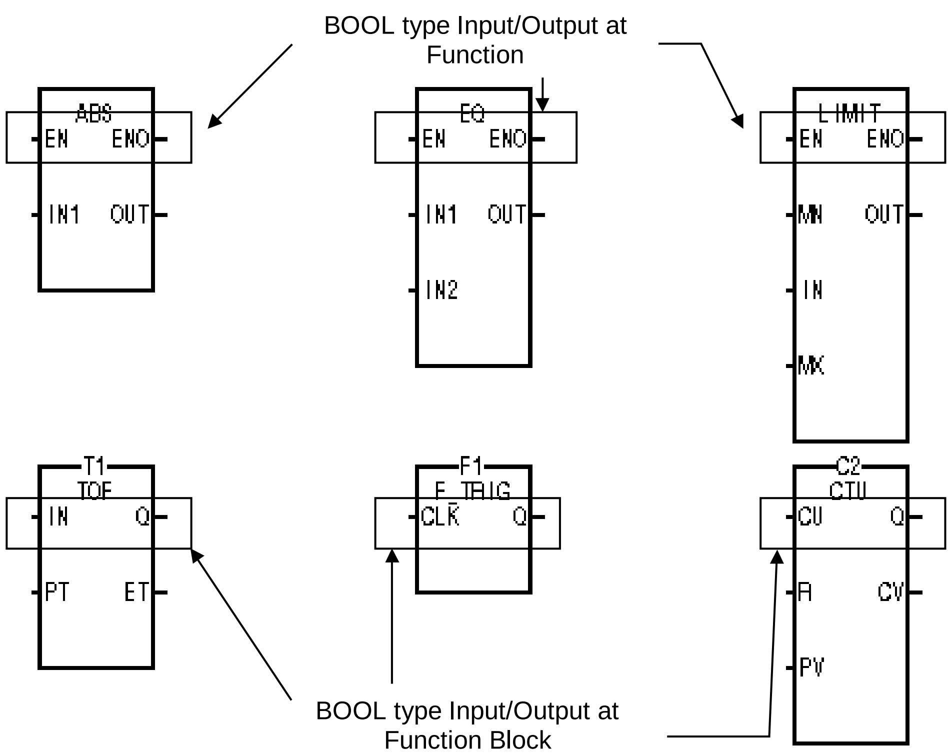




Function **Function block**

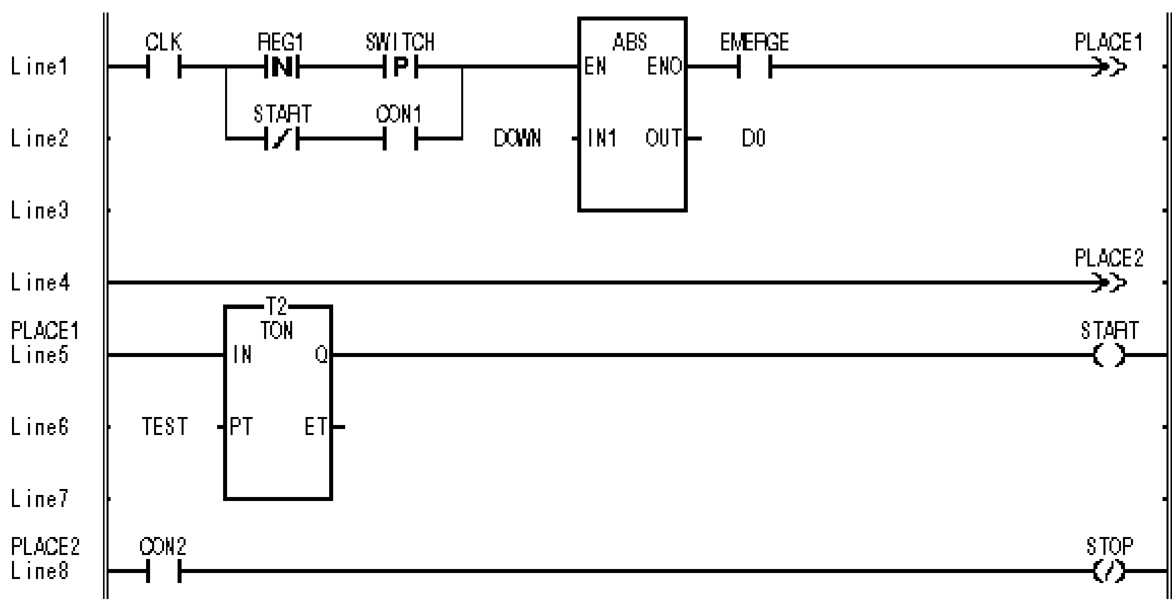
- A BOOL type input and output shall be located at each function or function block in order to allow the power flow into the function or function block. EN and ENO is BOOL type Input/Ouput at the function and first input and output is BOOL type at the function block.

Example



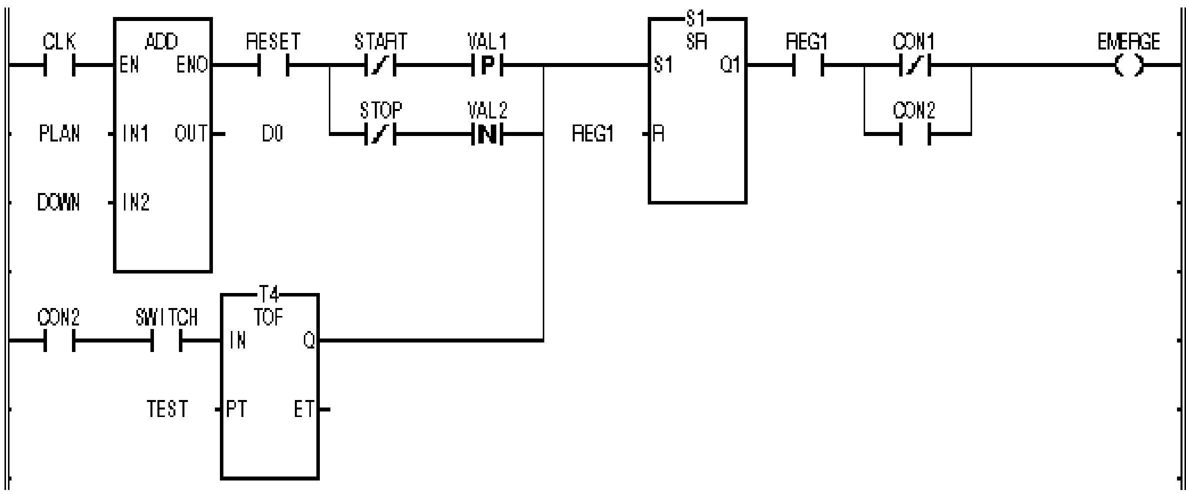
- The function at LD has EN input and ENO output unlike IL. EN and ENO are BOOL data type and the function is executed when EN input is BOOL 1 and is not executed when it is BOOL 0. ENO output is generally EN value but ENO will be BOOL 0 though EN is BOOL 1 when the function error occurs. The function's EN shall be power flow line but ENO may not be. However, when the power flow line is connected to the function output not ENO, the output data type shall be BOOL. Further, if the power flow line is connected to the function output not ENO, ENO shall not be connected to anything. All inputs of function are allocated by describing the value to left of the function and shall not be missed. The function's output is stored to the variable selected to right of the function.
- The function block at LD is used as same method at IL. The function blocks input is selected as same method. As the function blocks output is stored in the instance, the variable may not be selected. As EN and ENO Input/Output are not in the function block, the function is executed at every function block. Therefore, the jump(-->>) shall be used to define the execution of function block according to logical result. When the power flow line is connected to the function block, it shall be connected to Input/Output of BOOL data type.

Example



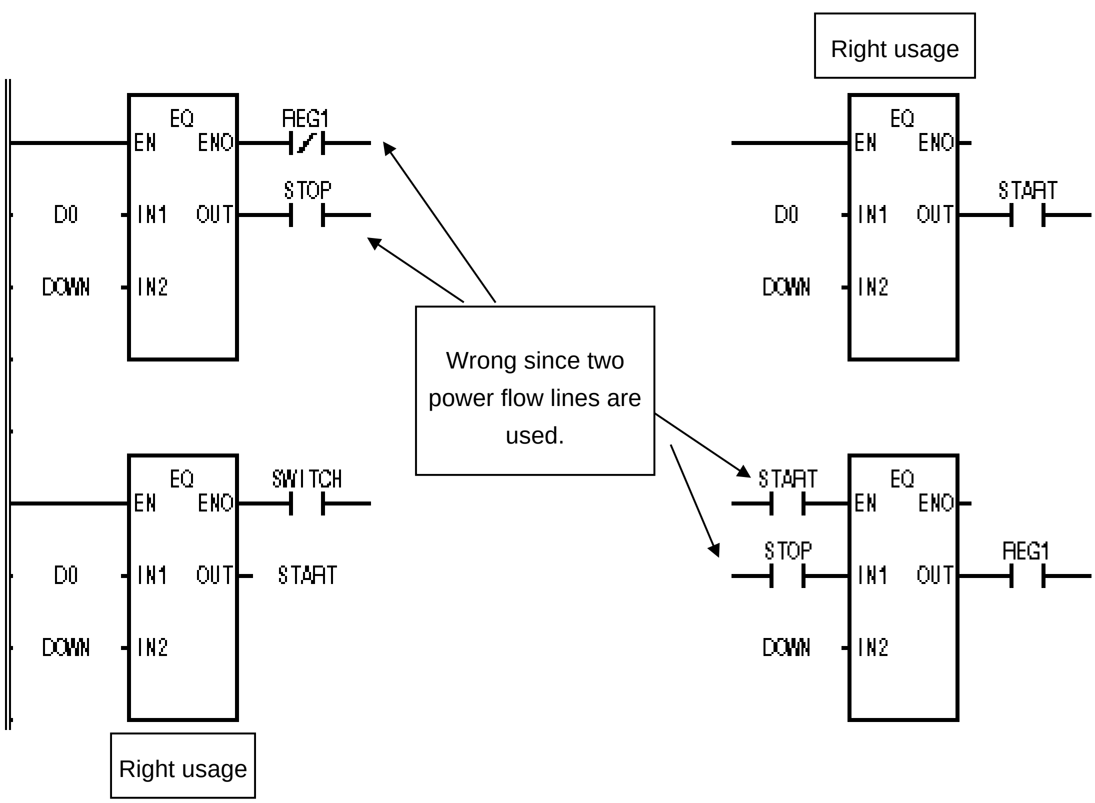
- The function and function block can be located everywhere at LD. Connecting power flow line to function and function blocks Input/Output and contact to power flow line can continue the logic operation.

Example



- The power flow line, which can be connected to one function and one function block, is only one.

Example



MEMO

Chapter 7 Functions and function blocks

7. Functions and function blocks

The list summary of functions and functions block is described in this chapter. Please refer to Chapter 8. Basic functions and function block libraries, Chapter 9. Special function block libraries and Chapter 10. Communication function block libraries.

7.1. Function

7.1.1. Type conversion function

Convert each input data types to output data types.

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.		

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.		

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
INT_TO_***	INT_TO_BCD	INT	WORD(BCD)	.	.	.
	INT_TO_REAL	INT	REAL	.		
	INT_TO_LREAL	INT	LREAL	.		
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	.		

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
LINT_TO_***	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	.		
	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	.	.	.

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.		
	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	.		

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.	.	.
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	.	.	.

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.		

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.	.	.
	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	.	.	.

Function group	Function name	Input data type	Output data type	Applied model		
				GM1~2	GM3	GM4~6
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	.	.	.

7.1.2. Numerical operation function

7.1.2.1. Numerical operation function with single input

Supported only at GM1 and GM2(GM3, GM4, GM5 and GM6 supports ABS.)

No.	Function name	Description
	General functions	
1	ABS	Absolute value operation
2	SQRT	Square root operation
	Log functions	
3	LN	Natural logarithm operation
4	LOG	Logarithm base to 10 operation
5	EXP	Natural Exponential
	Trigonal functions	
6	SIN	Sine of input in radians
7	COS	Cosine in radians
8	TAN	Tangent in radians
9	ASIN	Arc Sine value operation
10	ACOS	Arc Cosine value operation
11	ATAN	Arc Tangent value operation

7.1.2.2. Basic numerical operation function

EXPT is restricted to GM1 and GM2.

No.	Function name	Description
	Operation functions, which can extend the input number(but, n shall be OK to 8)	
1	ADD	Adds from 2 to n numbers (OUT <= IN1 + IN2 + ... + INn)
2	MUL	Multiples from 2 to n numbers (OUT <= IN1 * IN2 * ... * INn)
	Operation function with constant input number	
3	SUB	Performs the subtraction operation on 2 numbers (OUT <= IN1 - IN2)
4	DIV	Performs the division operation and (OUT <= IN1 / IN2)
6	MOD	Performs the division operation and returns the remainder (OUT <= IN1 Modulo IN2)
10	EXPT	Exponentiation (OUT <= $IN1^{IN2}$)
11	MOVE	Data copy (OUT <= IN)

7.1.3. Bit function

7.1.3.1. Bit shift function

No.	Function name	Description
1	SHL	OUT : = IN left-shifted by N bits (zero-filled on right)
2	SHR	OUT : = IN right-shifted by N bits (Zero-filled on left)
3	ROL	OUT : = IN left-rotated by N bits, circular
4	ROR	OUT : = IN right-rotated by N bits, circular

7.1.3.2. Bit operation function

No.	Function name	Description (n shall be ok 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	Logical exclusive OR(OUT : = IN1 XOR IN2 XOR ... XOR INn)
4	NOT	Logical inversion(OUT : = NOT IN1)

7.1.4. Selection function

No.	Function name	Description (n shall be ok to 8)
1	SEL	Output(Selected input between IN0 or IN1)
2	MAX	Output put max. value among IN1,...INn
3	MIN	Output put min. value among IN1,...INn
4	LIMIT	Outputs the upper or lower limit orf input
5	MUX	Outputs Kth input among IN0,…INn

7.1.5. Comparison function

No.	Function name	Description (n shall be OK to 8)
1	GT	’Greater than’ comparison OUT : = (IN1>IN2) & (IN2>IN3) & ... & (INn-1 > INn)
2	GE	’Greater than equal’ comparison OUT : = (IN1>=IN2) & (IN2>=IN3) & ... & (INn-1 >= INn)
3	EQ	’Equal’ comparison OUT : = (IN1=IN2) & (IN2=IN3) & ... & (INn-1 = INn)
4	LE	’Less than or equal’ 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’ comparison OUT : = (IN1<>IN2) & (IN2<>IN3) & ... & (INn-1 <> INn)

7.1.6. Character function

No.	Function name	Description
1	LEN	Character string length
2	LEFT	Leftmost L characters of IN
3	RIGHT	Rightmost L characters of IN
4	MID	L character of IN, beginning at the p-th
5	CONCAT	Extensible concatenation
6	INSERT	Insert IN2 into IN1 after the p-th character position
7	DELETE	Delete L characters of IN, beginning at the p-th character position
8	REPLACE	Replace L characters of IN1 by IN2, starting at the p-th character position
9	FIND	Find the character position of the beginning of the first occurrence of IN2 in IN1, if no occurrence of IN2 is found, then OUT : = 0

7.1.7. Functions of time data types

No.	Function name	Description
	Operation and concatenation functions	
1	ADD_TIME	Adds TIME to TIME, TOD or DT
2	SUB_TIME	Subtracts TIME from TIME, TOD or DT
	SUB_DATE	Subtracts DATE from DATE, result is TIME
	SUB_TOD	Subtracts TOD from TOD, result is TIME
	SUB_DT	Subtracts DT from DT, result is TIME
3	MUL_TIME	Multiply the TIME by number
4	DIV_TIME	Divide the TIME by number
5	CONCAT_TIME	Concatenates DATE and TOD to DT

7.1.8. System control function

No.	Function name	Description
1	DI	Task program operation prohibit
2	EI	Task program operation allow
3	STOP	Operation stop by program
4	ESTOP	Emergency stop by program
5	DIREC_IN	Instant refresh of input data (applicable in GM1-GM4, GM6)
6	DIREC_IN5	Instant refresh of input data (applicable in GM5)
7	DIREC_O	Instant refresh of output data (applicable in GM1-GM4, GM6)
8	DIREC_O5	Instant refresh of output data (applicable in GM5)
9	WDT_RST	Watch_Dog Timer reset

7.2. MK(MASTER-K) function libraries

No.	Function name	Description (n shall be below than 8)
1	ENCO_B,W,D,L	Outputs the most high ON bit position
2	DECO_B,W,D,L	Sets the assigned bit position to 1
3	BSUM_B,W,D,L	Outputs the number of ON bits
4	SEG	Converts BCD or HEX value to 7 segment display code
5	BMOV_B,W,D,L	Copy and move some part of bit strings
6	INC_B,W,D,L	Increases IN data
7	DEC_B,W,D,L	Decreases IN data

7.3. Function blocks

7.3.1. Bistable function block

No.	Function block name	Description
1	SR	Set priority bistable output
2	RS	Reset priority bistable output
3	SEMA	Semaphore for system resource control

7.3.2. Edge detection function block

No.	Function block name	Description
1	R_TRIG	Rising Edge Detector
2	F_TRIG	Falling Edge Detector

7.3.3. Counter function block

No.	Function block name	Description
1	CTU	Up Counter
2	CTD	Down Counter
3	CTUD	Up Down Counter

7.3.4. Timer function block

No.	Function block name	Description
1	TP	Pulse Timer
2	TON	On-Delay Timer
3	TOF	Off-Delay Timer

7.4. Analog function blocks(For special module only)

7.4.1. A/D function block

No.	GM3		GM4		GM6	Description
	Local	Remote	Local	Remote	Local	
1	AD4INI	ADR4INI	AD2INI	ADR2INI	AD2INI	Module initialization
2	AD4ARD	ADR4RD	AD2ARD	ADR2RD	AD2ARD	A/D converting value reading(Array type)
3	AD4RD	-	AD2RD	-	AD2RD	A/D converting value reading(Stand-alone type)

7.4.2. A/T(Analog Timer) function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	AT4TON	-	AT3TON	-	Activates analog timer

7.4.3. D/A function block

No.	GM3		GM4		GM6	Description
	Local	Remote	Local	Remote	Local	
1	DA4INI	DAR4INI	DA1INI	DAR1INI	-	Module initialization
2	DA4AWR	DAR4WR	DA1AWR	DAR1WR	DA1AWR	Digital data writing(Array type)
3	DA4WR	-	DA1WR	-	DA1WR	Digital data writing(Stand-alone type)

7.4.4. T/C(Thermo-Couple) function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	TC4INI	TCR4INI	TC2INI	TCR2INI	Module initialization
2	TC4ARD	TCR4RD	TC2ARD	TCR2RD	Temperature conversion value reading (Array type)
3	TC4RD	-	TC2RD	-	Temperature conversion value reading (Stand-alone type)

7.4.5. RTD(Resistor Temperature Detection) function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	RTD3INI	RTDR3INI	RTD2INI	RTDR2INI	Module initialization
2	RTD3ARD	RTDR3RD	RTD2ARD	RTDR2RD	RTD temperature conversion value reading (Array type)
3	RTD3RD	-	RTD2RD	-	RTD temperature conversion value reading (Stand-alone type)

7.4.6. PID function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	PID5INI	-	PID3INI	-	Module initialization
2	PID5ARD	-	PID3ARD	-	PID operation value reading(Array type)
3	PID5RD	-	PID3RD	-	PID operation value reading(Stand-alone type)

7.4.7. High-speed counter function block

No.	GM3		GM4		GM6	Description
	Local	Remote	Local	Remote	Local	
1	HSC_CMP	HSCR1CMP	HSC_CMP	HSCR0CMP	HSC_CMP	Module comparison value selection
2	HSC_PRE	HSCR1PRE	HSC_PRE	HSCR1PRE	HSC_PRE	Module Preset value selection
3	HSC_WR	HSCR1WR	HSC_WR	HSCR0WR	HSC_WR	Output information selection
4	HSC_RD	HSCR1RD	HSC_RD	HSCR0RD	HSC_RD	Input information selection

7.4.8. Position control(Analog output) function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	POSA_AST	-	-	-	General automatic positioning operation instruction
2	POSA_CRD	-	-	-	Current operation status reading
3	POSA_EMG	-	-	-	Emergency stop instruction
4	POSA_FLT	-	-	-	Movable zero point set instruction
5	POSA_JOG	-	-	-	JOG operation instruction
6	POSA_MOF	-	-	-	M code off instruction
7	POSA_NM	-	-	-	Next move operation instruction
8	POSA_OR	-	-	-	Override operation instruction
9	POSA_ORG	-	-	-	Zero point return instruction
10	POSA_RES	-	-	-	Error reset instruction
11	POSA_RTP	-	-	-	Position retrieval instruction before manual operation
12	POSA_SMC	-	-	-	Operation data number change instruction
13	POSA_SRD	-	-	-	Current operation status bit reading
14	POSA_TEA	-	-	-	Position teaching instruction
15	POSA_TMP	-	-	-	Deceleration stop instruction
16	POSA_TPB	-	-	-	Teaching playback instruction
17	POSA_VCG	-	-	-	Speed change instruction
18	POSA_VLT	-	-	-	Speed teaching instruction

7.4.9. Position control(Pulse output) function block

No.	GM3		GM4		Description
	Local	Remote	Local	Remote	
1	POSP_AST	-	POSP_AST	-	General automatic positioning operation instruction
2	POSP_CRD	-	POSP_CRD	-	Current operation status reading
3	POSP_EMG		POSP_EMG		Emergency stop instruction
4	POSP_FLT	-	POSP_FLT	-	Movable zero point set instruction
5	POSP_INC	-	POSP_INC	-	Inching operation instruction
6	POSP_INT	-	-	-	Interpolation instruction
7	POSP_JOG	-	POSP_JOG	-	JOG operation instruction
8	POSP_MOF	-	POSP_MOF	-	M Code off instruction
9	POSP_MPG	-	-	-	Manual pulse generator enable instruction
10	POSP_NM	-	POSP_NM	-	Next move operation instruction
11	POSP_OFF		POSP_OFF		Output prohibit release instruction
12	POSP_OR	-	POSP_OR	-	Override operation instruction
13	POSP_ORG	-	POSP_ORG	-	Zero point return instruction
14	POSP_PRE	-	POSP_PRE	-	Current position preset instruction
15	POSP_RES	-	POSP_RES	-	Error reset instruction
16	POSP_RTP	-	POSP_RTP	-	Position return instruction before the manual operation
17	POSP_SMC	-	POSP_SMC	-	Operation data number change instruction
18	POSP_SRD	-	POSP_SRD	-	Current operation status bit reading
19	POSP_TEA	-	POSP_TEA	-	Position teaching instruction
20	POSP_TMP	-	POSP_TMP	-	Deceleration stop instruction
21	POSP_VCG	-	POSP_VCG	-	Speed change instruction
22	POSP_VLT	-	POSP_VLT	-	Speed Teaching instruction

7.5. Communication function blocks

No.	Function block name	Description
1	CONNECT	Establishes logical communication channel between self-station and other station (only for Mini_MAP)
2	RDARRAY	Reading array type data from other station
3	RDBLOCK	Reading the block data from other station (Max. 450 Bytes)
4	RDTYPE(BOOL...DT)	Reading data from other station
5	STATUS	Reading status of other station
6	WRARRAY	Writing the array type data to other station
7	WRBLOCK	Writing block data to other station (Max. 450 Bytes)
8	WRTYPE(BOOL...DT)	Writing data to other station

7.6. Computer communication module function blocks

No.	Function block name	Description
1	SND_MSG	Sending the defined frame data to other station
8	RCV_MSG	Receiving data from other station

Chapter 8 Function/Function block libraries

8. Function/Function block libraries

8.1 Function libraries

This chapter describes function libraries.

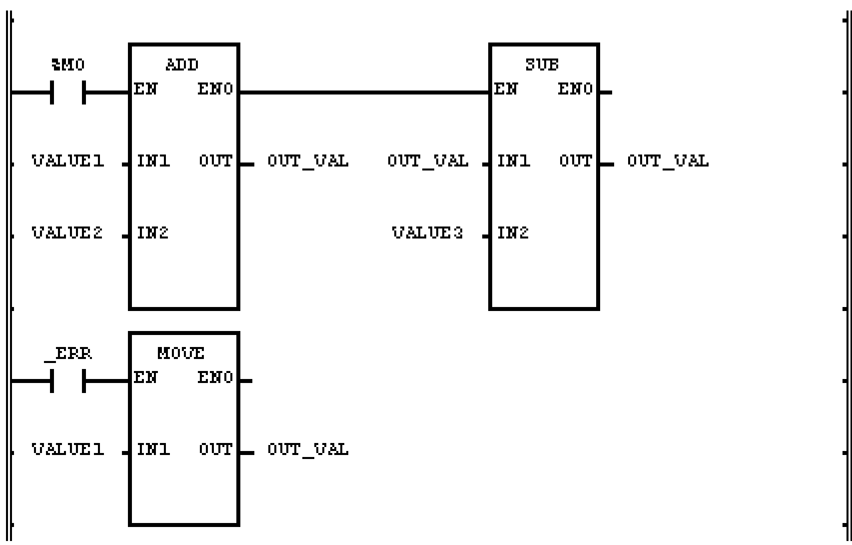
Point Please refer to below description when the function error occurs.

- Function error
  When the error occurs during the function, ENO will be 0 and error flag(_ERR, _LER) will be 1.
  ENO of the function without error outputs EN input. EN and ENO are used only in LD(Ladder Diagram).
- Error flag
  _ERR (Error)
  - _ERR value will be changed as below after operating the function marking no error.
    (The function marking no error maintains _ERR status before operation.)
  - For the operation error, _ERR value will be 1.
  - Except the operation error, _ERR value will be 0.

  _LER (Latched Error)
  - _LER will be 1 for the error after operation and will be maintained till the current program block is completed.
  - 0 can be writable by program.

■ Program example

Program that does not execute SUB function while ADD function error and stores VALUE1 to OUT_VAL.



(1) If two inputs of function(ADD) are as below, the function error occurs.
Input(IN1) : VALUE1(SINT) = 100(16#64)
(IN2) : VALUE2(SINT) = 50(16#32)
Output(OUT) : OUT_VAL(SINT) = -106(16#96)

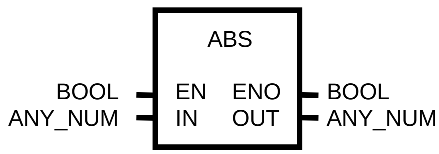
(2) The output exceeds the range of output data type and OUT_VAL(SINT) stores abnormal value.
ENO of function(ADD) will be 0 and the function(SUB) is not executed and the error flag _ERR and _LER will be on.

(3) _ERR is on and the function(MOVE) will be executed.
Input(IN1): VALUE1(SINT) = 100(16#64)
Output(OUT): OUT_VAL(SINT) = 100(16#64)

ABS

Absolute value operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td></td><td>Input EN : Execute the function in case of 1
IN : Input value of absolute operation

Output ENO : Output 1 in case of no error
OUT : Absolute value

IN and OUT shall be same data type.</td></tr>
</table>

■ **Function**

Convert IN value to absolute and output to OUT.

X of absolute.X.will be

.X.= X if X>=0,

.X.= -X if X<0.

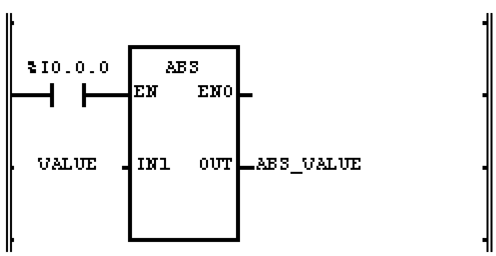
OUT = .IN.

■ **Error**

If IN value is lower limit of minus value of given data type, _ERR and _LER flag will be set.

Ex) If the data type is SINT and IN value is -128, it is error.

■ **Program example**

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td></td><td>

LD	%I0.0.0
JMPN	AL
LD	VALUE
ABS	
ST	ABS_VALUE
AL :	

</td></tr>
</table>

(1) If the execution condition(% I0.0.0) is On, the function ABS is 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	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	1	1	1

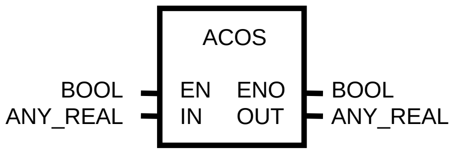
(16#0007)

Note Negative expression of INT type is described by 2˚S Complement form(Refer to 3.2.4. Data type structure)

ACOS

Arc Cosine operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of Arc Cosine operation **Output** ENO : Output 1 in case of no error OUT : Radian value of the result IN and OUT shall be same data type.

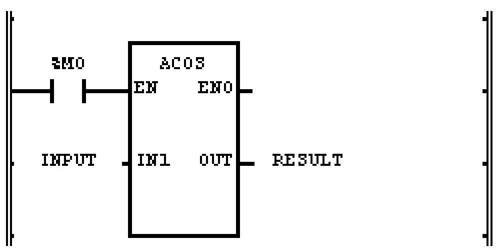
■ Function

Calculate IN's Arc Cosine and output to OUT. The output value will be between 0 and ..
OUT = ACOS (IN)

■ Error

If IN1 exceeds the range from -1.0 to 1.0, _ERR and _LER flag is set.

■ Program example

LD	IL
	LD %M0 JMPN LL LD INPUT ACOS ST RESULT LL :

(1) If the execution condition(%M0) is On, Arc Cosine operation function ACOS is executed.

(2) If INPUT variable is 0.8660 .... ($\sqrt{3}$ /2), the result will be 0.5235 ... (./6 rad = 30°).

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

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

Input(IN1) : INPUT(REAL) = 0.866

.(ACOS)

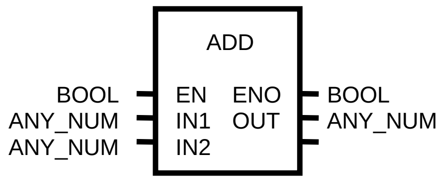
Output(OUT) : RESULT (REAL) = 5.23499966E-01

Note Expression of REAL type mark is based on IEEE Standard 754-1984 (Refer to 3.2.4. Data type structure)

ADD

Add

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Augend IN2 : Addend Can be extended to 8 inputs **Output** ENO : Output 1 in case of no error OUT : ADD Result Variables connected to IN1, IN2, ..., OUT shall be same data type.

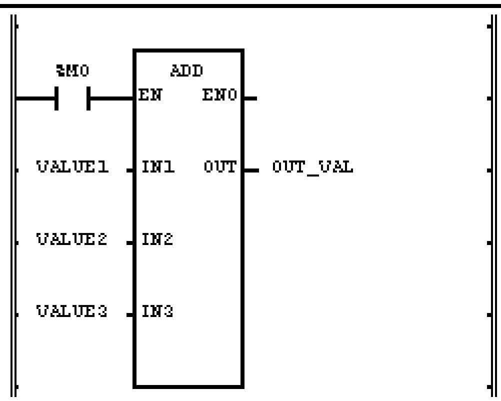
■ **Function**

Add IN1, IN2,..., INn (n: input number) and output to OUT.
OUT = IN1 + IN2 + ... + INn

■ **Error**

If the output exceeds the range of given data type, _ERR and _LER flag will be set.

■ **Program example**

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

(1) When the execution condition(%M0) is On, ADD function is executed.
(2) If VALUE1 = 300, VALUE2 = 200, VALUE3 = 100, OUT_VAL = 300 + 200 + 100 = 600.

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

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	1	1	0	0	0	1	0	0	0

+ (ADD)

(IN2) : VALUE3(INT) = 100(16#0064)

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

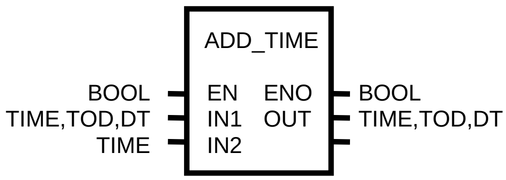
Output(OUT) : OUT_VAL(INT) = 600(16#0258)

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

ADD_TIME

Add time

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Reference time, time of day or date IN2 : Time to be added **Output** ENO : Output 1 in case of no error OUT : Add result of day or time or date OUT type depends on input IN1. If IN1 type is TIME_OF_DAY, OUT type will be also TIME_OF_DAY.

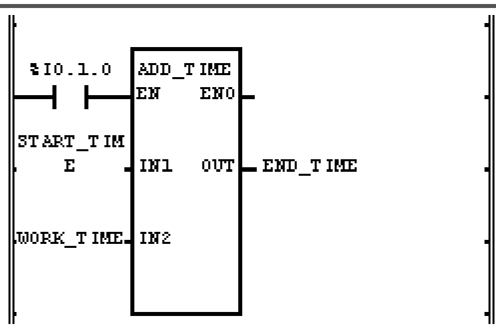
■ **Function**

. If IN1 is TIME, added TIME will be output.
. If IN1 is TIME_OF_DAY, add the TIME to reference TIME_OF_DAY and output the TIME_OF_DAY.
. If IN1 is DATE_AND_TIME, add the TIME to reference DATE_OF_TIME and output the DATE_AND_TIME.

■ **Error**

. If the output exceeds the range of given data type, _ERR and _LER flag will be set.
. If the result of adding TIME exceeds the range of TIME data type, T#49D17H2M47S295MS, the result of adding TOD and TIME exceeds 24 hours or the result of adding, DT and time exceeds 2083 YEAR, it will be error.

■ **Program example**

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

(1) If the execution condition(%I0.1.0) is On, time ADD function, ADD_TIME, is executed.
(2) If START_TIME is TOD#08:30:00 and WORK_TIME is T#2H10M20S500MS, TOD#10:40:20.5 will be output to END_TIME.

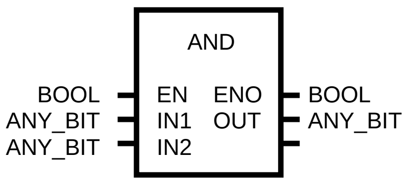
Input(IN1) : START_TIME(TOD) = TOD#08:30:00
+ (ADD_TIME)
(IN2) : WORK_TIME(TIME) = T#2H10M20S500MS

Output(OUT) : END_TIME(TOD) = TOD#10:40:20.5

AND

Logical AND

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Input1 IN2 : Input2 Can be extended to 8 inputs. **Output** ENO : Output EN value itself OUT : AND result IN1, IN2 and OUT shall be same type.

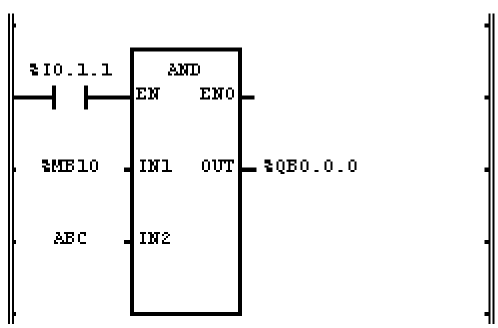
■ **Function**

Execute AND IN1 to IN2 by bit and output the result to OUT.

IN1 1111 0000
&
IN2 1010 1010
OUT 1010 0000

■ **Program example**

LD



IL

```
LD                  %I0.1.1
JMPN                AA
LD                  %MB10
AND     IN1:=       CURRENT RESULT
        IN2:=       ABC
ST               %QB0.0.0
AA :
```

(1) If the execution condition(%I0.1.1) is On, the function AND is executed.
(2) The AND result of INI= %MB10 and IN2 = ABC is output to 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

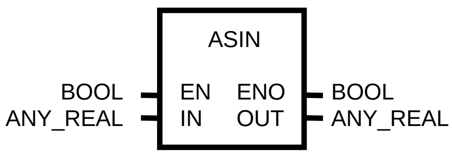
Output(OUT) : %QB0.0.0(BYTE) = 16#C0

1	1	0	0	0	0	0	0

ASIN

Arc Sine operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of Arc Sine operation **Output** ENO : Output 1 in case of no error OUT : Radian result of operation result IN and OUT shall be same type.

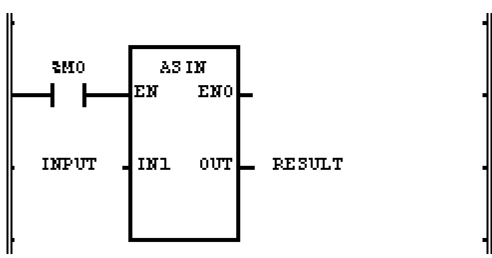
■ **Function**

Output IN's Arc Sine value to OUT. The output value is between -./2 to ./2 .
OUT = ASIN (IN)

■ **Error**

If the input value exceeds the range from -1.0 to 1.0, _ERR and _LER flag is set.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD INPUT ASIN ST RESULT AAA :

(1) If the execution condition(%M0) is On, Arc Sine operation function ASIN is executed.

(2) If INPUT variable is 0.8660 .... ($\sqrt{3}$ /2), RESULT declared as output variable will be 1.0471 .... (./3 rad = 60°).

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

(SIN(./3) = ($\sqrt{3}$ /2)

Input(IN1) : INPUT(REAL) = 0.866

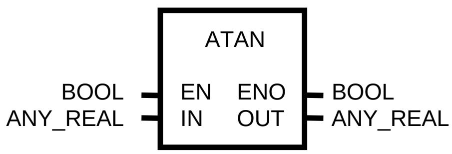
.(ASIN)

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

ATAN

Arc Tangent operation

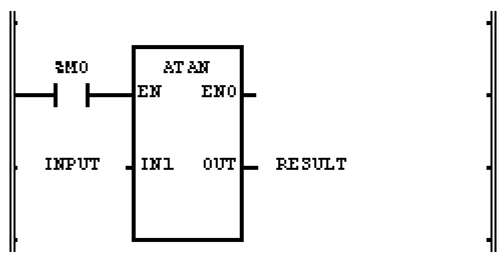
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of Arc Tangent operation **Output** ENO : Output EN value itself OUT : Radian output value of operation result IN and OUT shall be same type.

■ **Function**

Output IN's Arc Tangent value to OUT. The output value is between -./2 and ./2.
OUT = ATAN (IN)

■ **Program example**

LD	IL
	LD %M0 JMPN AA LD INPUT ATAN ST RESULT AA :

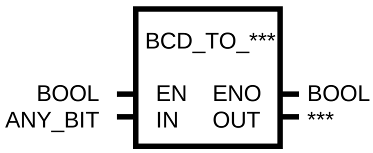
(1) If the execution condition(%M0) is On, Arc Tangent operation function ASIN is executed.
(2) If INPUT variable is 1.0, RESULT declared as output variable will be ./4 = 0.7853 ....
ATAN (1) = ./4
(TAN(./4) = 1)

Input(IN1) : INPUT(REAL) = 1.0
.(ATAN)
Output(OUT) : RESULT(REAL) =7.85398185E-01

BCD_TO_***

Convert BCD type to integer

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : ANY_BIT input with BCD type data **Output** ENO : Output EN value itself OUT : Type converted data

■ Function

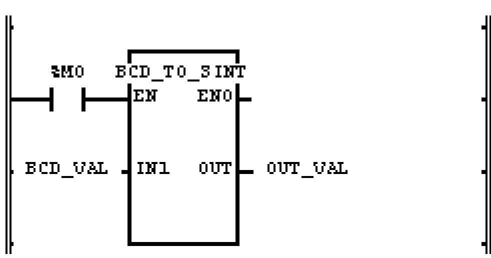
Convert IN to OUT data type.

FUNCTION	Input type	Output type	Description
BCD_TO_SINT	BYTE	SINT	Convert BCD to output data type. Normal conversion is executed only if the input is BCD value. (If input data type is WORD, 0.16#9999 value is 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 BCD type data, the output will be 0 and _ERR and _LER flag is set.

■ Program example

LD	IL
	LD %M0 JMPN ABC LD BCD_VAL BCD_TO_SINT ST OUT_VAL ABC :

(1) If the execution condition(%M0) is On, the function BCD_TO_*** is executed.

(2) If BCD_VAL(BYTE type) = 16#22(2#0010_ 0010), OUT_VAL(SINT type) = 22(2#0001_ 0110) declared as output variable will be output.

Input(IN1) :BCD_VAL(BYTE) = 16#22

0	0	1	0	0	0	1	0

.(BCD_TO_SINT)

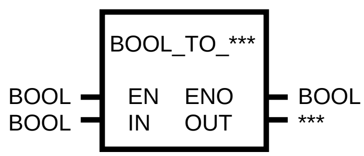
Output(OUT): OUT_VAL(SINT) = 22

0	0	0	1	0	1	1	0

BOOL_TO_***

BOOL type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

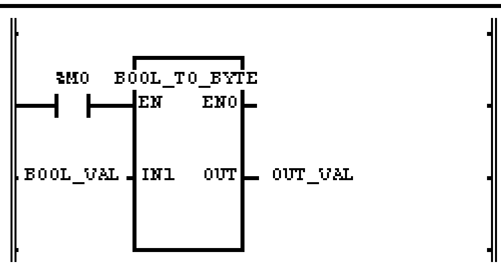
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit to be converted(1 bit) **Output** ENO : Output EN value itself OUT : Type converted data

■ Function

Convert IN to OUT data type and output.

FUNCTION	Output type	Description
BOOL_TO_SINT	SINT	If BOOL input is 2#0, output integer ’0’ and if it is 2#1, output integer ’1’, according to 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	Convert BOOL to output data type filling upper bit with 0.
BOOL_TO_WORD	WORD	
BOOL_TO_DWORD	DWORD	
BOOL_TO_LWORD	LWORD	
BOOL_TO_STRING	STRING	Convert BOOL to STRING type. Convert it to ’0’ or ’1’.

■ Program example

LD	IL
	LD %M0 JMPN ABC LD BOOL_VAL BOOL_TO_BYTE ST OUT_VAL ABC :

(1) If the execution condition(%M0) is On, the function BOOL_TO_*** is executed.

(2) If BOOL_VAL(BOOL type) = 2#1, OUT_VAL(BYTE type) = 2#0000_ 0001 declared as output variable will be output.

Input(IN1) : BOOL_VAL(BOOL) = 2#1

1

(BOOL_TO_SINT).

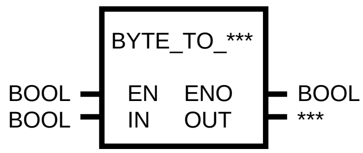
Output(OUT): OUT_VAL(BYTE) = 16#1

0	0	0	0	0	0	0	1

BYTE_TO_***

BYTE type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

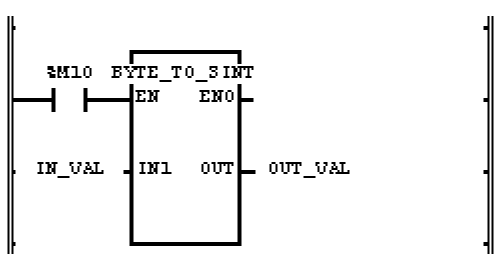
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit string to be converted(8bit) **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type and output.

FUNCTION	Output type	Description
BYTE _TO_SINT	SINT	Convert internal bit array to SINT type without conversion.
BYTE _TO_INT	INT	Convert INT to output data type filling upper bit with 0.
BYTE _TO_DINT	DINT	Convert DINT to output data type filling upper bit with 0.
BYTE _TO_LINT	LINT	Convert LINT to output data type filling upper bit with 0.
BYTE _TO_USINT	USINT	Convert internal bit array to USINT type without conversion.
BYTE _TO_UINT	UINT	Convert UINT to output data type filling upper bit with 0.
BYTE _TO_UDNT	UDINT	Convert UDINT to output data type filling upper bit with 0.
BYTE _TO_ULINT	ULINT	Convert ULINT to output data type filling upper bit with 0.
BYTE _TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
BYTE _TO_WORD	WORD	Fill upper bit with 0 to convert it to WORD type.
BYTE _TO_DWORD	DWORD	Fill upper bit with 0 to convert it to DWORD type.
BYTE _TO_LWORD	LWORD	Fill upper bit with 0 to convert it to LWORD type.
BYTE _TO_STRING	STRING	Convert input value to STRING type.

■ **Program example**

LD	IL
	LD %M10 JMPN LLL LD IN_VAL BYTE_TO_SINT ST OUT_VAL LLL :

(1) If the execution condition(%M10) is On, the function BYTE_TO_SINT is executed.
(2) If IN_VAL(BYTE type) = 2#0001_1000, OUT_VAL(SINT type) = 24(2#0001_1000).

Input(IN1) : IN_VAL(BYTE) = 16#18

0	0	0	1	1	0	0	0

.(BYTE_TO_SINT)

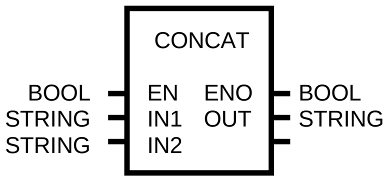
Output(OUT) : OUT_VAL(SINT) = 24

0	0	0	1	1	0	0	0

CONCAT

Character string concatenation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
 	**Input** EN : Execute the function in case of 1 IN1 : Character string input IN2 : Character string input Can be extended to 8 inputs. **Output** ENO : Output 1 in case of no error OUT : Character string output

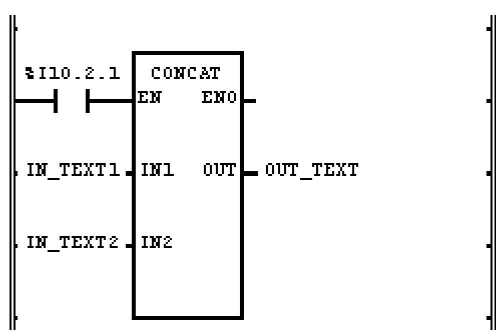
■ **Function**

Concatenates input character string in order of IN1, IN2, IN3,...., INn(n: input number) and outputs to the output character string OUT.

■ **Error**

If (character sum of all input character string) > 30, just 30 characters of concatenated each input character strings are output and _ERR and _LER flag is set.

■ **Program example**

LD	IL
 	LD %0.2.1 JMPN THERE LD IN_TEXT1 CONCAT IN1:= CURRENT RESULT IN2:= IN_TEXT2 ST OUT_TEXT THERE :

(1) If the execution condition(%I0.2.1) is On, the function CONCAT is executed.
(2) If IN_TEXT1=‘ABCD‘ and IN_TEXT2=‘DEF‘, OUT_TEXT=‘ABCDDEF‘.

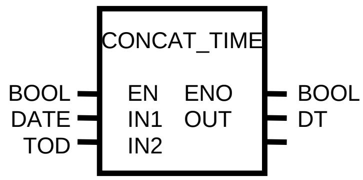
Input(IN1) : IN_TEXT1(STRING) = ‘ABCD‘
(CONCAT)
(IN2) : IN_TEXT2(STRING) = ‘DEF‘

Output(OUT) : OUT_TEXT(STRING) = ’ABCDDEF’

CONCAT_TIME

DATE and TOD concatenation

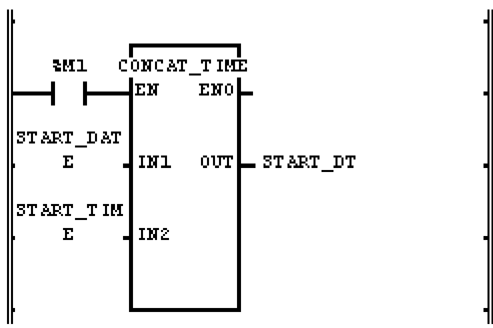
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Date data input IN2 : DOT data input **Output** ENO : Output EN value itself OUT : Output the date and DOT

■ **Function**

Concatenates IN1(DATE) and IN2(TOD) and outputs the resulting DT to OUT.

■ **Program example**

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

(1) If the execution condition(%M1) is On, the function CONCAT_TIME is executed.

(2) If the operation start data is START_DATE = D#1995-12-06 and operation start time is START_TIME = TOD#08:30:00, START_DT outputs DT#1995-12-06-08:30:00.

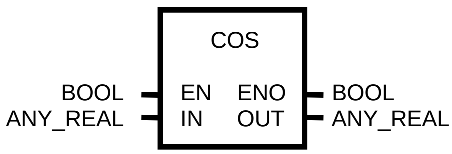
Input(IN1) : START_DATE(DATE) = D#1995-12-06
(CONCAT_TIME)
(IN2) : START_TIME(TOD) = TOD#08:30:00

Output(OUT) : START_DT(DT) = DT#1995-12-06-08:30:00

COS

Cosine operation

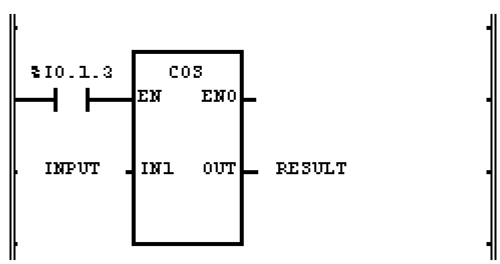
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Radian value of Cosine operation **Output** ENO : Output EN value itself OUT : Cosine result IN and OUT shall be same data type.

■ Function

Calculate IN's Cosine value and output the result to OUT.
OUT = COS (IN)

■ Program example

LD	IL
	LD %I0.1.3 JMPN CCC LD INPUT COS ST RESULT CCC :

(1) If the execution condition(%I0.1.3) is On, the function COS is executed.

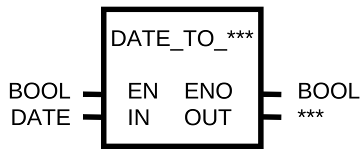
(2) If INPUT variable is 0.5235 (./6 rad = 30°), output variable RESULT will be 0.8660 .... ($\sqrt{3}$/2).
COS (./6) = $\sqrt{3}$/2 = 0.866

Input(IN1) : INPUT(REAL) = 0.5235
.(COS)
Output(OUT) : RESULT(REAL) = 8.66074800E-01

DATE_TO_***

DATE type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

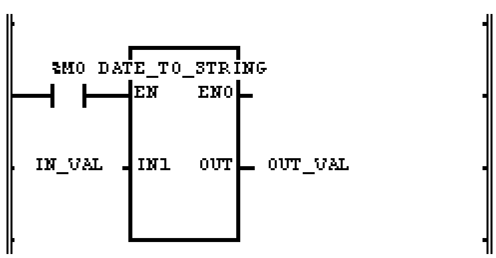
Function	Description
	**Input** EN : Execute the function in case of 1 IN : DATE data to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ Function

Convert IN to OUT data type.

FUNCTION	Output type	Description
DATE_TO_UINT	UINT	Convert DATE to UINT type.
DATE_TO_WORD	WORD	Convert DATE to WORD type.
DATE_TO_STRING	STRING	Convert DATE to STRING type.

■ Program example

LD	IL
	LD %M0 JMPN LL LD IN_VAL DATE_TO_STRING ST OUT_VAL LL :

(1) If the execution condition(%M0) is On, the function DATE_TO_STRING is executed.

(2) If INPUT variable IN_VAL(DATE type) isD#1995-12-01, output variable OUT_VAL (STRING type) will be 'D#1995-12-01'.

Input(IN1) : IN_VAL(DATE) = D#1995-12-01

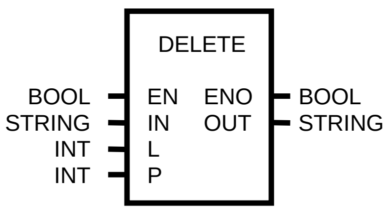
.(DATE_TO_STRING)

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

DELETE

Character string deletion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string input L : Character string length to be deleted P : Delete position of character string **Output** ENO : Output 1 in case of no error OUT : Character string output

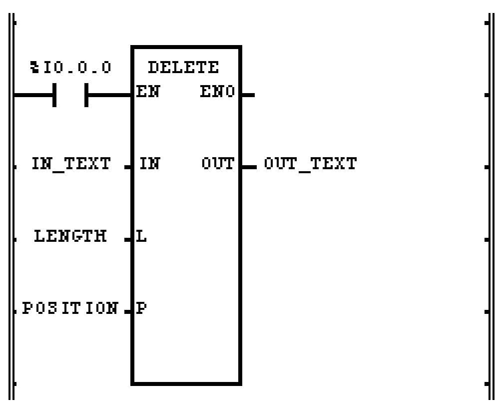
■ **Function**

After deleting L characters from P of Character string IN, output it to the character string OUT.

■ **Error**

If P.0 or L<0 or P> (Character number of IN1 input character string), _ERR,_LER flag is set.

■ **Program example**

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

(1) If the execution condition(%I0.0.0) is On, the character string deletion DELETE is executed.

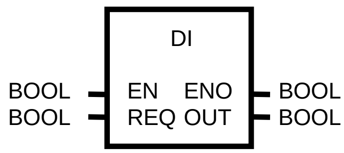
(2) If INPUT variable IN_TEXT(input character)='ABCDEF' and LENGTH(Character string length to be deleted)=3 and POSITION(Deletion position of character string)=3, output variable OUT_TEXT(STRING type) will be 'ABF'.

Input(IN) : IN_TEXT(STRING) = 'ABCDEF'
(L) : LENGTH(INT) = 3
(P) : POSITION(INT) = 3
.(DELETE)
Output(OUT) : OUT_VAL(STRING) = 'ABF'

DI

Prohibits task program operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

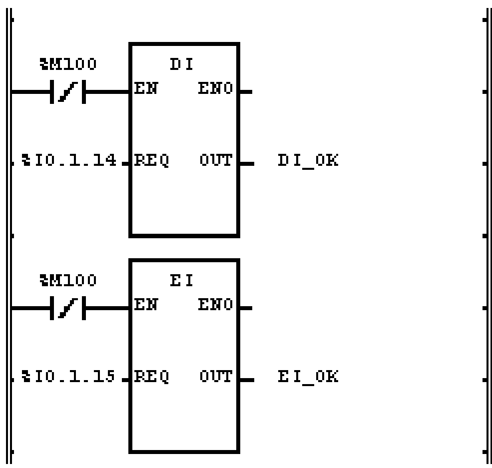
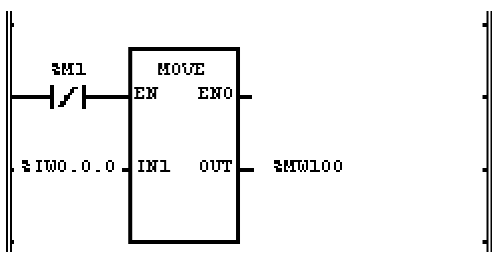
Function	Description
	**Input** EN : Execute the function in case of 1 REQ : Reguest to prohibit task program operation **Output** ENO : Output EN value itself OUT : Output 1 in case of DI execution

■ Function

- If EN is 1 and REQ has 1, prohibit the driving the task program(interval, interrupt) programmed by the user.
- If the normal task program operation is required. Please use 'EI' function.
- If the normal task program operation is required, please use 'EI' function.
- The task generated during task program operation is prohibited is executed as below.
  - – Interval task, interrupt : These are executed after EI' function execution or completion of current task program. But, if the task is generated more than twice, the task collision error (TASK_ERR) occurs and counts the collision time(TC_CNT)

■ **Program example**

Program to control task program increasing the value every second using task program driving prohibit function DI and task program driving allowance function E1

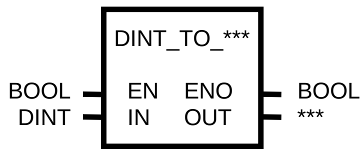
LD	IL
(1) Scan program(TASK program control)  	(1)Scan program controls(TASK program) LDN %M100 JMPN KK LD %I0.1.14 DI ST DI_OK KK : LDN %M100 JMPN LL LD %I0.1.15 EI ST EI_OK LL :
(2) Task program increasing the value every second  	(2)Task program increasing the value every second LDN %M1 JMPN MM LD %IW0.0.0 MOVE ST %MW100 MM :

(1) If REQ(Direct variable %I0.1.14) of DI driving prohibit request is On, the function DI is executed and DI_OK will be 1.
(2) When function DI is executed, the task program executing every second will be stopped.
(3) If REQ(Direct variable %I0.1.15) of EI driving prohibit request is On, the function EI is executed and EI_OK will be 1.
(4) When the function EI is executed, the task program stopped by function DI will be executed again.

DINT_TO_***

DINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Double Integer value to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ Function

Convert IN type and output it to OUT.

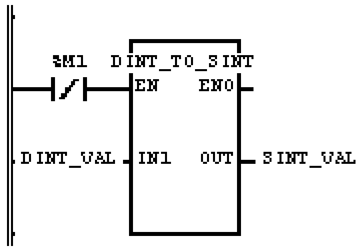
FUNCTION	Output type	Description
DINT_TO_SINT	SINT	If the input is -128.127, it is normally converted but the others cause the error.
DINT_TO_INT	INT	If the input is -32768.32767, it is normally converted but the others cause the error.
DINT_TO_LINT	LINT	Convert LINT type normally.
DINT_TO_USINT	USINT	If the input is 0.255, it is normally converted but the others cause the error.
DINT_TO_UINT	UINT	If the input is 0.65535, it is normally converted but the others cause the error.
DINT_TO_UDINT	UDINT	If the input is 0.2^{32}-1, it is normally converted but the others cause the error.
DINT_TO_ULINT	ULINT	If the input is 0.2^{32}-1, it is normally converted but the others cause the error.
DINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
DINT_TO_BYTE	BYTE	Convert lower 8 bit to BYTE type.
DINT_TO_WORD	WORD	Convert lower 16 bit to WORD type.
DINT_TO_DWORD	DWORD	Convert internal bit array to DWORD type without conversion.
DINT_TO_LWORD	LWORD	Fill upper bit with 0 to convert it to LWORD type.
DINT_TO_BCD	DWORD	If the input is 0.99,999,999, it is normally converted but the others cause the error.
DINT_TO_REAL	REAL	Convert DINT to REAL type. Conversion error rate is depend on precision.
DINT_TO_LREAL	LREAL	Convert DINT to LREAL type. Conversion error rate is depend on precision.

■ Error

When the conversion error occurs, _ERR, _LER flag is set.

Note When the error occurs, outputs internal bit array without the conversion by taking from lower bit as much as output type bit.

■ Program example

LD	IL
	LD %M1 JMPN LSB LD DINT_VAL DINT_TO_SINT ST SINT_VAL LSB :

(1) When the execution condition(%M1) is On, the data type conversion function DINT_TO_SINT is executed.
(2) If INI = DINT_VAL(DINT type) = -77, SINT_VAL(SINT type) = -77.

Input(IN1) : DINT_VAL(DINT) = -77

Upper	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Lower	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1

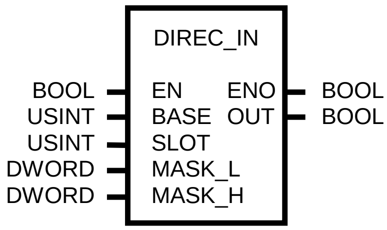
(DINT_TO_SINT)

Output(OUT) : OUT_VAL(SINT) = -77

1	0	1	1	0	0	1	1

DIREC_IN

Instant refresh of input data		Product	GM1	GM2	GM3	GM4	GM6
		Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 BASE : The base number of input module is located SLOT : The slot number of input module on the base MASK_L : Mask data for bits that never wanted to refresh among lower 32 bits input data MASK_H : Mask data for bits that never wanted to refresh among upper 32 bits input data **Output** ENO : Output 1 in case of no error OUT : Output 1 if the input data is completely refreshed.

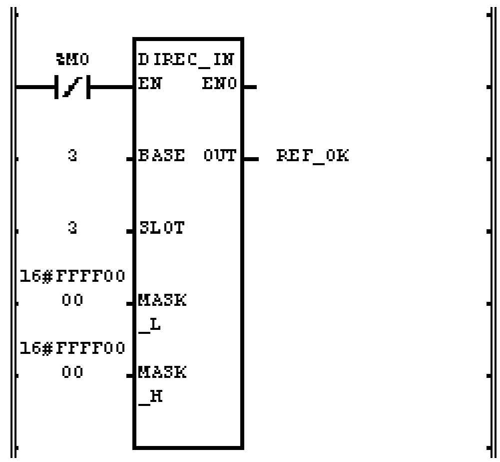
■ Function

- When EN of DIREC_IN is 1 during scanning, read 64 bit data of input module at allocated location of BASE and SLOT and refresh input image by this data.
- Refreshed image region is limited by contact points of input module installed at respective slot.
- Function DIREC_IN is available to change inputs(%I) On/Off status during scanning.
- As the scan synchronization batch processing processes input data reading and output data writing after completing scan program, the input data during 1Scan can not be refreshed. Function DIREC_IN can refreshes the relating input during executing the program.

■ **Program example**

1. Program that instartly refreshes lower 16 bits of assigned input image region when (16 point input module is at 4th slot of 4th base) and input data is 2#1010_1010_1110_1011.

LD



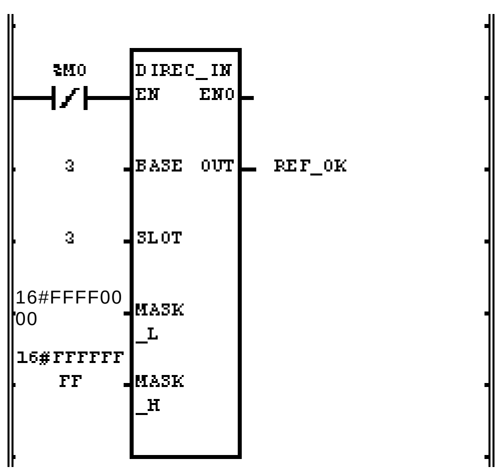
IL

```
LD                          %M0
JMPN                        ABC
LD                          3
DIREC_INBASE:=         CURRENT RESULT
            SLOT:=          3
            MASK_L:=        16#FFFF0000
            MASK_H:=        16#FFFF0000
ST                     REF_OK
ABC :
```

(1) When the input condition(%M0) is On, DIREC_IN function is executed.
(2) As the installed module is 16 point module, update image region will be %IW3.3.0 and as lower 16Bit is set to refresh in MASK_L(input lower 32Bit), %IW3.3.0 is updated to #1010_1010_1110_1011 during DIREC_IN execution.
(3) MASK_H(input upper 32Bit) value is ignored since 16 point module is installed at assigned slot

2. Program that instartly refreshes lower 16 bit of input image when 32 points (16 point input module is at 4th slot of 4th base) 2#0000_0000_1111_1111_1100_1100_0011_0011.

LD

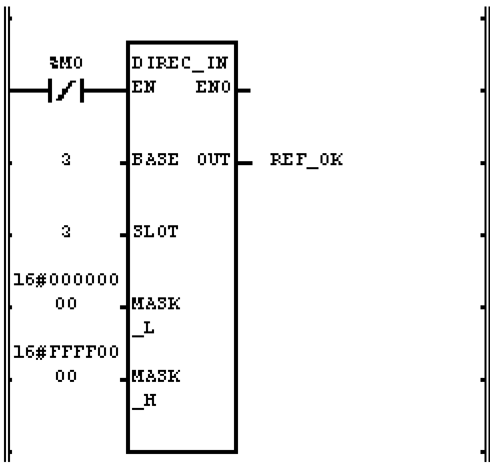


IL

```
LD                          %M0
JMPN                        ABC
LD                          3
DIREC_INBASE:=         CURRENT RESULT
            SLOT:=          3
            MASK_L:=        16#00000000
            MASK_H:=        16#FFFFFFFF
ST                     REF_OK
ABC :
```

(1) If the input condition(%M0) is on, DIREC_IN function is executed.
(2) As the installed module is 32 point, the refreshed image region is %ID3.3.0 but as lower 16Bits of MASK_L(input lower 32Bit) is allowed to update, %IW3.3.0 is refreshed to 2#1100_1100_0011_0011.

3. Program that updates lower 48Bits of 64Bits input image promptly when 64 point module is at 4th slot of 4th base and input data is 16#0000_FFFF_AAAA_7777 (2#0000_0000_0000_1111_1111_1111_1111_1010_ 1010_1010_ 1010_0111_0111_ 0111_0111).

LD	IL
	LD %M0 JMPN ABC LD 3 DIREC_INBASE:= CURRENT RESULT SLOT:= 3 MASK_L:= 16#00000000 MASK_H:= 16#FFFF0000 ST REF_OK ABC :

(1) If input condition(#M0) is on, DIREC_IN(input data prompt update) function is executed.

(2) As the installed module is 64 point, update image region will be %IL3.3.0, i.e., %ID3.3.0 and %ID3.3.1.
As all lower 32Bit(MASK_L) is allowed to update, %ID3.3.0 will be updated.
As lower 16Bits of upper 32Bit(MASK_H) is allowed to update, %IW.3.3.2 is updated but %IW3.3.3 is not updated.
Therefore, the data update of image region is as below.

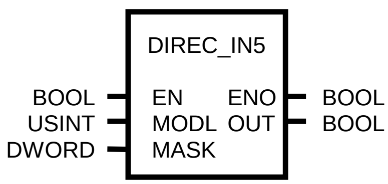
%IL3.3.0
- %ID3.3.0
  - %IW.3.3.0:2#0111_0111_0111_0111
  - %IW.3.3.1:2#1010_1010_1010_1010
- %ID3.3.1
  - %IW3.3.2:2#1111_1111_1111_1111
  - %IW3.3.3: maintain previous value

(3) When the input refresh is completed, REF_OK(input data refresh completion) outputs 1.

DIREC_IN5

Input data prompt update

Product	GM1	GM2	GM3	GM4	GM6
Applicable					

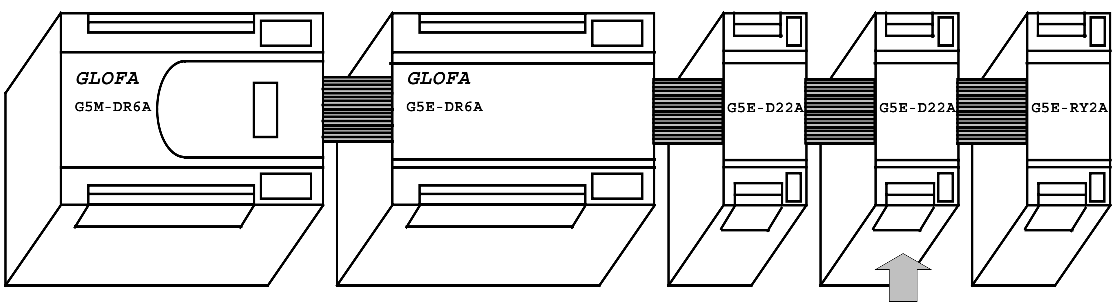
Function	Description
	**Input** EN : Execute the function in case of 1 MODL : Location number of input module MASK : Bit assignment if input lower 32Bit data is not updated **Output** ENO : Output 1 in case of no error OUT : Output 1 if input data is completely updated

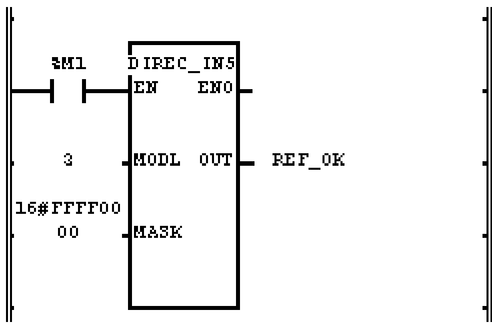
■ Function

- EN of DIREC_IN5 is 1 during scanning, read data of input module at allocated location and update it to input image region.
- Refreshed image region is limited by contact points of input module installed at respective slot.
- Function DIREC_IN5 is available to change input(%I) On/Off status during scanning.
- As the scan synchronization batch processing processes input data reading and output data writing after completing scan program, the input data from outside during 1Scan can not be updated. Function DIREC_IN5 can updates the relating input during executing the program.

■ Program example

Program that promptly refreshes lower 16bits of assigned input image when 4th module's input data is 2#1010_1010_1010_1010.



<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M1
JMPN AAA
LD 3
DIREC_IN5 MODL:= CURRENT RESULT
MASK:= 16#FFFF0000
ST
AAA :</td></tr>
</table>

(1) As the installation position is third expansion, set the location number MODL of input module to 3.

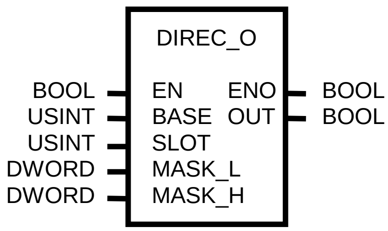
(2) As the input module is 16 point, lower 16Bit of MASK value is allowed to refresh.(16#FFFF 0000)

(3) If the execution condition(%M1) is On, DIREC_IN5(input data prompt upgrade) is executed and input data of module is updated promptly.

DIREC_O

Instant refresh of output data	

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

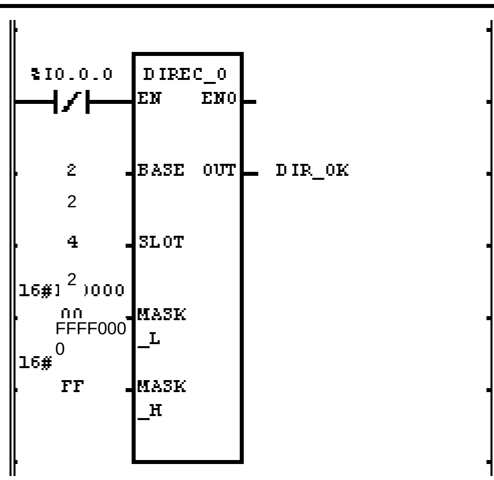
Function	Description
	**Input** EN : Execute the function in case of 1 BASE : The base number of input module is located SLOT : The slot number of input module on the base MASK_L : Mask data for bits that never wanted to refresh among lower 32 bits input data MASK_H : Mask data for bits that never wanted to refresh among upper 32 bits input data **Output** ENO : Output 1 in case of no error OUT : Output 1 if output data is completely updated

■ Function

- When EN of DIREC_O(input data prompt update) is 1 during scanning, read 64bits data from assigned output image and outputs these bits instantly, but masked bits by mask data (MASK_L, MASK_H) are not refreshed. '1' means masked.
- Function DIREC_0 is available to change output(%Q) On/Off status during scanning.
- As the batch processing processes the input data reading and output data writing after completing scan program, the data refresh to output module during scan is not possible.
- Function DIREC_O can output the bit data during scan.
- If different type module is inserted or data is not written to output module, output EN0 and OUT to 0.(Normal operation : output 1)

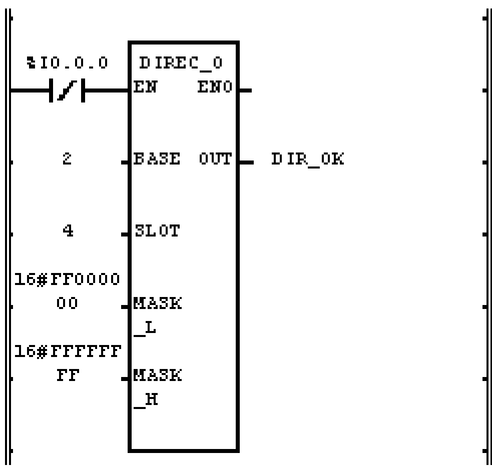
■ Program example

1. Program that instantly outputs the output data of 2#0111_0111_0111_0111 to 16 point relay output module installed at third slot of 3rd base.

LD	IL
	LD %I0.0.0 JMPN AAA LD 2 DIREC_O BASE: = CURRENT RESULT SLOT:= 2 MASK_L:= 16#FFFF0000 MASK_H:= 16#FFFFFFFF ST REF_OK AAA :

(1) Input BASE number 2 and SLOT number 4.
(2) As the data wanted to output is 16bits, set lower 16Bit of MASK_L to enable output.(16#FFFF0000)
(3) If the execution condition(%I0.0.0) is On, DIREC_O function is executed and the output module has 2#0111_ 0111_0111_0111 during scanning.

2. Program that instantly outputs the lower 24Bit of 32 point TR installed at 5th slot of 3rd base, output data is 2#1111_0000_1111_0000_1111_0000.

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>

```
LD                    %I0.0.0
JMPN                  AAA
LD                    2
DIREC_O   BASE: =     CURRENT RESULT
          SLOT:=      4
          MASK_L:=    16#FF000000
          MASK_H:=    16#FFFFFFFF
ST                  REF_OK
AAA :
```

</td></tr>
</table>

(1) Input BASE number 2 and SLOT number 4.
(2) As the data wanted to output is 24bits, lower 24Bit of MASK_L value is allowed to output.(16# FF000000)
(3) If the execution condition(%I0.0.0) is off, function DIREC_0 is executed and output module has 2#...._...._1111_0000_1111_0000_1111_0000.

Previous value

DIREC_O5

Instant refresh of output data	

Product	GM1	GM2	GM3	GM4	GM6
Applicable					

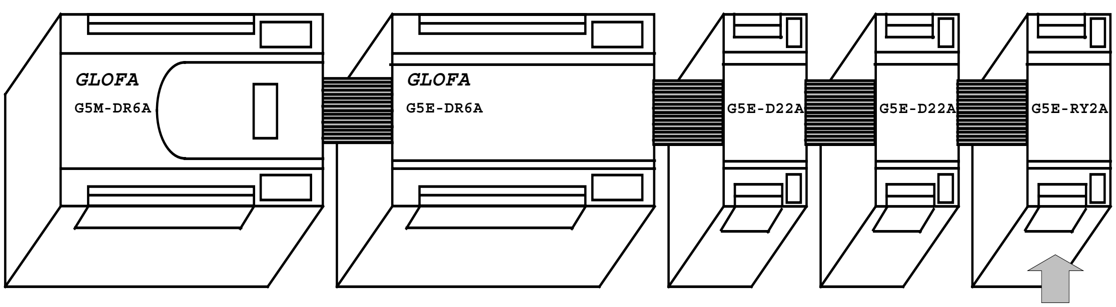
Function	Description
DIREC_O5 BOOL — EN ENO — BOOL USINT — MODL OUT — BOOL DWORD — MASK	**Input** EN : Execute the function in case of 1 MODL : Position number of output module MASK : Bit assignment not to be updated among output lower 32Bit data **Output** ENO : Output 1 in case of no error OUT : Output 1 if the output data is updated

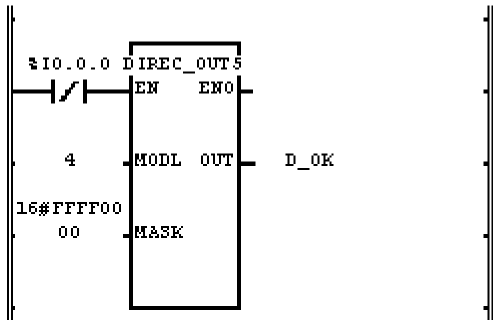
■ Function

- When EN of DIREC_05(input data prompt update) is 1 during scanning, read 64bits data from assigned output image and outputs these bits instantly, but masked bits by mask data (MASK_L, MASK_M) are not refreshed. '1' means masked.
- Function DIREC_05 is available to change output(%Q) On/Off status during 1scanning.
- As the batch processing processes the input data reading and output data writing after completing scan program, the data refresh to output module during scan is not possible.
- Function DIREC_O5 can output the bit data to outside during scan.
- If different type module is inserted or data is not written normally to output module, output EN0 and OUT to 0.(Normal operation : output 1)

■ Program example

Program that fifth installed output module outputs the output data of 2#1111_0000_1111_0000 under below GM5 system configuration.



<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %I0.0.0
JMPN CCC
LD 4
DIREC_O5 MODL:= CURRENT RESULT
MASK:= 16#FFFF0000
ST D_OK
CCC :</td></tr>
</table>

(1) As the output module is located at expansion fifth, input the location number of output module with 4.

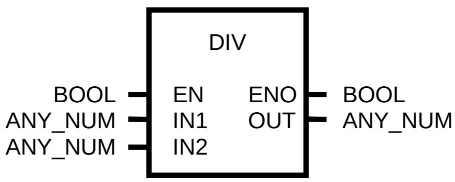
(2) As the module point to output the data during scanning is 16, set MASK value to update allowance of lower 16Bit only.(16#$\underbrace{\text{FFFF}}_{\text{Update prohibit}}\ \underbrace{\text{0000}}_{\text{Update allowance}}$)

(3) If the execution condition(%I0.0.0) is OFF, DIREC_05 is executed and fifth expanded output module has refreshed data during scan.

DIV

Divide

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Dividend IN2 : Divisor **Output** ENO : Output 1 in case of no error OUT : Result(Quotient) Variable connected to IN1, IN2 and OUT shall be same data type.

■ **Function**

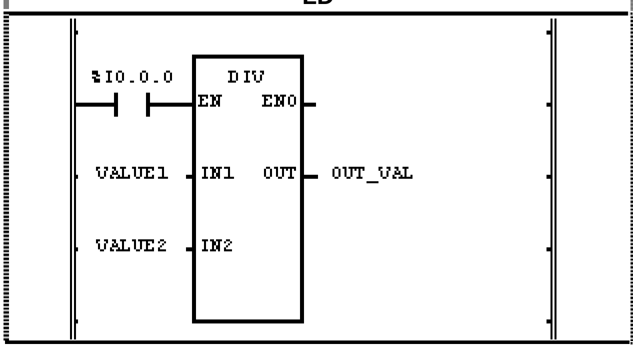
Divide IN1 by IN2 and output the quotient excluding the value below decimal point to OUT. OUT = IN1/IN2

IN1	IN2	OUT	Remark
7 7 -7 -7	2 -2 2 -2	3 -3 -3 3	Delete the value below decimal point
7	0	x	Error

■ **Error**

If the divisor is '0', _ERR and _LER flag is set.

■ **Program example**

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

(1) If the execution condition(%I0.0.0) is On, division function DIV is executed.

(2) If input variable VALUE1 = 300 and VALUE2 = 100, output OUT_VAL = 300/100 = 3.

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

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	1	1	0	0	1	0	0

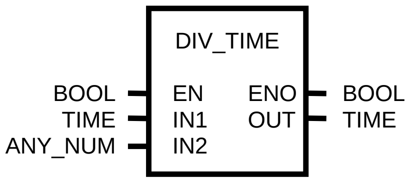
Output(OUT) : OUT_VAL(INT) = 3(16#3)

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

DIV_TIME

Time divide

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Dividing time IN2 : Dividing value **Output** ENO : Output 1 in case of no error OUT : Result

■ **Function**

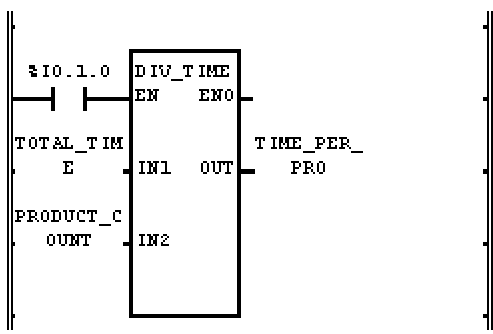
Divide IN1(time) by IN2(number) and output the result to OUT.

■ **Error**

If divisor(IN2) is 0, _ERR and _LER flag is set.

■ **Program example**

Program that calculates the time to manufacture one product if one-day work time is 12 hours 24 minutes 24 seconds and one-day production capacity is 12 products.

LD	IL
	LD %I0.0.0 JMPN SS LD TOTAL_TIME DIV_TIME IN1:= CURRENT RESULT IN2:= PRODUCT_COUNT ST TIME_PER_PRO SS :

(1) If the execution condition(%I0.1.0) is On, the time division function DIV_TIME is executed.

(2) Dividing TOTAL_TIME(T#12H24M24S) by PRODUCT_COUNT(12), TIME_PER_PRO(T#1H2M2S), 1 hour 2 minutes 2 seconds, is output.

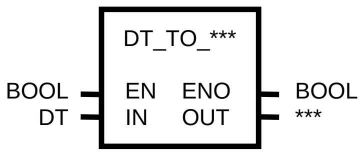
Input(IN1) : TOTAL_TIME(TIME) = T#12H24M24S
/ (DIV_TIME)
(IN2) : PRODUCT_COUNT(INT) = 12

Output(OUT) : TIME_PER_PRO(TIME) = T#1H2M2S

DT_TO_***

DT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

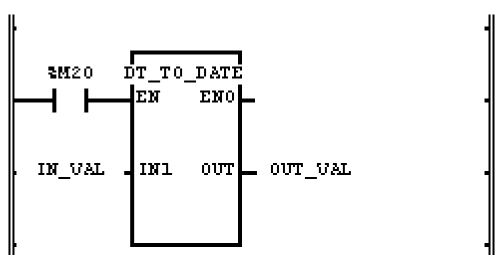
Function	Description
	**Input** EN : Execute the function in case of 1 IN : DATE_AND_TIME data to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

FUNCTION	Output type	Description
DT_TO_LWORD	LWORD	Convert DT to LWORD type. (Reverse conversion is available since inter data is not converted.)
DT_TO_DATE	DATE	Convert DT to DATE type.
DT_TO_TOD	TOD	Convert DT to TOD type.
DT_TO_STRING	STRING	Convert DT to STRING type.

■ **Program example**

LD	IL
	LD %M20 JMPN L LD IN_VAL DT_TO_DATE ST OUT_VAL L :

(1) If the execution condition(%M20) is On, DT type conversion function DT_TO_DATE is executed.
(2) If IN_VAL(DT type) = DT#1995-12-01-12:00:00, OUT_VAL(DATE type) = D#1995-12-01.

Input(IN1) : IN_VAL(DT) = DT#1995-12-01-12:00:00

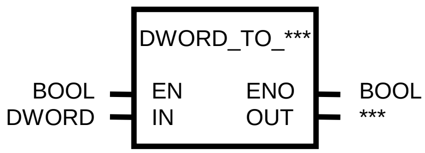
(DT_TO_DATE)

Output(OUT) : OUT_VA(DATE) = D#1995-12-01

DWORD_TO_***

DWORD type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

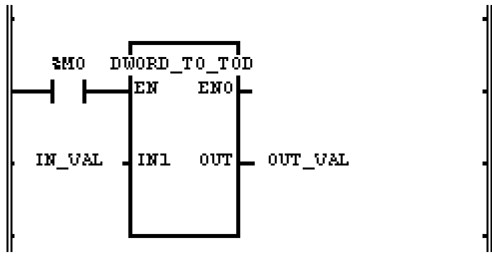
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit array to be converted(32Bit) **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

FUNCTION	Output type	Description
DWORD _TO_SINT	SINT	Convert lower 8Bit to SINT type.
DWORD _TO_INT	INT	Convert lower 16Bit to INT type.
DWORD _TO_DINT	DINT	Convert internal bit array to DINT type without conversion.
DWORD _TO_LINT	LINT	Fill upper bit with 0 to convert it to LINT type.
DWORD _TO_USINT	USINT	Convert lower 8Bit to USINT type.
DWORD _TO_UINT	UINT	Convert lower 16Bit to UINT type.
DWORD _TO_UDINT	UDINT	Convert internal bit array to UDINT type without conversion.
DWORD _TO_ULINT	ULINT	Fill upper bit with 0 to convert it to ULINT type.
DWORD _TO_BOOL	BOOL	Convert lower 1Bit to BOOL type.
DWORD _TO_BYTE	BYTE	Convert lower 8Bit to BYTE type.
DWORD _TO_WORD	WORD	Convert lower 16Bit to WORD type.
DWORD _TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
DWORD _TO_REAL	REAL	Convert internal bit array to REAL type without conversion.
DWORD _TO_TIME	TIME	Convert internal bit array to TIME type without conversion.
DWORD _TO_TOD	TOD	Convert internal bit array to TOD type without conversion.
DWORD _TO_STRING	STRING	Convert input value to decimal and STRING type.

■ **Program example**

LD	IL
	LD %M0 JMPN AA LD IN_VAL DWORD_TO_WORD ST OUT_VAL AA :

(1) If the execution condition(%M0) is On, type conversion function DWORD_TO_TOD is executed.
(2) If IN_VAL(DWORD type) = 16#3E8(1000), OUT_VAL(TOD type) = TOD#1S.

Input(IN1) : IN_VAL(DWORD) = 16#3E8(1000)

Upper	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lower	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

Data type is converted without the data change (internal bit array status)

Output(OUT) : OUT_VAL(TOD) = TOD#1S

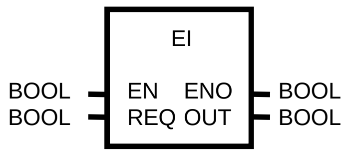
Upper	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lower	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

Note TIME and TOD calculates decimal value by ms unit, i.e., 1000 is calculated as 1000ms = 1s. (Refer to 3.2.4. Data type structure)

EI

Task program operation allow

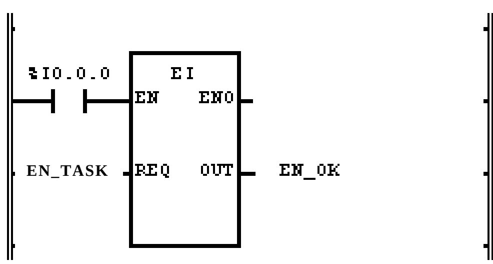
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 REQ : Task program driving allowance request **Output** ENO : Output EN value itself OUT : Output 1 if EI is executed

■ Function

- If EN is 1 and REQ has 1, task program blocked by 'DI' function is operated normally.
- Once 'EI' command is executed, the task program is operated normally though REQ input is 0.
- Tasks generated at the driving prohibit status of task program is executed after executing 'EI' function execution or completing current task program.

■ Program example(Refer to DI)

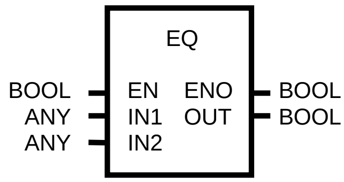
LD	IL
	LD %I0.0.0 JMPN LSB LD EN_TASK EI ST EN_OK LSB :

If EN_TASK is 1, the task program is normally operated.
When the task execution is allowed by 'EI' function, EN_OK outputs 1.

EQ

'Equal' comparison

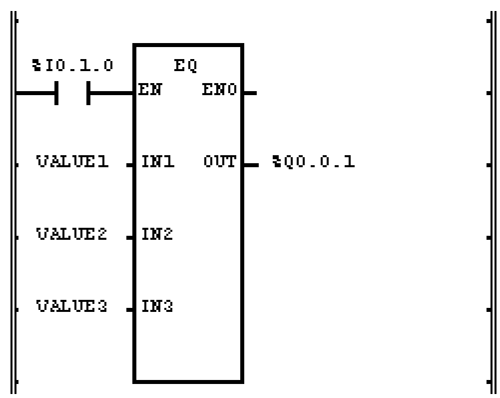
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be compared IN2 : Comparing value Can be extended to 8 inputs. IN1, IN2, ... shall be same type. **Output** ENO : Output EN value itself OUT : Comparison result

■ **Function**

If IN1=IN2=IN3...=INn(n: input number), OUT outputs 1.
Otherwise, OUT outputs 0.

■ **Program example**

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

(1) If the execution condition(%I0.1.0) is On, comparison function 'EQ' is executed.
(2) If VALUE1 = 300 and VALUE2 = 300 and VALUE3 = 300, output %Q0.0.1 = 1 since comparison result VALUE1 = VALUE2 = VALUE3.

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

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

= (EQ)

(IN2) : VALUE2(INT) = 300(16#012C)

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

= (EQ)

(IN3) : VALUE3(INT) = 300(16#012C)

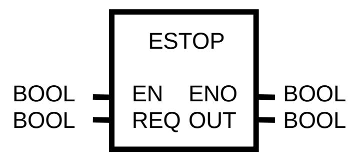
0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0

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

1

ESTOP

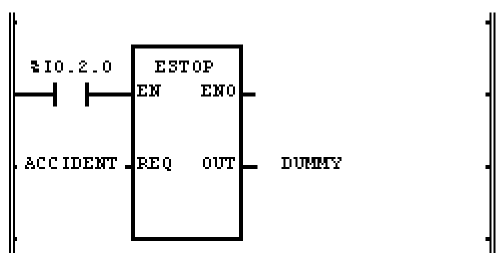
Emergency stop by program	Product	GM1	GM2	GM3	GM4	GM6
	Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 REQ : Emergency stop request **Output** ENO : Output EN value itself OUT : Output 1 if ESTOP is executed

■ **Function**

. If the function execution condition EN is 1 and emergency stop request signal REQ is 1, stop current executing program promptly and go to STOP mode.
. In case of stop by 'ESTOP' function, the operation is not available though the power is supplied again.
. Set the operation mode to STOP and from STOP to RUN, the operation starts again.
. Since 'ESTOP' function stops the program, there may be error of data continuity if not cold restart mode during restarting.

■ **Program example**

LD	IL	
	LD JMPN LD ESTOP (ST SSS :	%I0.2.0 SSS ACCIDENT DUMMY)

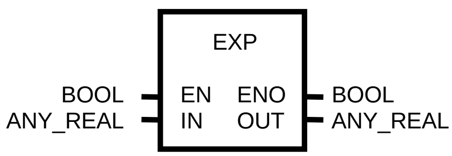
(1) If the execution condition(%I0.2.0) is On, 'ESTOP' function is executed.
(2) If ACCIDENT is 1, the running program stops promptly and go to STOP mode.

Note In case of emergency situation, ESTOP function can be used as redundancy safety device with mechanical interrupt.

EXP

Natural exponent operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

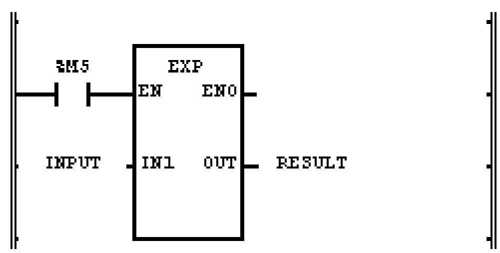
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of exponent operation **Output** ENO : Output EN value itself OUT : Exponent operation result IN and OUT shall be same data type.

■ **Function**

Calculate INs exponent value and output it to OUT.

OUT = e^{IN}

■ **Program example**

LD	IL
	LD %M5 JMPN JJ LD INPUT EXP ST RESULT JJ :

(1) If the execution condition(%M5) is On, natural exponent function 'EXP' is executed.

(2) If input variable INPUT is 2.0, output variable RESULT will be 7.3890 ..... $e^{2.0}$ = 7.3890.....

Input(IN1) : INPUT(REAL) = 2.0

Upper	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lower	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

(16#40000000)

(EXP)

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

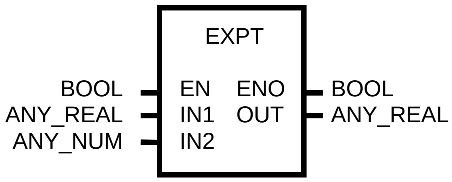
Upper	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Lower	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

(16#40EC7326)

EXPT

Exponent operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Real IN2 : Exponent **Output** ENO : Output 1 in case of no error OUT : Result Variable connected to IN1 and OUT shall be same data type.

■ **Function**

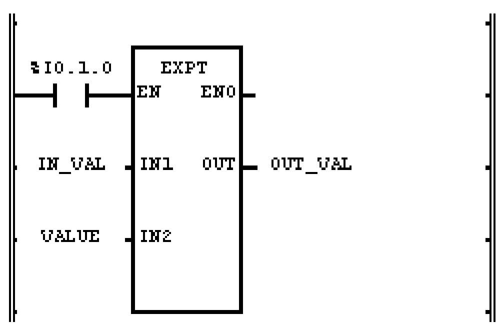
Exponent IN1 by IN2 and output it to OUT.

$OUT = INI^{IN2}$

■ **Error**

If the output exceeds the range of related data type, _ERR and _LER flag is set.

■ **Program example**

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

(1) If the execution condition(%I0.1.0) is On, natural exponent function 'EXPT' is executed.

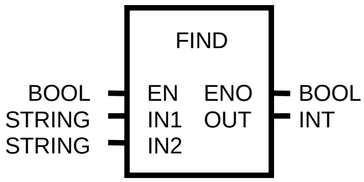
(2) If IN_VAL = 1.5 and VALUE = 3, OUT_VAL = 1.53 = 1.5x1.5x 1.5 = 3.375.

Input(IN1) : IN_VAL(REAL) = 1.5
(IN2) : VALUE(INT) = 3
.(EXPT)
Output(OUT) : OUT_VAL(REAL) = 3.37500000E+00

FIND

Find character string

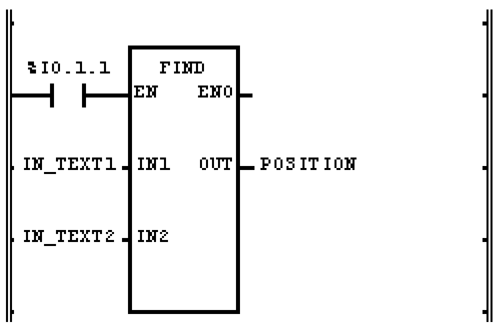
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execution the function in case of 1 IN1 : Character string input IN2 : Character to be found **Output** ENO : Output EN value itself OUT : Location of found character string

■ **Function**

Find character string IN2 in input character string IN1. If find, output the character location of IN2 in IN1 to OUT and if not, output 0 to OUT.

■ **Program example**

LD	IL
	LD %I0.1.1 JMPN XYZ LD IN_TEXT1 FIND IN1:= CURRENT RESULT IN2:= IN_TEXT2 ST POSITION XYZ :

(1) If the execution condition(%I0.1.1) is On, execute FIND(character string find) function.

(2) If IN_TEXT1=‘ABCEF and IN_TEXT2=‘BC‘, output variable POSITION=2 is declared. (The location of IN_TEXT2=‘BC‘ in input character string IN_TEXT1=‘ABCEF‘ is second)

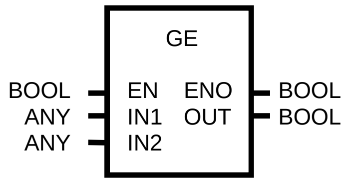
Input(IN1) : IN_TEXT1(STRING) = ’ABCEF’
(FIND)
(IN2) : IN_TEXT2(STRING) = ’BC’

Output(OUT) : POSITION(INT) = 2

GE

'Greater than or equal' comparison

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

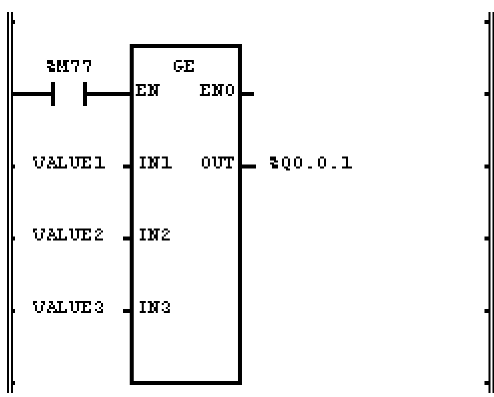
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td></td><td>Input EN : Execute the function in case of 1
IN1 : Value to be compared
IN2 : Comparing value
Can be extended to 8 inputs.
IN1, IN2, ... shall be same type.

Output ENO : Output EN value itself
OUT : Comparison result</td></tr>
</table>

■ Function

If IN1.IN2.IN3... .INn(n: Input number), OUT outputs 1.
Otherwise, OUT outputs 0.

■ Program example

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

(1) If the execution condition(%M77) is On, GE(comparison: larger or equal) function is executed.
(2) If input variable VALUE1=300 VALUE2=200 and VALUE3=100, output result %Q0.01 will be 1 since comparison result VALUE1.VALUE2.VALUE3.

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

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	1	1	0	0	1	0	0	0

. (GE)

(IN3) : VALUE3(INT) = 100(16#0064)

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

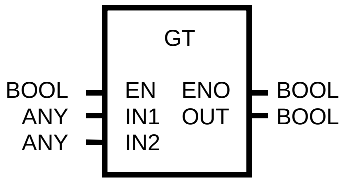
.

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

GT

'Greater than' comparison

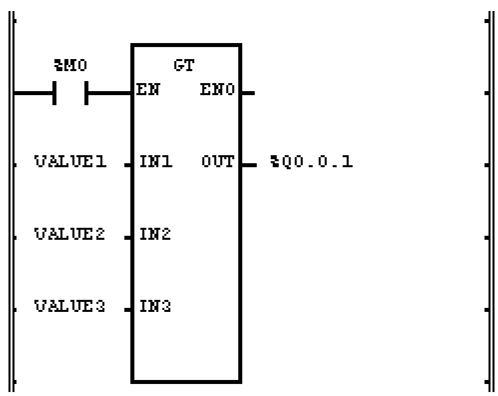
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be compared IN2 : Comparing value Can be extended to 8 inputs. IN1, IN2, ... shall be same type. **Output** ENO : Output EN value itself OUT : Comparison result

■ **Function**

If IN1>IN2>IN3...>INn(n: input number), OUT outputs 1.
Otherwise, OUT outputs 0.

■ **Program example**

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

(1) If the execution condition(%M0) is On, T(Comparison: larger) function is executed.
(2) If input variable VALUE1 = 300, VALUE2 = 200 and VALUE3 = 100, output result %Q0.0.1 will be 1 since comparison result VALUE1 > VALUE2 > VALUE3.

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

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

> (GT)

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

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

> (GT)

(IN3) : VALUE3(INT) = 100(16#0064)

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

INSERT

Character string insertion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
INSERT BOOL — EN ENO — BOOL STRING — IN1 OUT — STRING STRING — IN2 INT — P	**Input** EN : Execute the function in case of 1 IN1 : Character string to be added IN2 : Adding character string P : Character string insert position **Output** ENO : Output 1 in case of no error OUT : Output character string

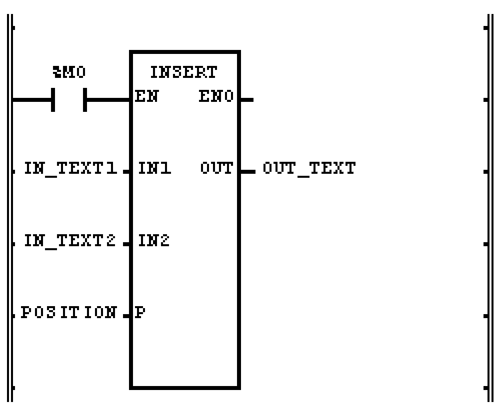
■ **Function**

Insert character string IN2 to position of P-th character of IN1 and output it to OUT.

■ **Error**

If P.0 or (character number of variable IN1) < P or if the character number of result exceeds 30, _ERR and _LER flag is set and just 30 characters are output to OUT.

■ **Program example**

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

(1) If the execution(%M0) is On, INSERT(character string insert) function is executed.

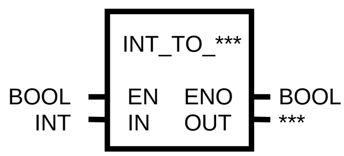
(2) If input variable IN_TEXT1= ‘ABCD‘ and IN_TEXT2=‘XY‘ and POSITON=2, output variable OUT_TEXT= ‘ABXYCD‘.

Input(IN1) : IN_TEXT1(STRING) = ’ABCD’
(IN2) : IN_TEXT2(STRING) = ’XY’
(P) : POSITION(INT) = 2
.(INSERT)
Output(OUT) : OUT_TEXT = ‘ABXYCD‘

INT_TO_***

INT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Integer to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ Function

Convert IN to OUT data type.

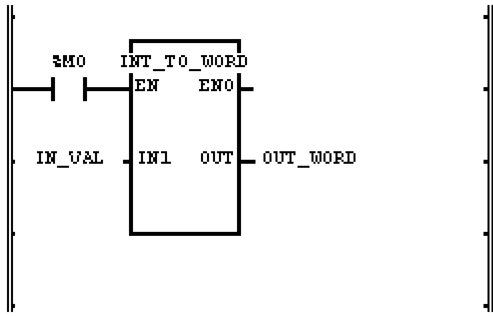
FUNCTION	Output type	Description
INT_TO_SINT	SINT	If input is -128.127, convert integer normally and for other value, the error occurs.
INT_TO_DINT	DINT	Convert to DINT type normally.
INT_TO_LINT	LINT	Convert to LINT type normally.
INT_TO_USINT	USINT	If input is 0.255, convert integer normally and for other value, the error occurs.
INT_TO_UINT	UINT	If input is 0.32767, convert integer normally and for other value, the error occurs.
INT_TO_UDINT	UDINT	If input is 0.32767, convert integer normally and for other value, the error occurs.
INT_TO_ULINT	ULINT	If input is 0.32767, convert integer normally and for other value, the error occurs.
INT_TO_BOOL	BOOL	Convert lower 1Bit to BOOL type.
INT_TO_BYTE	BYTE	Convert lower 8Bit to BYTE type.
INT_TO_WORD	WORD	Convert internal bit array to WORD type without conversion.
INT_TO_DWORD	DWORD	Convert upper bit filled with 0 to DWORD type.
INT_TO_LWORD	LWORD	Convert upper bit filled with 0 to LWORD type.
INT_TO_BCD	WORD	If input is 0.9,999, convert integer normally and for other value, the error occurs.
INT_TO_REAL	REAL	Convert INT to REAL type normally.
INT_TO_LREAL	LREAL	Convert INT to LREAL type normally.

■ Error

When conversion error occurs, _ERR _LER flag is set.

Note When error occurs, outputs bits from lower bit of IN as much as output type bit number without conversion of internal bit array.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD IN_VAL INT_TO_WORD ST OUT_WORD AAA :

(1) If the execution(%M0) is On, INT_TO_WORD function is executed.
(2) If input variable IN_VAL(INT type) = 512(16#200), output variable OUT_WORD(WORD type) = 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

.(INT_TO_WORD)

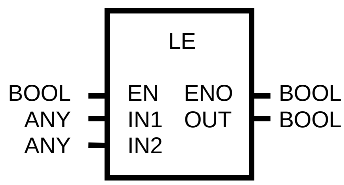
Output(OUT) : OUT_WORD(WORD) = 16#200

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

LE

'Less than or equal' comparison

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

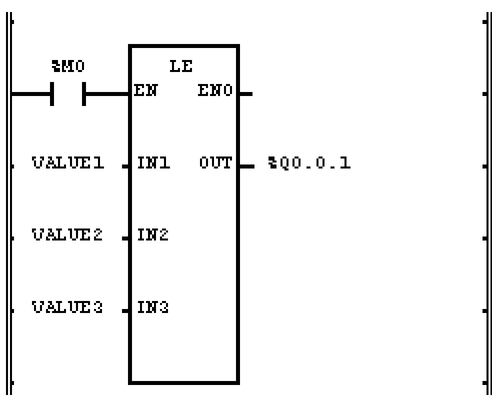
Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be compared IN2 : Comparing value Can be extended to 8 inputs. IN1, IN2, ... shall be same type. **Output** ENO : Output EN value itself OUT : Comparison result

■ **Function**

If IN1.IN2.IN3....INn (n: input number), OUT outputs 1.
Otherwise, OUT outputs 0.

■ **Program example**

LD



IL

```
LD                  %M0
JMPN                BBB
LD                  VALUE1
LE       IN1:=      CURRENT RESULT
         IN2:=      VALUE2
         IN3:=      VALUE3
ST                 %Q0.0.1
BBB :
```

(1) If the execution(%M0) is On, LE(Comparison: less or equal) function is executed.
(2) If input variable VALUE1=150 and VALUE2=200 and VALUE3 = 250, output result %Q0.0.1 will be 1.

Input(IN1) : VALUE1(INT) = 150(16#0096)

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

. (LE)

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

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

. (LE)

(IN3) : VALUE1(INT) = 250(16#00FA)

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

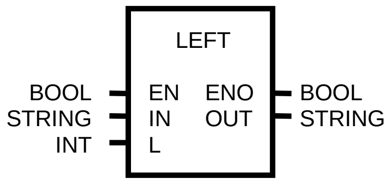
Output(OUT) : %Q0.0.1(BOOL) = 1(16#1)

1

LEFT

Left of character string

Product	GM1	GM2	GM3	GM4	GM6
Applicable	·	·	·	·	·

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string input L : Character string length to be output **Output** ENO : Output 1 in case of no error OUT : Character string output

■ Function

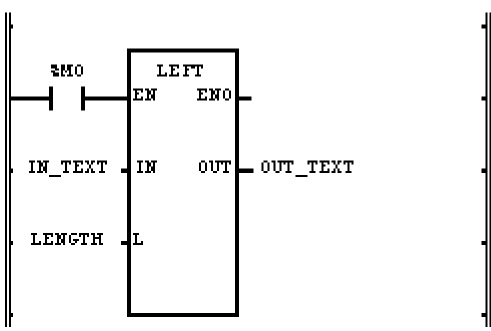
Outputs L characters from left character of In to OUT.

■ Error

If L < 0, _ERR and _LER flag is set.

■ Program example

LD



IL

```
LD                      %M0
JMPN                    FF
LD                      IN_TEXT
LEFT        IN:=        CURRENT RESULT
            L:=         LENGTH
ST                      OUT_TEXT
FF:
```

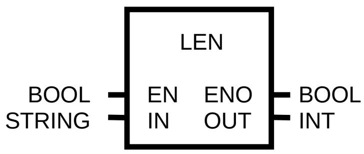
(1) If the execution condition(%M0) is On, LEFT(get left of character string) function is executed.
(2) If IN_TEXT=‘ABCDEFG‘ and LENGTH=3, output character string variable OUT_TEXT=‘ABC‘.

Input(IN1) : IN_TEXT(STRING) = ’ABCDEFG’
(IN2) : LENGTH(INT) = 3
.(LEFT)
Output(OUT) : OUT_TEXT(STRING) = ‘ABC‘

LEN

Character string length

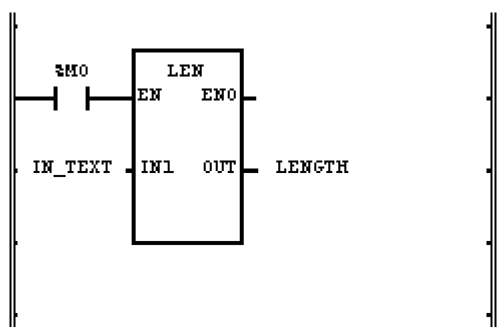
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string input **Output** ENO : Output EN value itself OUT : Character string length

■ **Function**

Output the length of input character string(IN) to OUT.

■ **Program example**

LD	IL
	LD %M0 JMPN II LD IN_TEXT LEN IN:= CURRENT RESULT ST LENGTH II:

(1) If the execution condition(%M0) is On, LEN(character string length) function is executed.
(2) If input variable IN_TEXT=‘ABCD‘, output length LENGTH=4.

Input(IN1) : IN_TEXT(STRING) = ’ABCD’

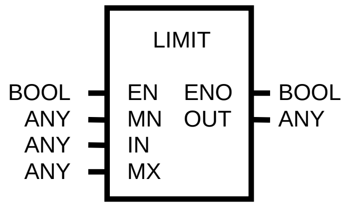
.(LEN)

Output(OUT) : LENGTH(INT) = 4

LIMIT

Upper/Lower limit

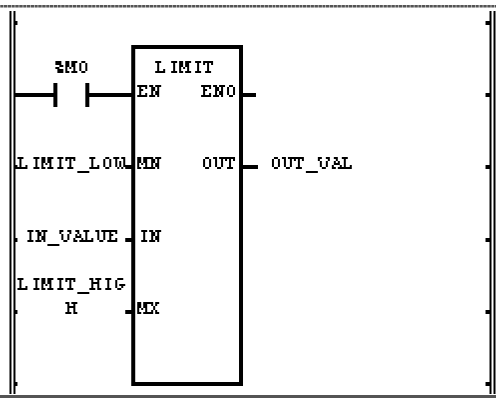
Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 MN: Minimum value IN : Limit value MX : Maximum value **Output** ENO : Output EN value itself OUT : Value in the range MN, IN, MX and OUT shall be same data type.

■ **Function**

. If input value IN is between MN and MX, OUT outputs IN. Therefore, if MN ≤ IN ≤ MX, OUT = IN
. If input value IN is less than MN, OUT outputs MN. Therefore, if IN < MN, OUT = MN
. If input value IN is greater than MX, OUT outputs MX. Therefore, if IN > MX, OUT = MX.

■ **Program example**

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

(1) If the execution condition(%M0) is On, LIMIT(upper/lower limit) function is executed.
(2) The output variable(OUT_VAL) on lower limit input variable(LIMIT_LOW), upper limit input variable(LIMIT_HIGH) and limited value input variable(IN_VALUE) is as below.

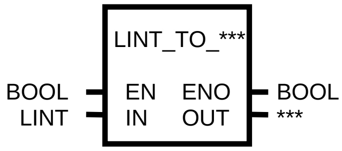
LIMIT_LOW	IN_VALUE	LIMIT_HIGH	OUT_VAL
1000	2000	3000	2000
1000	500	3000	1000
1000	4000	3000	3000

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

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Long Integer to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

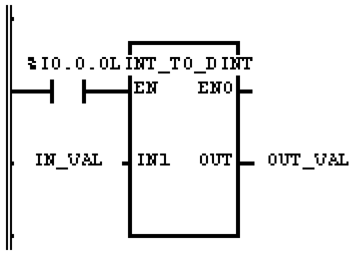
FUNCTION	Output type	Description
LINT_TO_SINT	SINT	If input is -128.127, convert it normally. Otherwise, the error occurs.
LINT_TO_INT	INT	If input is -32,768.32,767, convert it normally. Otherwise, the error occurs.
LINT_TO_DINT	DINT	If input is -2^{31}.2^{31}-1, convert it normally. Otherwise, the error occurs.
LINT_TO_USINT	USINT	If input is 0.255, convert it normally. Otherwise, the error occurs.
LINT_TO_UINT	UINT	If input is 0.65,535, convert it normally. Otherwise, the error occurs.
LINT_TO_UDINT	UDINT	If input is 0.2^{32}-1, convert it normally. Otherwise, the error occurs.
LINT_TO_ULINT	ULINT	If input is 0.2^{64}-1, convert it normally. Otherwise, the error occurs.
LINT_TO_BOOL	BOOL	Convert lower 1Bit to BOOL type.
LINT_TO_BYTE	BYTE	Convert lower 8Bit to BOOL type.
LINT_TO_WORD	WORD	Convert lower 16Bit to BOOL type.
LINT_TO_DWORD	DWORD	Convert lower 32Bit to BOOL type.
LINT_TO_LWORD	LWORD	Convert LINT to LWORD without conversion of internal bit array.
LINT_TO_BCD	LWORD	If input is 0~9,999,999,999,999,999, convert it normally. Otherwise, the error occurs.
LINT_TO_REAL	REAL	Convert LINT to REAL type. Conversion error rate is depend on precision.
LINT_TO_LREAL	LREAL	Convert LINT to LREAL type. Conversion error rate is depend on precision.

■ **Error**

When conversion error occurs, _ERR and _LER flag is set.

Note When error occurs, outputs bits from lower bit of IN as much as output type bit number without conversion of internal bit array.

■ **Program example**

LD	IL
	LD %I0.0.0 JMPN AAA LD IN_VAL LINT_TO_DINT ST OUT_VAL AAA:

(1) If the execution condition(%I0.0.0) is On, LINT_TO_DINT function is executed.
(2) The output variable IN_VAL(LINT type) = 123_456_789, OUT_VAL(DINT type) = 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)

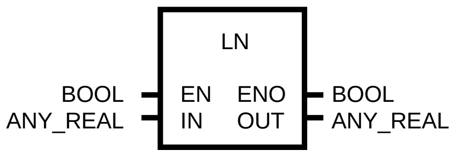
Output(OUT) : OUT_VAL(DINT) = 123,456,789
(16#75BCD15)

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

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of natural logarithm operation **Output** ENO : Output 1 in case of no error OUT : Natural logarithm value IN and OUT shall be same data type.

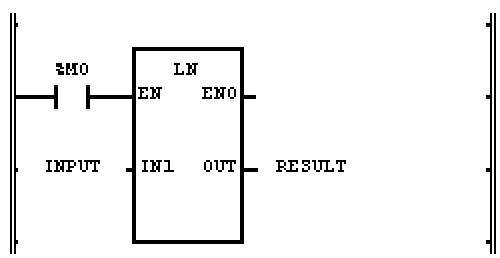
■ **Function**

Output IN's natural logarithm value to OUT.
OUT = ln IN

■ **Error**

If the input value is 0 or negative, _ERR and _LER flag is set.

■ **Program example**

LD	IL
	LD %M0 JMPN AE LD INPUT LN ST RESULT AE:

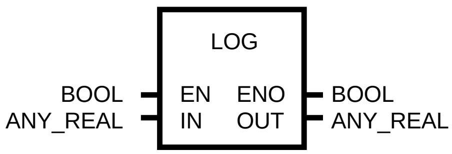
(1) If the execution condition(%M0) is On, LN(natural logarithm operation) function is executed.
(2) The output variable INPUT value is 2.0, output variable RESULT is 0.6931 ....
ln (2.0) = 0.6931....

Input(IN1) : INPUT(REAL) = 2.0
(LN)
Output(OUT) : RESULT(REAL) = 6.93147182E-01

LOG

Logarithm operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input value of logarithm operation **Output** ENO : Output 1 in case of no error OUT : Commercial logarithm value IN and OUT shall be same data type.

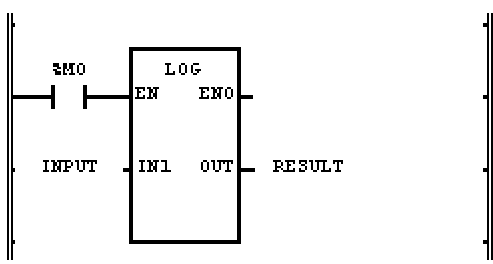
■ **Function**

Output IN's logarithm value to OUT.
OUT = $\log_{10}$ IN = log IN

■ **Error**

If the input value is 0 or negative, _ERR and _LER flag is set.

■ **Program example**

LD	IL
	LD %M0 JMPN BB LD INPUT LOG ST RESULT BB:

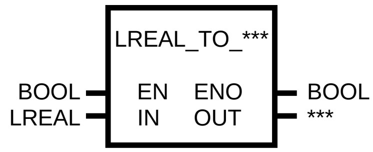
(1) If the execution condition(%M0) is On, LOG(commerical logarithm operation) function is executed.
(2) The output variable INPUT value is 2.0, output variable RESULT is 0.3010 ....
$\log_{10}(2.0) = 0.3010....$

Input(IN1) : INPUT(REAL) = 2.0
(LOG)
Output(OUT) : RESULT(REAL) = 3.01030010E-01

LREAL_TO_***

LREAL type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : LREAL value to be converted **Output** ENO : Execute 1 in case of no error OUT : Type converted data

■ Function

Convert IN to OUT data type.

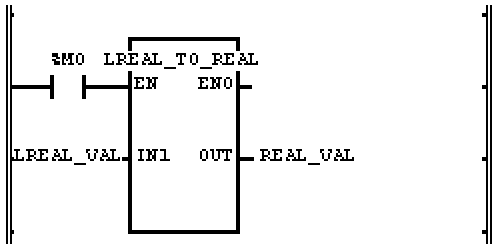
FUNCTION	Output type	Description
LREAL_TO_SINT	SINT	If input integer is -128.127, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_INT	INT	If input integer is -32768.32767, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_DINT	DINT	If input integer is $-2^{31}.2^{31}-1$, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_LINT	LINT	If input integer is $-2^{31}.2^{31}-1$, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_USINT	USINT	If input integer is 0.255, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_UINT	UINT	If input integer is 0.65,535, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_UDINT	UDINT	If input integer is $0.2^{32}-1$, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_ULINT	ULINT	If input integer is $0.2^{64}-1$, convert it normally. Otherwise, the error occurs.(round to decimal point)
LREAL_TO_LWORD	LWORD	Convert LREAL to LWORD type without conversion of internal bit array.
LREAL_TO_REAL	REAL	Convert LREAL to REAL normally. Conversion error rate is depend on precision.

■ Error

If the overflow occurs because input is larger than the storage capacity of output type, _ERR and _LER flag is set.

Note When the error occurs, output 0.

■ **Program example**

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M0
JMPN LLL
LD LREAL_VAL
LREAL_TO_REAL
ST REAL_VAL
LLL:</td></tr>
</table>

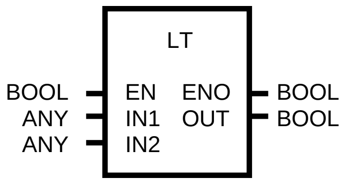
(1) If the execution condition(%M0) is On, LREAL_TO_REAL function is executed.
(2) The input variable LREAL_VAL(LREAL type) = -1.34E-12, output variable REAL_VAL (REAL type)=-1.34E-12.

Input(IN1) : LREAL_VAL (LREAL) = -1.34E-12
(LREAL_TO_REAL)
Output(OUT) : REAL_VAL (REAL) = -1.34E-12

LT

'Less than' comparison

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

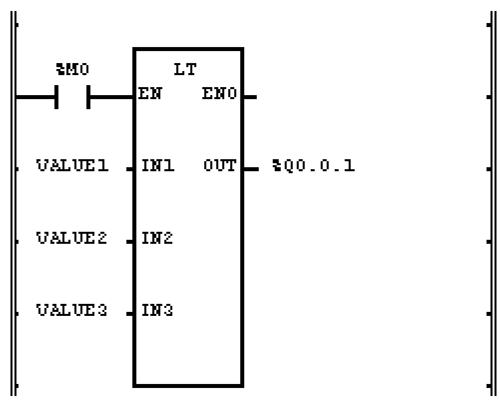
Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be compared IN2 : Comparing value Can be extended to 8 inputs. IN1, IN2, ... shall be same type. **Output** ENO : Execute EN itself OUT : Comparison result

■ Function

If IN1<IN2<IN3...<INn(n: input number), OUT outputs 1.
Otherwise, OUT outputs 0.

■ Program example

LD



IL

```
LD                  %M0
JMPN                AA
LD                  VALUE1
LT        IN1:=     CURRENT RESULT
          IN2:=     VALUE2
          IN3:=     VALUE3
ST                  %Q0.0.1
AA:
```

(1) If the execution condition(%M0) is On, LT(comparison:less) function is executed.
(2) The input variable VALUE1 = 100 and VALUE2 = 200 and VALUE3 = 300, the output result %Q0.0.1 will be 1 since comparison result VALUE1 < VALUE2 < VALUIE3.

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

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

< (LT)

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

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

< (LT)

(IN3) : VALUE3(INT) = 300(16#012C)

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

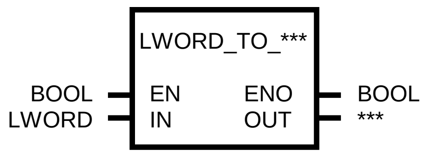
Output(OUT) : %Q0.0.1(BOOL) = 1(16#1)

1

LWORD_TO_***

LWORD type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.			

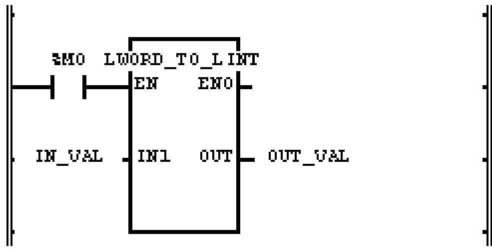
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit array to be converted(64Bit) **Output** ENO : Output EN value itself OUT : Type converted data

■ Function

Convert IN to OUT data type.

FUNCTION	Output type	Description
LWORD _TO_SINT	SINT	Convert lower 8Bit to SINT type.
LWORD _TO_INT	INT	Convert lower 16Bit to INT type.
LWORD _TO_DINT	DINT	Convert lower 32Bit to DINT type.
LWORD _TO_LINT	LINT	Convert LWORD to LINT without conversion of internal bit array.
LWORD _TO_USINT	USINT	Convert lower 8Bit to USINT type.
LWORD _TO_UINT	UINT	Convert lower 16Bit to UINT type.
LWORD _TO_UDINT	UDINT	Convert lower 32Bit to UDINT type.
LWORD _TO_ULINT	ULINT	Convert LWORD to ULINT without conversion of internal bit array.
LWORD _TO_BOOL	BOOL	Convert lower 1Bit to BOOL type.
LWORD _TO_BYTE	BYTE	Convert lower 8Bit to BYTE type.
LWORD _TO_WORD	WORD	Convert lower 16Bit to WORD type.
LWORD _TO_DWORD	DWORD	Convert lower 32Bit to LWORD type.
LWORD _TO_LREAL	LREAL	Convert LWORD to LREAL type.
LWORD _TO_DT	DT	Convert LWORD to DT type without conversion of internal bit array.
LWORD _TO_STRING	STRING	Convert input value to STRING type.

■ **Program example**

LD	IL
	LD %M0 JMPN PPP LD IN_VAL LWORD_TO_LINT ST OUT_VAL PPP:

(1) If the execution condition(%M0) is On, LWORD_TO_LINT function is executed.

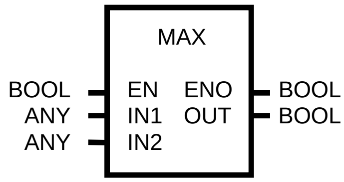
(2) The input variable IN_VAL(LWORD type) = 16#FFFFFFFFFFFFFFFF, the output variable OUT_VAL(LINT type) = -1(16#FFFFFFFFFFFFFFFF).

Input(IN1) : IN_VAL(LWORD) = 16#FFFFFFFFFFFFFFFF
(LWORD_TO_LINT)
Output(OUT) : OUT_VAL(LINT) = -1

MAX

Maximum value

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td></td><td>Input EN : Execute the function in case of 1
IN1 : Value to be compared
IN2 : Comparing value
Can be extended to 8 inputs.

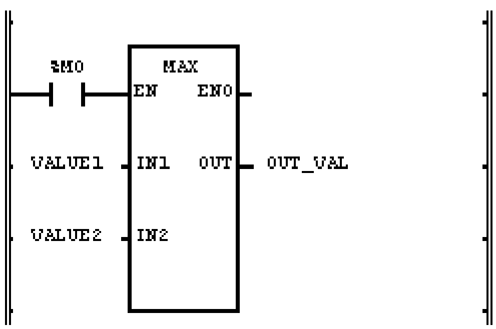
Output ENO : Output EN value itself
OUT : Maximum value of input value

IN1, IN2, ..., OUT shall be same type.</td></tr>
</table>

■ Function

Output maximum value of input among IN1, IN2, ..., INn(n: input number) to OUT.

■ Program example

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

(1) If the execution condition(%M0) is On, MAX(maximum value) function is executed.

(2) Compare the input variable VALUE1 = 100 and VALUE2 = 200, output OUT_VALUE = 200 since maximum is 200.

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

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	1	1	0	0	1	0	0	0

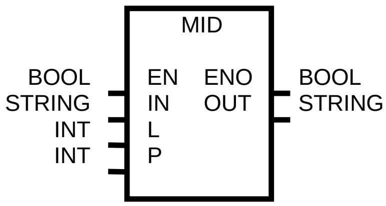
Output(OUT): OUT_VAL(INT) = 200(16#00C8)

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

MID

Middle of character string

Product	GM1	GM2	GM3	GM4	GM6
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string input L : Character string length to be output P : Start position of character string to be output **Output** ENO : Execute 1 in case of no error OUT : Character string output

■ **Function**

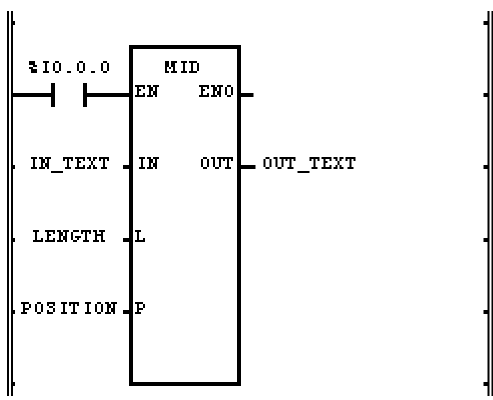
Output the character string from P-th character of IN as many as length L to OUT.

■ **Error**

If (Character number of variable IN) < P or P <= 0 and L < 0, _ERR and _LER flag is set.

■ **Program example**

LD



IL

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

(1) If the execution condition(%I0.0.0) is On, MID(middle of character string) function is executed.

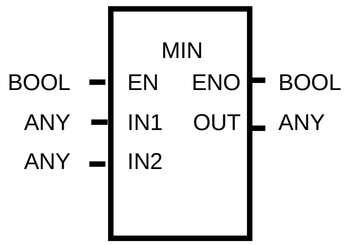
(2) If input character string is IN_TEXT=‘ABCDEFG‘, character string length is LENGTH=3 and start position of output character string is POSITION=2, output character string variable is OUT_TEXT=‘BCD‘.

Input(IN) : IN_TEXT1(STRING) = ’ABCDEFG’
(L) : LENGTH(INT) = 3
(P) : POSITION(INT) = 2
.(MID)
Output(OUT) : OUT_TEXT = ‘BCD‘

MIN

Minimum value

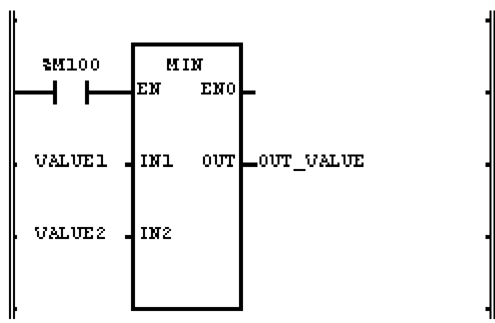
Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
 	**Input** EN : Execute the function in case of 1 IN1 : Value to be compared IN2 : Comparing value Input can be extended to 8 **Output** ENO : Output EN value itself OUT : Minimum value of input IN1, IN2, ..., OUT shall be same type.

■ **Function**

Output minimum value of input IN1, IN2,...., INn(n: input number) to OUT.

■ **Program example**

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

(1) If the execution condition(%M100) is On, MIN(minimum value) function is executed.

(2) The input variable VALUE1 = 100 and VALUE2 = 200, the output variable OUT_VALUE will be 100 since minimum value is 100.

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

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	1	1	0	0	1	0	0	0

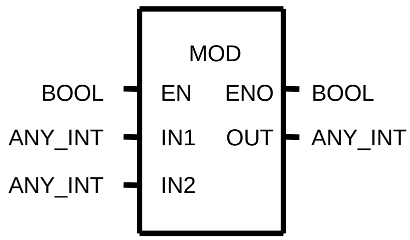
Output(OUT): OUT_VAL(INT) = 100(16#0064)

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

MOD

Remainder calculation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

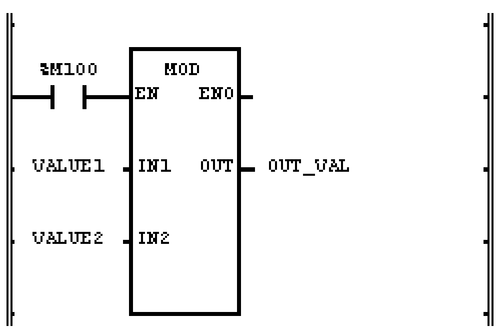
Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Dividend IN2 : Divisor **Output** ENO : Output EN value itself OUT : Dividing result(remainder) Variable connecting to IN1, IN2, OUT shall be same data type.

■ **Function**

Divide IN1 by IN2 and output the remainder to OUT.
OUT = IN1 - (IN1/IN2) × IN2 (But, 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
	LD %M100 JMPN BB LD VALUE1 MOD IN1:= CURRENT RESULT IN2:= VALUE2 ST OUT_VAL BB:

(1) If the execution condition(%M100) is On, MON(remainder calculation) function is executed.
(2) If the dividend VALUE1 = 37 and divisor VALUE2 = 10, the output variable OUT_VAL will be 7.

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

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

(MOD)

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

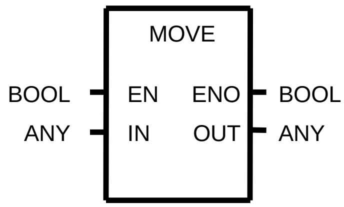
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Output(OUT): OUT_VAL(INT) = 7(16#0007)

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

MOVE

Data move		Product	GM1	GM2	GM3	GM4	GM6
		Applicable	●	●	●	●	●

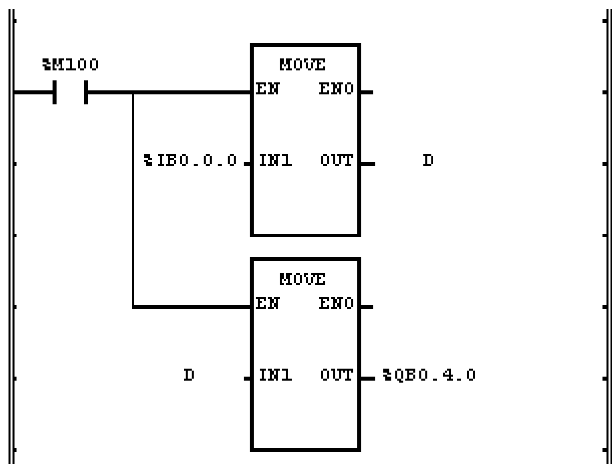
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Value to be moved **Output** ENO : Output EN value itself OUT : Moved value Variable connected to IN and OUT shall be same type.

■ Function

Move IN to OUT.

■ Program example

Program that transfers 8 point input of input (%I0.0.0□%I0.0.7) to 8 point output %Q0.4.0□%Q0.4.7.

LD	IL
	LD %M100 JMPN AAA LD %IB0.0.0 MOVE ST D LD D MOVE ST %QB0.4.0 AAA :

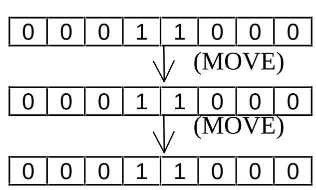
(1) If the execution condition(%M100) is On, MOVE(data copy) function is executed.

(2) Move 8 point input data of input module to variable D area by first MOVE function and output the input module status in variable D to the output module by second MOVE function.

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

D(BYTE) = 16#18

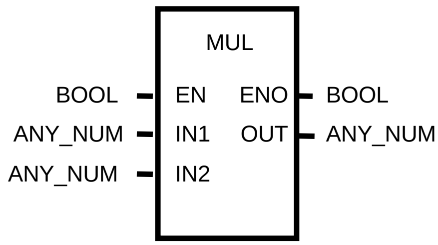
Output(OUT) : %QB0.4.0(BYTE) = 16#18



MUL

Multiply

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Moltiplicand IN2 : Multiplier Can be extended to 8 inputs. **Output** ENO : Output EN value itself OUT : Multiplied value Variable connected to IN1, IN2, ..., OUT shall be same type.

■ **Function**

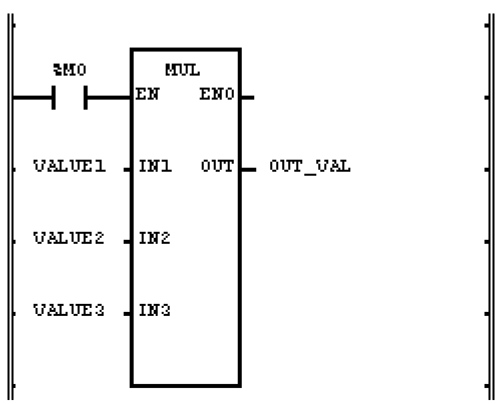
Multiply IN1, IN2,..., INn (n: input number) and output it to OUT.

OUT = IN1 × IN2× ... × INn

■ **Error**

If the output exceeds the range of respective data type, _ERR and _LER flags are set.

■ **Program example**

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

(1) If the execution condition(%M0) is On, MUL(multiply) function is executed.

(2) The input variable VALUE1 = 30 and VALUE2 = 20 and VALUE3 = 10, the output variable OUT_VAL = 30 × 20× 10 will be 6000.

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

+ (MUL)

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

+ (MUL)

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

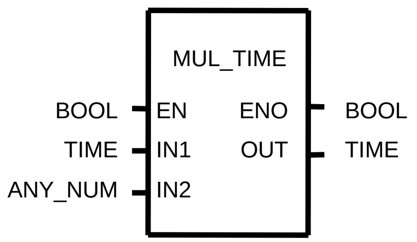
Output(OUT) : OUT_VAL(INT) = 6000(16#1770)

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

MUL_TIME

Time multiply	

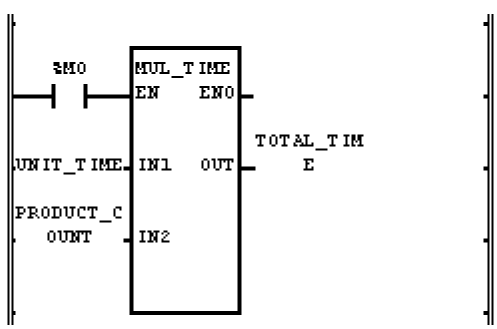
Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Time to be multiplied IN2 : Multiplying value **Output** ENO : Execute 1 in case of no error OUT : Multiplied result

■ Function
Multiply IN1(time) by IN2(number) and output the result to OUT.

■ Error
If the output exceeds the range of TIME data type, _ERR and _LER flags are set.

■ Program example
Program that calculates the required work time if average production time is 20 minutes 2 seconds and 20 products shall be manufactured.

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

(1) Input UNIT_TIME:T#20M2S to the input variable(1N1:production time of unit product).
(2) Input PRODUCT_COUNT:20 to the input variable(1N2:production quantity).
(3) Input the output variable (OUT:total required work time) to TOTAL_TIME.
(4) If the execution condition(% M0) is On, TOTAL_TIME outputs T#6H40M40S.

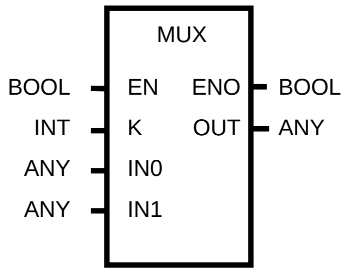
Input(IN1) : UNIT_TIME(TIME) = T#20MS2S
(MUL_TIME)
(IN2) : PRODUCT_COUNT(INT) = 20

Output(OUT): TOTAL_TIME(TIME) = T#6H40M40S

MUX

Multiplexer

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 K : Selection IN0 : Value to be selected IN1 : Value to be selected Can be extended to 7 inputs(IN0, IN1,..., IN6). **Output** ENO : Output 1 in case of no error OUT : Selected value IN0, IN1, ..., OUT shall be same type.

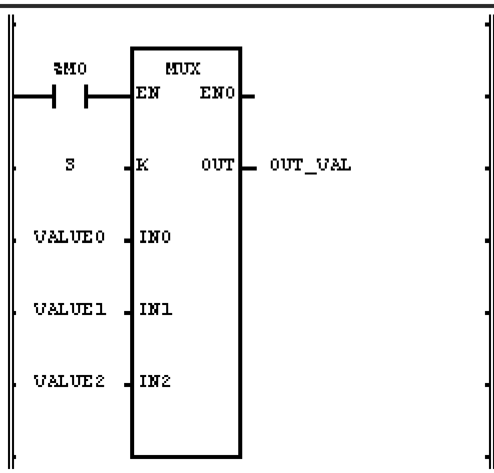
■ **Function**

Select one of several input(IN0, IN1,..., INn) and output it to K.
If K = 0, OUT outputs IN0, if K = 1, OUT outputs IN1, if K = n, INn will be output to OUT.

■ **Error**

If K is larger than or equal to the number of input variable Inn, OUT outputs IN0 and _ERR and _LER flags are set.

■ **Program example**

LD	IL
	LD %M0 JMPN ABC LD S MUX K := CURRENT RESULT IN0 := VALUE0 IN1 := VALUE1 IN2 := VALUE2 ST OUT_VAL ABC:

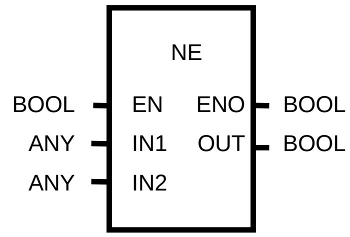
(1) If the execution condition(%M0) is On, MUX(select multiplex) function is executed.
(2) Select on of input variable VALUE0, 1, 2 and output it to OUT.

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

NE

'Not equal' comparison

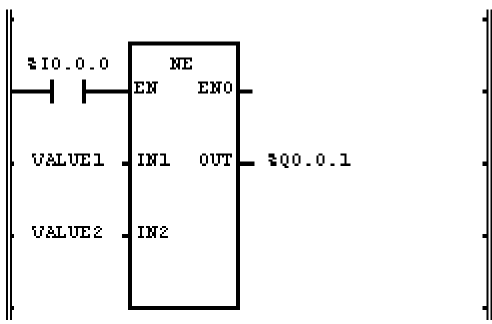
Product	GM1	GM2	GM3	GM4	GM6
Applicable	•	•	•	•	•

Function	Description
	**Input** EN : Execution allow IN1 : Value to be compared IN2 : Value to be compared IN1 and IN2 shall be same type. **Output** ENO : Execute EN value itself OUT : Comparison result

■ Function

If IN1 is not equal to IN2, OUT outputs 1.
If IN1 is equal to IN2, OUT outputs 0.

■ Program example

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

(1) If the execution condition(%I0.0.0) is On, NE(Comparison: not equal) function is executed.
(2) The input variable VALUE1 = 100 and VALUE2 = 200, the output result %Q0.0.1 will be 1 since VALUE1 is not equal to VALUE2.

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

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

(NE)

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

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

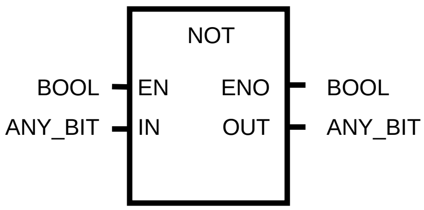
Output(OUT) : %Q0.0.1(BOOL) = 1(16#1)

1

NOT

Not

Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆	◆	◆	◆

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>
</td><td>Input EN : Execute the function in case of 1
IN : Value to be NOT

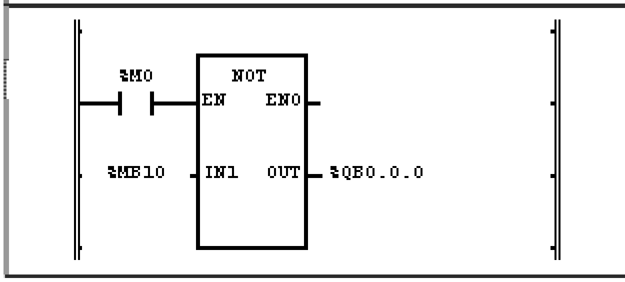
Output ENO : Output EN value itself
OUT : NOT value

IN1 and OUT shall be same type.</td></tr>
</table>

■ **Function**

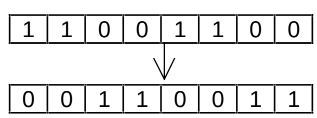
Execute NOT(inversion) to IN and output the result to OUT.
IN 1100 1010
OUT 0011 0101

■ **Program example**

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

(1) If the execution condition(%M0) is On, NOT function is executed.
(2) If NOT function is executed, invert input variable %MB10 and output the result to output variable %0B0.0.0.0.

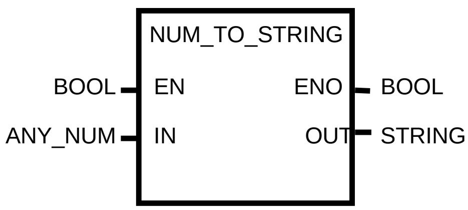
Input(IN1) : %MB10(BYTE) = 16#CC
(NOT)
Output(OUT) : %QB0.0.0(BYTE) = 16#33



NUM_TO_STRING

Number to character string conversion

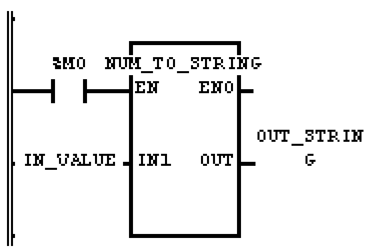
Product	GM1	GM2	GM3	GM4	GM6
Applicable	•	•	•	•	•

Function	Description
	**Input** EN : Execute the function in case of 1 IN : The number to be converted as STRING **Output** ENO : Output EN value itself OUT : Converted data

■ **Function**

Convert number to character string and output the result to OUT.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD IN_VALUE NUM_TO_STRING ST OUT_STRING AAA:

(1) If the execution condition(%M0) is On, NUM_TO_STRING(Number to character string conversion) function is executed.

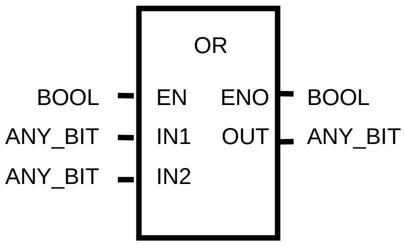
(2) If input variable of NUM_TO_STRING function IN_VALUE(INT type) is 123, output variable OUT_STRING will be ’123’ and if IN_VALUE(REAL type) is 123.0, OUT_STRING will be ’1.23E2’.

Input(IN1) : IN_VALUE(INT) = 123
↓ (NUM_TO_STRING)
Output(OUT) : OUT_STRING(STRING) = ’123’

OR

Logical OR

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

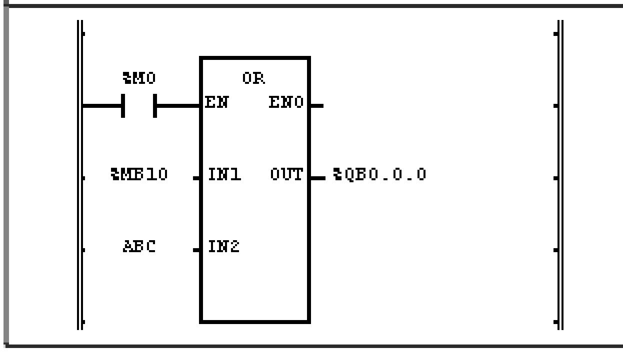
Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be OR IN2 : Value to be OR Can be extended to 8 inputs. **Output** ENO : Output EN value itself OUT : OR value IN1, IN2, ..., OUT shall be same type.

■ **Function**

Execute OR of IN1 and IN2 and output the result to OUT.

IN1 1111 0000
OR
IN2 1010 1010
OUT 1111 1010

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD %MB10 OR IN1:= CURRENT RESULT IN2:= ABC ST %QB0.0.0 AAA:

(1) If the execution condition(%M0) is On, OR function is executed.
(2) OR result of %MB10 = 11001100 and ABC = 11110000 is output to %QB0.0.0 = 11111100.

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

1	1	0	0	1	1	0	0

& (OR)

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

1	1	1	1	0	0	0	0

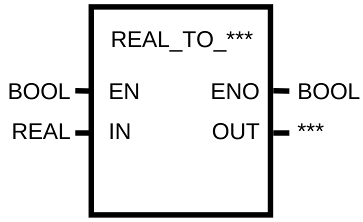
Output(OUT) : %QB0.0.0(BYTE) = 16#FC

1	1	1	1	1	1	0	0

REAL_TO_***

REAL type conversion	

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : REAL value to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ Function

Convert IN to OUT data type.

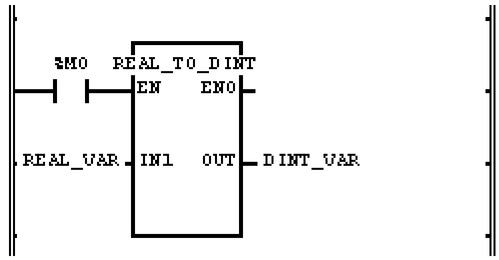
FUNCTION	Output type	Description
REAL_TO_SINT	SINT	If input integer is -128 □ 127, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_INT	INT	If input integer is -32768 □ 32767, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_DINT	DINT	If input integer is -2^{31} □ 2^{31}-1, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_LINT	LINT	If input integer is -2^{63} □ 2^{63}-1, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_USINT	USINT	If input integer is 0 □ 255, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_UINT	UINT	If input integer is 0 □ 65,535, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_UDINT	UDINT	If input integer is 0 □ 2^{32}-1, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_ULINT	ULINT	If input integer is 0 □ 2^{64}-1, convert it normally. Otherwise, the error occurs.(round to decimal point)
REAL_TO_DWORD	DWORD	Convert REAL to DWORD without conversion of internal bit array.
REAL_TO_LREAL	LREAL	Convert REAL to LREAL type.

■ Error

If the overflow occurs since input value is greater than the value to be stored at output type, _ERR and _LER flags are set.

Note If the error occurs, output 0.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD REAL_VAL REAL_TO_DINT ST DINT_VAL AAA:

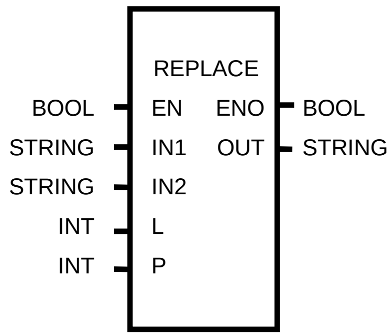
(1) If the execution condition(%M0) is On, REAL_TO_DINT function is executed.
(2) If REAL_VAL(REAL type) = 1.234E4, DINT_VAL(DINT type) = 12340.

Input(IN1) : REAL_VAL(REAL) = 1.234E4
(REAL_TO_DINT) ↓
Output(OUT) : DINT_VAL(DINT) = 12340

REPLACE

Character string replacement

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Character string to be replaced IN2 : Character string to be replaced L : Character string length to be replaced P : Character string position to be replaced **Output** ENO : Output 1 in case of no error OUT : Character string output

■ Function

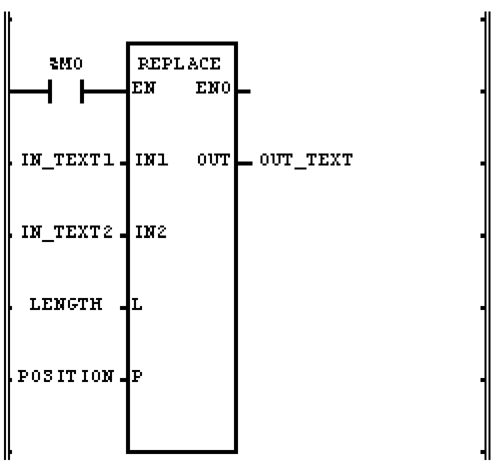
Replace character of length L from Pth character in character string IN1 to character string IN2 and output the result to character string OUT.

■ Error

For the below case, _ERR and _LER flags are set.

- P≤0 or L < 0
- P > (character number of input character string IN1)
- Character number of operation result > 30

■ Program example

LD	IL
	LD %M0 JMPN ABC LD IN_TEXT1 REPLACE IN1:= CURRENT RESULT IN2: = IN_TEXT2 L : = LENGTH P : = POSITION ST OUT_TEXT ABC:

(1) If the execution condition(%M0) is On, REPLACE(character string replace) function is executed.

(2) If input variable to be replaced IN_TEXT1 is ‘ABCDEF‘ and replacing input variable IN_TEXT2 is ‘X‘ and input variable length to be replaced LENGTH is 3 and location input variable to be replaced POSITION is 2, ‘BCD‘ of IN_TEXT is replaced to ‘X‘ of IN_TEXT2 and OUT_TEXT outputs ‘AXET‘.

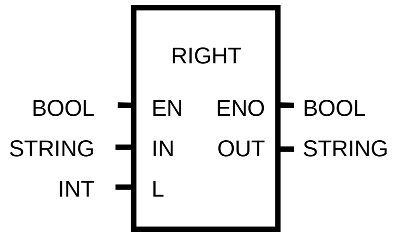
Input (IN1) : IN_TEXT1(STRING) = ‘ABCDEF‘
(IN2) : IN_TEXT2(STRING) = ‘X‘
(L) : LENGTH(INT) = 3
(P) : POSITION(INT) = 2

Output (OUT) : OUT_TEXT(STRING) =‘AXET‘

RIGHT

Right of character string

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string input L : Character string length to be output **Output** ENO : Output 1 in case of no error OUT : Character string output

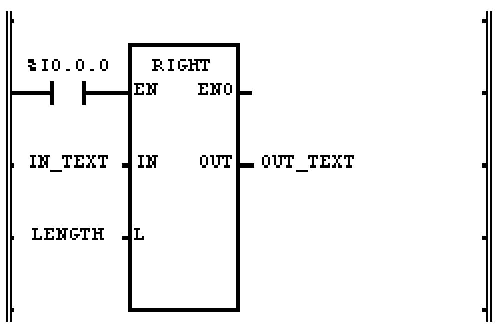
■ **Function**

Output character of length L from right of IN to character string OUT.

■ **Error**

If L < 0, _ERR and _LER flags are set.

■ **Program example**

LD	IL
	LD %I0.0.0 JMPN AAA LD IN_TEXT RIGHT IN := CURRENT RESULT L : = LENGTH ST OUT_TEXT AAA:

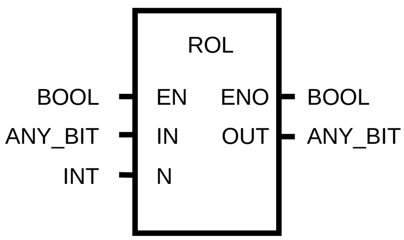
(1) If the execution condition(%M0) is On, RIGHT(right of character string) function is executed.
(2) The input variable IN_TEXT=‘ABCDEFG‘ and strength length LENGTH=3, output variable OUT_TEXT=‘EFG‘.

Input (IN1) : IN_TEXT(STRING) = ‘ABCDEFG’
(L) : LENGTH(INT) = 3
↓ (RIGHT)
Output(OUT) : OUT_TEXT(STRING) = ‘EFG‘

ROL

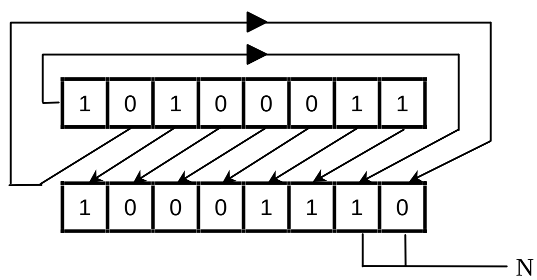
Rotate Left

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Value to be rotated N : Bit number to be rotated **Output** ENO : Output EN value itself OUT : Rotated value

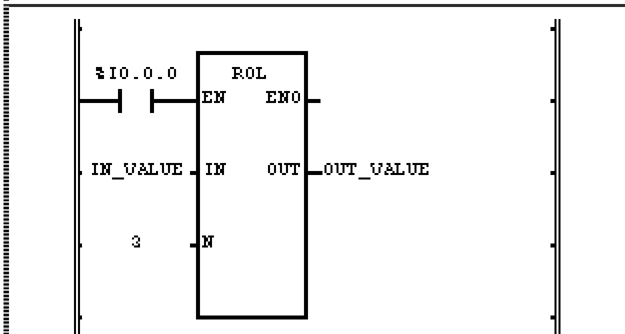
■ Function

Rotate input IN as many as N bit number to left direction.



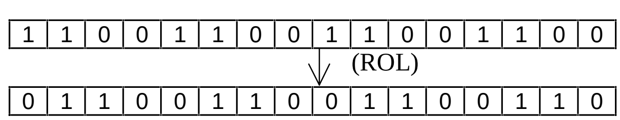
■ Program example

Program that rotates input data(1100_1100_1100_1100:16#CCCC) to left direction as 3 bit.

LD	IL
	LD %I0.0.0 JMPN PPP LD IN_VALUE ROL IN := CURRENT RESULT N := 3 ST OUT_VALUE PPP:

(1) Set the data to be rotated to input value IN_VALUE.
(2) Set bit 3 of rotating bit number to the input(N).
(3) Set the output variable, which outputs the rotated data value, to OUT_VALUE.
(4) If the execution condition %I0.0.0 is On, ROL(rotate left) function is execution so that rotates data bit of input variable to left 3bit and output to OUT_VALVE.

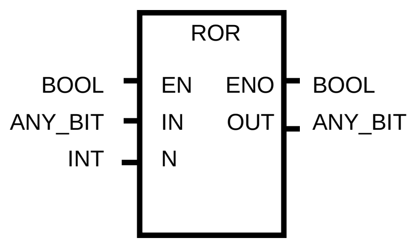
Input (IN1) : IN_VALUE(WORD) = 16#CCCC
(N) : 3
Output(OUT) :OUT_VALUE(WORD) = 16#6666



ROR

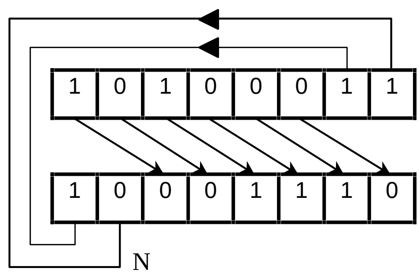
Rotate Right

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Value to be rotated N : Bit number to be rotated **Output** ENO : Output EN value itself OUT : Rotated value

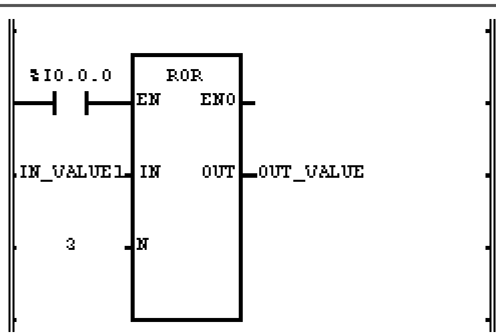
■ Function

Rotate input IN as many as N bit number to right direction.



■ Program example

Program that rotates input data(1110001100110001:16#E331) to right direction as 3 bit when input %I.0.0.0 is on.

LD	IL
	LD %I0.0.0 JMPN PO LD IN_VALUE1 ROR IN1:= CURRENT RESULT N:= 3 ST OUT_VALUE PO

(1) Set the data to be rotated to input value IN_VALUE1.
(2) Set bit 3 of rotating right bit number to the input(N).
(4) If the execution condition %I0.0.0 is On, ROR(rotate right) function is execution so that rotates data bit of input variable to right 3bit and output to OUT_VALVE.

Input (IN1) : IN_VALUE1(WORD) = 16#E331
(N) : 3
Output(OUT) :OUT_VALUE(WORD)=16#3C66

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

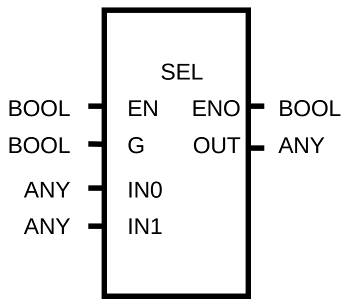
↓ (ROR)

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

SEL

Selection

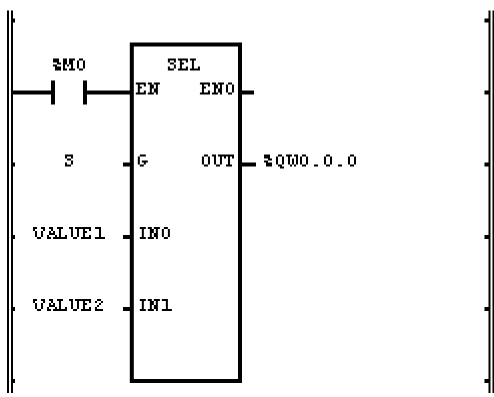
Product	GM1	GM2	GM3	GM4	GM6
Applicable	•	•	•	•	•

Function	Description
	**Input** EN : Execute the function in case of 1 G : Selection IN0 : Value to be selected IN1 : Value to be selected **Output** ENO : Output EN value itself OUT : Selected value IN1, IN2 and OUT shall be same type.

■ **Function**

As OUT outputs IN0 when G is 0, OUT outputs IN1 when G is 1.

■ **Program example**

LD	IL
	LD %M0 JMPN PPP LD S SEL G := CURRENT RESULT IN1:= VALUE1 IN2:= VALUE2 ST %QW0.0.0 PPP:

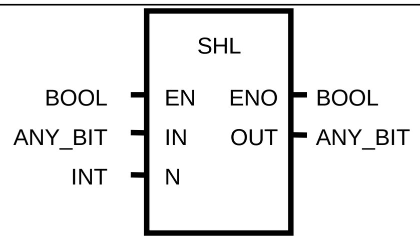
(1) If the execution condition(%M0) is On, SEL(selection) function is executed.
(2) If SEL function is executed, %QW0.0.0 = 16#FF0 when S = 1 and VALUE1 = 16#1110 and VALUE2 = 16#FF00.

Input (G) : S = 1
(IN0) : VALUE1(WORD) = 16#1110
(IN1) : VALUE2(WORD) = 16#FF00
↓ (SEL)
Output(OUT) : %QW0.0.0(WORD) = 16#FF00

SHL

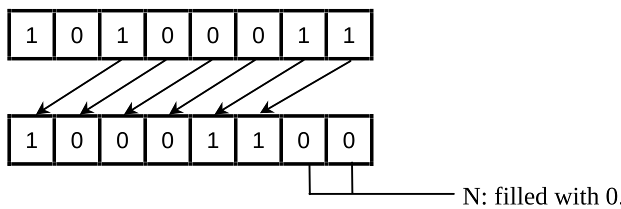
Shift Left

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit array to be shifted N : Bit number to be moved **Output** ENO : Output EN value itself OUT : Shifted value

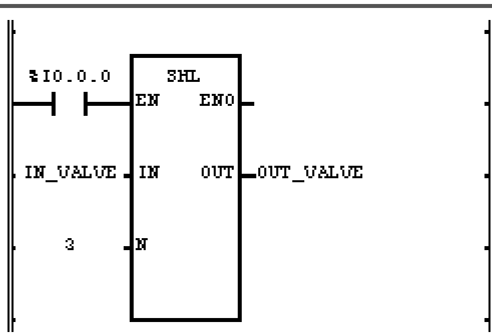
■ Function

Shift input IN as many as N bit number to 2>2> left direction.
Fill N bit at the right of input IN with 0.



■ Program example

Program that shifts left input data(1100_1100_1100_1100:16#CCCC) to 3 bit when input %I.0.0.0 is on.

LD	IL
	LD %I0.0.0 JMPN ABC LD IN_VALUE SHL IN:= CURRENT RESULT N := 3 ST OUT_VALUE ABC:

(1) Set the input variable to IN_VALUE(11001110:16#CE).
(2) Set Bit number 3 to the input(N).
(3) If the execution condition(%I0.0.0) is On, SHL(shift left) function is executed so that shifts left the input variable data bit to 3 bit and output the result to OUT_VALUE.

Input(IN1) : IN_VALUE(WORD) = 16#CCCC
(N) : 3
Output(OUT) :OUT_VALUE(WORD) = 16#6660

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

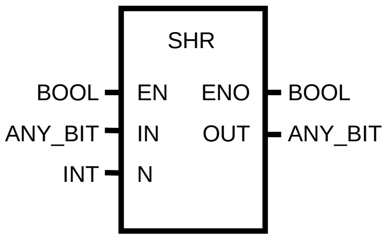
(SHL)

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

SHR

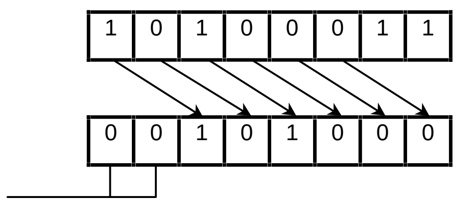
Shift right

Product	GM1	GM2	GM3	GM4	GM6
Applicable	•	•	•	•	•

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit array to be shifted N : Bit number to be moved **Output** ENO : Output EN value itself OUT : Shifted value

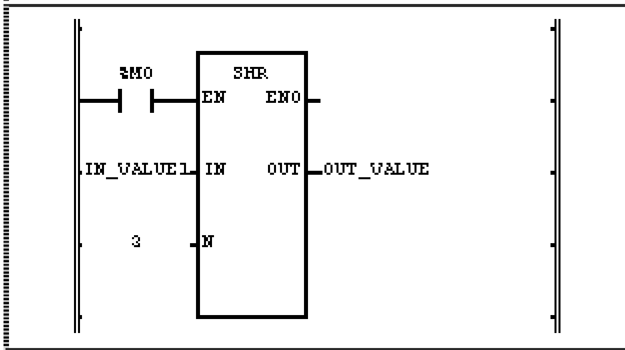
■ Function

Shift input IN as many as N bit number to 2>2> right direction.
Fill N bit at the right of input IN with 0.



N: filled with 0.

■ Program example

LD	IL
	LD %M0 JMPN AAA LD IN_VALUE SHR IN := CURRENT RESULT N := SHIFT_NUM ST OUT_VALUE AAA:

(1) If the execution condition(%M0) is On, SHR(shift right) function is executed.
(2) Shift right the input variable data bit to 3 bit and output it to output variable OUT_VALUE.

Input (IN1): IN_VALUE(WORD) = 16#E331
(N) : 3
Output(OUT) : OUT_VALUE(WORD) = 16#1C66

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

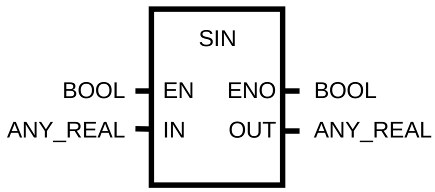
(SHR)

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

SIN

Sine operation

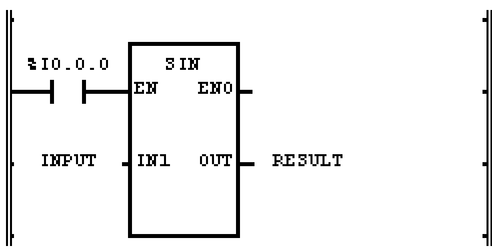
Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●			

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Radian value of sine operation **Output** ENO : Output EN value itself OUT : Sine operation result IN and OUT shall be same type.

■ **Function**

Output sine value of IN to OUT.
OUT = SIN (IN)

■ **Program example**

LD	IL
	LD %I0.0.0 JMPN PPP LD INPUT SIN ST RESULT PPP:

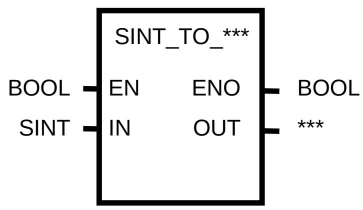
(1) If the execution condition(%I0.0.0) is On, SIN(Sine operation) function is executed.
(2) When input variable INPUT is 1.0471 .... (π/3 rad = 60°), output variable RESULT outputs 0.8660 ....($\sqrt{3}/2$).
SIN (π/3) = $\sqrt{3}/2$ = 0.8660

Input(IN1) : INPUT(REAL) = 1.0471
↓ (SIN)
Output(OUT) : RESULT(REAL) =8.65976572E-01

SINT_TO_***

SINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Short Integer to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

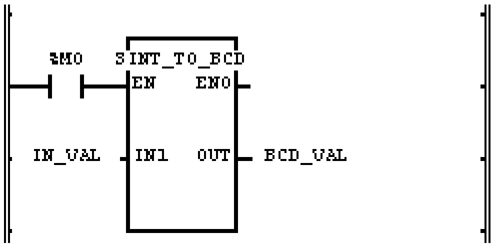
FUNCTION	Output type	Description
SINT_TO_INT	INT	Convert to INT type normally.
SINT_TO_DINT	DINT	Convert to DINT type normally.
SINT_TO_LINT	LINT	Convert to LINT type normally.
SINT_TO_USINT	USINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
SINT_TO_UINT	UINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
SINT_TO_UDINT	UDINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
SINT_TO_ULINT	ULINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
SINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
SINT_TO_BYTE	BYTE	Convert SINT to BYTE type without conversion of internal bit array.
SINT_TO_WORD	WORD	Convert upper bit to WORD type filled with 0.
SINT_TO_DWORD	DWORD	Convert upper bit to DWORD type filled with 0.
SINT_TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
SINT_TO_BCD	BYTE	If input is 0 ~ 99, convert it normally. Otherwise, the error occurs.
SINT_TO_REAL	REAL	Convert SINT to REAL type normally.
SINT_TO_LREAL	LREAL	Convert SINT to LREAL type normally.

■ **Error**

If conversion error occurs, _ERR and _LER flags are set.

Note If error occurs, outputs bits from lower bit of IN as many as output type bit without conversion of internal bit array.

■ **Program example**

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M0
JMPN AAA
LD IN_VAL
SINT_TO_BCD
ST BCD_VAL
AAA:</td></tr>
</table>

(1) If the execution condition(%M0) is On, SINT_TO_BCD function is executed.
(2) The input variable IN_VAL(SINT type) = 64(2#0100_0000), 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)

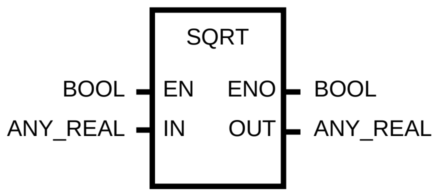
Output(OUT) : OUT_VAL(BCD) = 16#64(16#64)

0	1	1	0	0	1	0	0

SQRT

Square root operation

Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆			

Function	Description
 	**Input** EN : Execute the function in case of 1 IN : Input value of square root operation **Output** ENO : Output 1 in case of no error OUT : Square root value IN and OUT shall be same data type.

■ **Function**

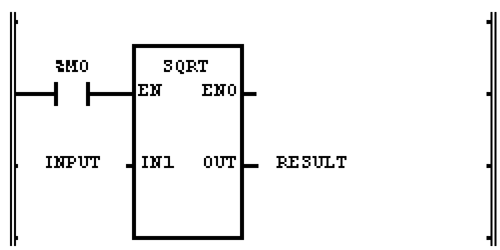
Output the square root of IN to OUT.

OUT = $\sqrt{IN}$

■ **Error**

If IN is negative number, _ERR and _LER flags are set.

■ **Program example**

LD	IL
 	LD %M0 JMPN AAA LD INPUT SQRT ST RESULT AAA:

(1) If the execution condition(%M0) is On, SQRT(square root operation) function is executed.

(2) The input variable INPUT is 9.0, output variable RESULT will be 3.0.

$\sqrt{9.0} = 3.0$

Input(IN1) : INPUT(REAL) = 9.0

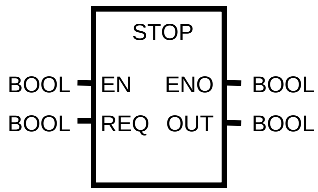
↓ (SQRT)

Output(OUT) : RESULT(REAL) = 3.0

STOP

STOP by program

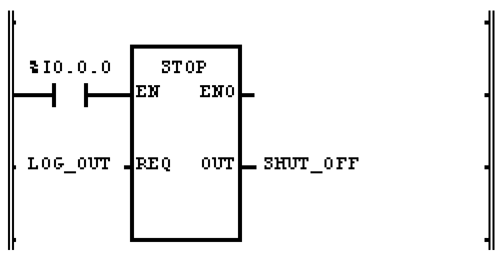
Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** EN : Execute the function in case of 1 RE : STOP request **Output** ENO : Output EN value itself OUT : Output 1 when STOP is executed

■ Function

- If EN is 1 and REQ has 1, stop the operation and go to STOP mode.
- If 'STOP' function is executed, scan program will be stopped after completing its last function.
- The operation will be run again by supplying the power or changing the mode to STOP and from STOP to RUN.

■ Program example

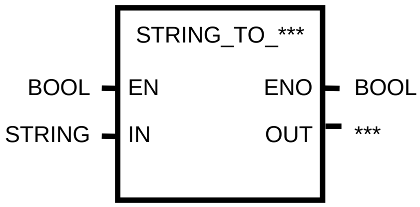
LD	IL
	LD %I0.0.0 JMPN PT LD LOG_OUT STOP ST SHUT_OFF PT:

(1) If the execution condition(%I0.0.0) is ON and LOG_OUT is 1, go to STOP mode after completing the running scan program.
(2) Switch off PLC power after executing 'STOP' function.

STRING_TO_***

STRING type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Character string to be converted **Output** ENO : Output 1 in case of no error OUT : Type converted data

■ Function

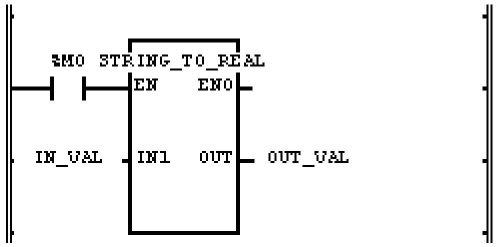
Convert IN to OUT data type.

FUNCTION	Output type	Description
STRING _TO_SINT	SINT	Convert STRING to SINT type.
STRING _TO_INT	INT	Convert STRING to INT type.
STRING _TO_DINT	DINT	Convert STRING to DINT type.
STRING _TO_LINT	LINT	Convert STRING to LINT type.
STRING _TO_USINT	USINT	Convert STRING to USINT type.
STRING _TO_UINT	UINT	Convert STRING to UINT type.
STRING _TO_UDINT	UDINT	Convert STRING to UDINT type.
STRING _TO_ULINT	ULINT	Convert STRING to ULINT type.
STRING _TO_BOOL	BOOL	Convert STRING to BOOL type.
STRING _TO_BYTE	BYTE	Convert STRING to BYTE type.
STRING _TO_WORD	WORD	Convert STRING to WORD type.
STRING _TO_DWORD	DWORD	Convert STRING to DWORD type.
STRING _TO_LWORD	LWORD	Convert STRING to LWORD type.
STRING _TO_REAL	REAL	Convert STRING to REAL type.
STRING _TO_LREAL	LREAL	Convert STRING to LREAL type.
STRING _TO_DT	DT	Convert STRING to DT type.
STRING _TO_DATE	DATE	Convert STRING to DATE type.
STRING _TO_TOD	TOD	Convert STRING to TOD type.
STRING _TO_TIME	TIME	Convert STRING to TIME type.

■ Error

If input character pattern does not match to output data type, _ERR and _LER flags are set.

■ Program example

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M0
JMPN ZZ
LD IN_VAL
STRING_TO_REAL
ST OUT_VAL
ZZ:</td></tr>
</table>

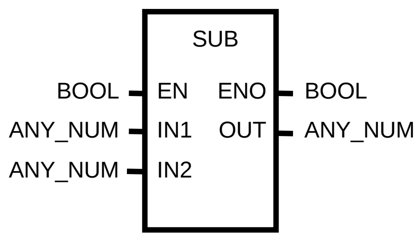
(1) If the input condition(%M0) is On, STRING_TO_REAL function is executed.
(2) If input variable IN_VAL(STRING type) = ’-1.34E12’, output variable OUT_VAL(REAL= -1.34E12.

Input(IN1) : IN_VAL(STRING) = ’-1.34E12’
↓ (STRING_TO_REAL)
Output(OUT) : OUT_VAL(REAL) = -1.34E12

SUB

Subtract

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Minuend IN2 : Subtrahend **Output** ENO : Output 1 in case of no error OUT : Maximum value of input value IN1, IN2, ..., OUT shall be same type.

■ **Function**

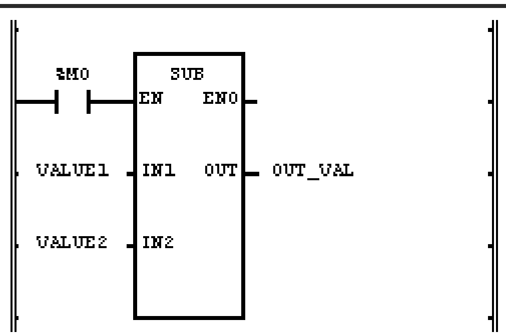
Subtract IN2 from IN1 and output the result to OUT.

OUT = IN1–IN2

■ **Error**

If the output exceeds the range of respective data type, _ERR and _LER flags are set.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD VALUE1 SUB IN1:= CURRENT RESULT IN2:= VALUE2 ST OUT_VAL AAA:

(1) If the execution condition(%M0) is On, SUB(subtract) function is executed.

(2) The input variable VALUE1 = 300 and VALUE2 = 200, the output variable OUT_VAL outputs (300-200=100).

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

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

- (SUB)

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

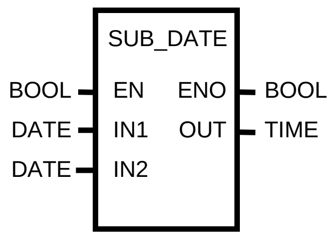
0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0

Output (OUT) : OUT_VAL(INT) = 100(16#0064)

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

SUB_DATE

Subtract date		Product	GM1	GM2	GM3	GM4	GM6
		Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Reference date IN2 : Date to be subtracted **Output** ENO : Output 1 in case of no error OUT : The difference of two dates as time

■ Function

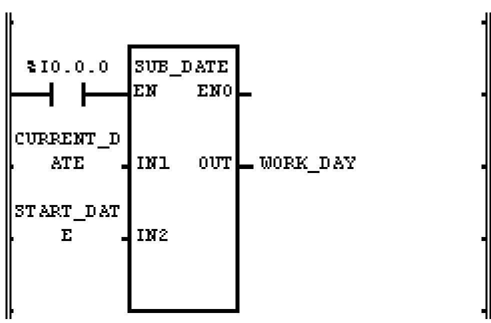
Subtract IN2(specific date) from IN1(reference date) and output the difference to OUT.

■ Error

If the output exceeds the range of TIME data type, _ERR and _LER flags are set.
If the difference exceeds the range of TIME data type T#49D17H2M47S295MS or the result is negative, the error occurs.

■ Program example

LD



IL

```
LD                      %I0.0.0
JMPN                    PPP
LD                      CURRENT_DATE
SUB_DATE     IN1:=      CURRENT RESULT
             IN2:=      START_DATE
ST                      WORK_DAY
PPP:
```

(1) If the execution condition(%I0.0.0) is On, SUB_DATE(subtract date) function is executed.
(2) If CURRENT_DATE is D#1995-12-15 and START_DATE is D#1995-11-1, WORK_DAY outputs T#44D.

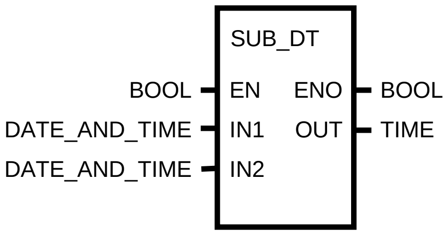
Input (IN1) : CURRENT_DATE(DATE) = D#1995-12-15
(SUB_DATE)
(IN2) : START_DATE(DATE) = D#1995-11-1

Output (OUT) : WORK_DAY(TIME) = T#44D

SUB_DT

Subtract DATE_AND_TIME

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>
</td><td>Input EN : Execute the function in case of 1
IN1 : Reference DATE_AND_TIME
IN2 : DATE_AND_TIME to be subtracted

Output ENO : Output 1 in case of no error
OUT : Subtracted time</td></tr>
</table>

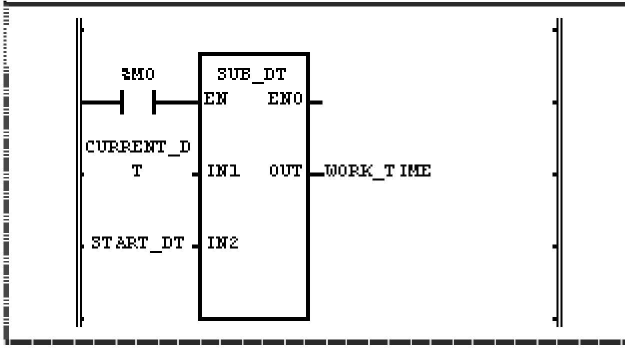
■ **Function**

Subtract IN2(specific date and time) from IN1(reference date and time) and output the difference to OUT.

■ **Error**

If the output exceeds the range of TIME data type, _ERR and _LER flags are set.
If the result is negative, the error occurs.

■ **Program example**

LD	IL
	

```
LD                %M0
JMPN              PPP
LD                CURRENT_DT
SUB_DT   IN1:=    CURRENT RESULT
         IN2:=    START_DT
ST                WORK_TIME
PPP:
```

(1) If the execution condition(%M0) is On, SUB_DT(subtract TIME and DATE) function is executed.
(2) The input variable CURRENT_DT is DT#1995-12-15-14:30:00 and work start date and time START_DT is DT#1995-12-13-12:00:00, WORK_TIME outputs T#2D2H30M.

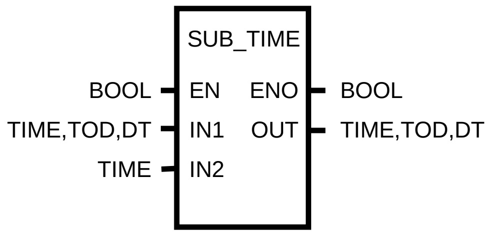
Input (IN1) : CURRENT_DT(DT) = DT#1995-12-15-14:30:00
(SUB_DATE)
(IN2) : START_DT(DT) = DT#1995-12-13-12:00:00

Output (OUT) : WORK_TIME(TIME) = T#2D2H30M

SUB_TIME

Subtract time

Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆	◆	◆	◆

Function	Description
	Input EN : Execute the function in case of 1 IN1 : Reference TIME, TOD, DT IN2 : TIME to be subtracted Output ENO : Output 1 in case of no error OUT : Subtracted TOD or TIME OUT type defends on input IN1 type. Therefore, if IN1 type is TIME, output OUT type is also TIME.

■ Function

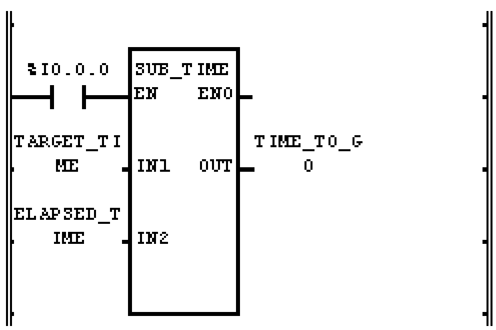
- If IN1 is TIME, subtract time from time and output the difference.
- If IN1 is TIME_OF_DAY, subtract the time from reference TOD and output TOD.
- If IN1 is DATE_AND_TIME, subtract the time from reference DT and time and output DT.

■ Error

If the output exceeds the range of respective data type, _ERR and _LER flags are set.
If the result is negative, the error occurs.

■ Program example

LD



IL

```
LD                  %I0.0.0
JMPN                AAA
LD                  TARGET_TIME
SUB_TIME    IN1:=   CURRENT RESULT
            IN2:=   ELAPSED_TIME
ST                  TIME_TO_GO
AAA:
```

(1) If the execution condition(%I0.0.0) is On, SUB_TIME(subtract TIME) function is executed.

(2) If the input variable TARGET_TIME is T#2H30M and elapsed time ELAPSED_TIME is T#1H10M30S300MS, the output variable work time TIME_TO_G outputs T#1H19M29S700MS.

Input (IN1) : TARGET_TIME(TIME) = T#2H30M
(SUB_TIME)
(IN2) : ELAPSED_TIME(TIME) = T#1H10M30S300MS

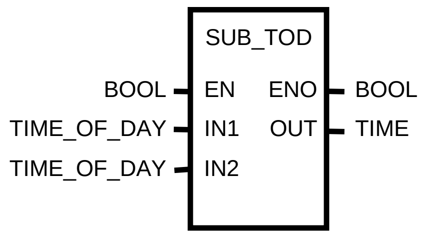
↓

Output (OUT) : TIME_TO_GO(TIME) = T#1H19M29S700MS

SUB_TOD

Subtract TOD from TOD (Time of Day)

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Reference TOD IN2 : TOD to be subtracted **Output** ENO : Output 1 in case of no error OUT : Subtracted TIME

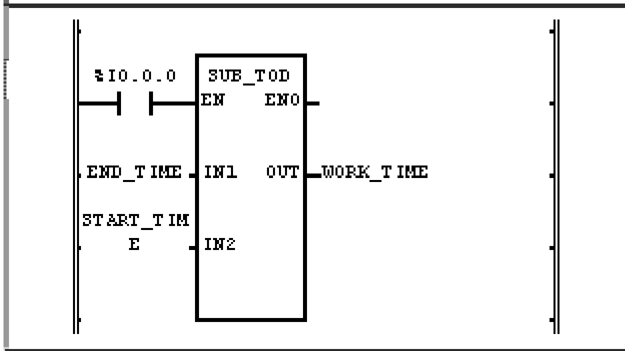
■ **Function**

Subtract IN2(specific TOD) from IN1(reference TOD) and output the difference to OUT.

■ **Error**

If the result is negative, the error occurs.

■ **Program example**

LD	IL
	LD %I0.0.0 JMPN AAA LD END_TIME SUB_TOD IN1:= CURRENT RESULT IN2:= START_TIME ST WORK_TIME AAA:

(1) If the execution condition(%I0.0.0) is On, SUB_TOD(subtract TOD from TOD) function is executed.

(2) IF the input variable END_TIME is TOD#14:20:30.5 and work start time START_TIME is TOD#12:00:00, output variable work time WORK_TIME outputs T#2H20M30S500MS.

Input(IN1) : END_TIME(TOD) = TOD#14:20:30.5

(SUB_TOD)

(IN2) : START_TIME(TOD) = TOD#12:00:00

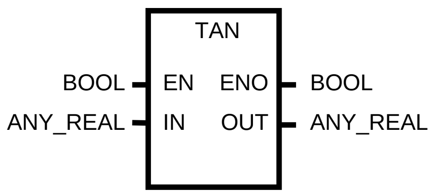
↓

Output (OUT) : WORK_TIME(TIME) = T#2H20M30S500MS

TAN

Tangent operation

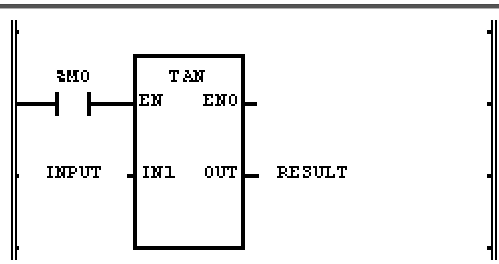
Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●		

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Tangent angle value(in radians) input **Output** ENO : Output EN value itself OUT : Tangent operation result IN and OUT shall be same type.

■ **Function**

Output Tangent value of IN to OUT.
OUT = TAN(IN)

■ **Program example**

LD	IL
	LD %M0 JMPN BBB LD INPUT TAN ST RESULT BBB:

(1) If the execution condition(%M0) is On, TAN(Tangent operation) function is executed.
(2) IF the input variable INPUT is 0.7853 .... (π/4 rad = 45°, output variable RESULT will be 1.0000.

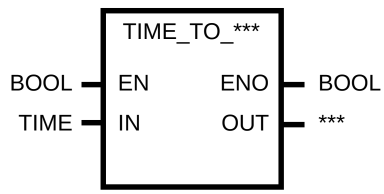
TAN(π/4) = 1

Input(IN1) : INPUT(REAL) = 0.7853
↓ (TAN)
Output(IN2) : RESULT(REAL) = 9.99803722E-01

TIME_TO_***

TIME type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

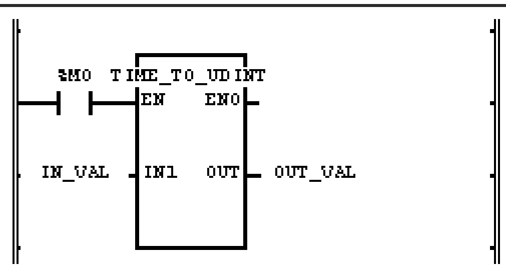
Function	Description
	**Input** EN : Execute the function in case of 1 IN : TIME data to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

FUNCTION	Output type	Description
TIME_TO_UDINT	UDINT	Convert TIME to UDINT type. Convert the data type without the conversion of internal bit data type.
TIME_TO_DWORD	DWORD	Convert TIME to DWORD type. Convert the data type without the conversion of internal bit data type.
TIME_TO_STRING	STRING	Convert TIME to STRING type.

■ **Program example**

LD	IL
	LD %M0 JMPN AA LD IN_VAL TIME_TO_UDINT ST OUT_VAL AA :

(1) If the execution condition(%M0) is On, TIME_TO_UDINT function is executed.
(2) IF the input variable IN_VAL(TIME type) = T#120MS, output variable OUT_VAL(UDINT type) will be 120.

Input(IN1) : IN_VAL(TIME) = T#120MS(16#78)

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

↓ (TIME_TO_UDINT)

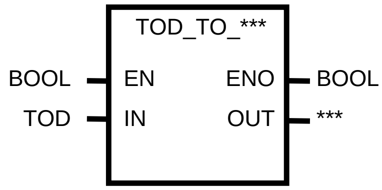
Output(OUT) : OUT_VAL(UDINT) = 120(16#78)

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

TOD_TO_***

TOD type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

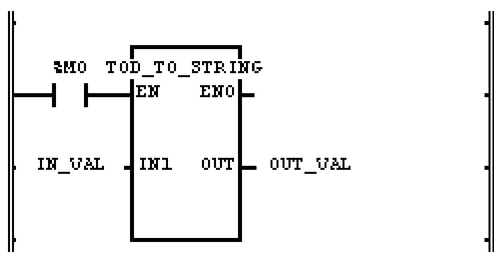
Function	Description
	**Input** EN : Execute the function in case of 1 IN : TOD to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ Function

Convert IN to OUT data type.

FUNCTION	Output type	Description
TOD_TO_UDINT	UDINT	Convert TOD to IDINT type. Convert data type only without conversion of internal bit array.
TOD_TO_DWORD	DWORD	Convert TOD to DWORD type. Convert data type only without conversion of internal bit array.
TOD_TO_STRING	STRING	Convert TOD to STRING type.

■ Program example

LD	IL
	LD % M0 JMPN AA LD IN_VAL DATE_TO_STRING ST OUT_VAL AA :

(1) If the execution condition(%M0) is On, TOD_TO_STRING function is executed.

(2) IF the input variable IN_VAL(TOD type) is TOD#12:00:00, output variable OUT_VAL(STRING type) will be 'TOD#12:00:00'.

Input(IN1) : IN_VAL(TOD) = TOD#12:00:00

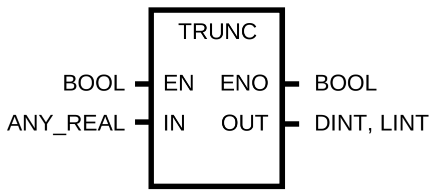
↓ (TOD_TO_STRING)

Output(IN2) : OUT_VAL(STRING) = 'TOD#12:00:00'

TRUNC

Integer conversion with round off (Truncate)

Product	GM1	GM2	GM3	GM4	GM6
Applicable	•	•	•	•	•

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Real value to be converted **Output** ENO : Execute 1 in case of no error OUT : Integer value

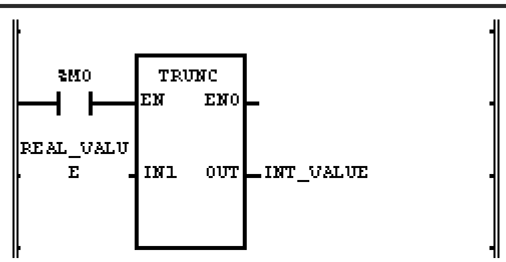
■ **Function**

FUNCTION	Input File	Output type	Description
TRUNC	REAL LREAL	DINT LINT	Round off the floating point and output the integer to OUT.

■ **Error**

If the converted value is greater than maximum value of out data type _ERR and _LER flags are set and result will be zero.

■ **Program example**

LD	IL
	LD %M0 JMPN XYZ LD REAL_VALUE TRUNC ST INT_VALUE XYZ :

(1) If the execution condition(%M0) is On, TRUNC(Integer conversion with round off) function is executed.

(2) IF the input variable REAL_VALUE(REAL type) is 1.6, INT_VALUE(INT type) will be 1.
If REAL_VALUE(REAL type) is -1.6, INT_VALUE(INT type) will be -1.

Input(IN1) : REAL_VALUE(REAL) = 1.6

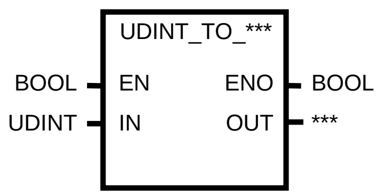
↓ (TRUNC)

Output(OUT) : INT_VALUE(INT) = 1

UDINT_TO_***

UDINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Unsigned Double Integer to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

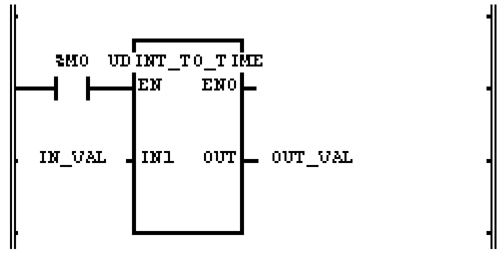
FUNCTION	Output type	Description
UDINT_TO_SINT	SINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
UDINT_TO_INT	INT	If input is 0 ~ 32767, convert it normally. Otherwise, the error occurs.
UDINT_TO_DINT	DINT	If input is 0 ~ 2,147,483,64, convert it normally. Otherwise, the error occurs.
UDINT_TO_LINT	LINT	Convert UDINT to LINT type normally.
UDINT_TO_USINT	USINT	If input is 0 ~ 255, convert it normally. Otherwise, the error occurs.
UDINT_TO_UINT	UINT	If input is 0 ~ 65535, convert it normally. Otherwise, the error occurs.
UDINT_TO_ULINT	ULINT	Convert UDINT to ULINT type normally.
UDINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
UDINT_TO_BYTE	BYTE	Convert lower 8 bit to BYTE type.
UDINT_TO_WORD	WORD	Convert lower 16 bit to WORD type.
UDINT_TO_DWORD	DWORD	Convert UDINT to DWORD type without conversion of internal bit array.
UDINT_TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
UDINT_TO_BCD	DWORD	If input is 0 ~ 99,999,999, convert it normally. Otherwise, the error occurs.
UDINT_TO_REAL	REAL	Convert UDINT to REAL type. The tolerance may be generated during converting by the precision.
UDINT_TO_LREAL	LREAL	Convert UDINT to LREAL type. The tolerance may be generated during converting by the precision.
UDINT_TO_TOD	TOD	Convert UDINT to TOD type without conversion of internal bit array.
UDINT_TO_TIME	TIME	Convert UDINT to TIME type without conversion of internal bit array.

■ **Error**

If the error occurs, _ERR and _LER flags are set.

Note If the error occurs, outputs bits from lower bit of IN as many as the bit of output type without the conversion of internal bit array.

■ **Program example**

LD	IL
	LD %M0 JMPN ZZ LD IN_VAL UDINT_TO_TIME ST OUT_VAL ZZ :

(1) If the input condition(%M0) is On, UDINT_TO_TIME function is executed.
(2) IF the input variable IN_VAL(UDINT type) is 123, output variable OUT_VAL(TIME type) will be T#123MS.

Input(IN1) : IN_VAL(UDINT) = 123

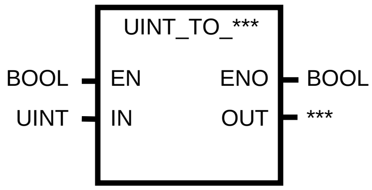
↓

Output(OUT) : OUT_VAL(TIME) =T#123MS

UINT_TO_***

UINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Unsigned Integer value to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

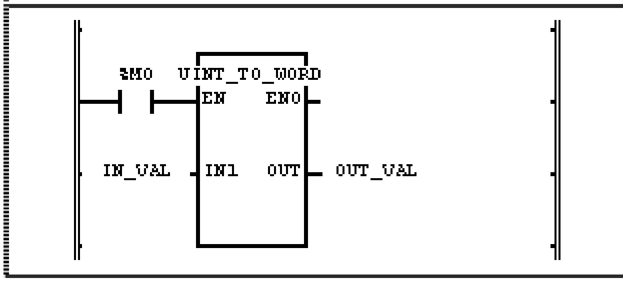
FUNCTION	Output type	Description
UINT_TO_SINT	SINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
UINT_TO_INT	INT	If input is 0 ~ 32,767, convert it normally. Otherwise, the error occurs.
UINT_TO_DINT	DINT	Convert UINT to UDINT type normally.
UINT_TO_LINT	LINT	Convert UINT to ULINT type normally.
UINT_TO_USINT	USINT	If input is 0 ~ 255, convert it normally. Otherwise, the error occurs.
UINT_TO_UDINT	UDINT	Convert UINT to UDINT type normally.
UINT_TO_ULINT	ULINT	Convert UINT to ULINT type normally.
UINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
UINT_TO_BYTE	BYTE	Convert lower 8 bit to BOOL type.
UINT_TO_WORD	WORD	Convert UINT to WORD type without conversion of internal bit array.
UINT_TO_DWORD	DWORD	Convert upper bit to WORD type filled with 0.
UINT_TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
UINT_TO_BCD	BCD	If input is 0 ~ 99,999,999, convert it normally. Otherwise, the error occurs.
UINT_TO_REAL	REAL	Convert UINT to REAL type.
UINT_TO_LREAL	LREAL	Convert UINT to LREAL type.
UNIT_TO_DATE	DATE	Convert UINT to DATE type without conversion of internal bit array.

■ **Error**

If the error occurs, _ERR and _LER flags are set.

Note If the error occurs, outputs bits from lower bit of IN as many as the bit of output type without the conversion of internal bit array.

■ **Program example**

LD	IL
	LD %M0 JMPN PO LD IN_VAL UINT_TO_WORD ST OUT_VAL PO :

(1) If the execution condition(%M0) is On, UINT_TO_WORD function is executed.

(2) IF the input variable IN_VAL(UINT type) is 255(2#0000_0000_1111_1111), OUT_VAL(WORD type) is 2#0000_0000_1111_1111.

Input(IN1) : IN_VAL(UINT) = 255

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

↓ (UINT_TO_WORD)

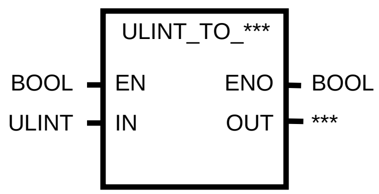
Output(OUT) : OUT_VAL(WORD) = 16#FF

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

ULINT_TO_***

ULINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Unsigned Long Integer value to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

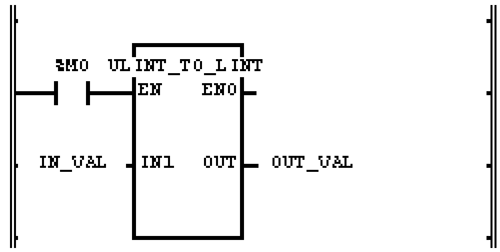
FUNCTION	Output type	Description
ULINT_TO_SINT	SINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
ULINT_TO_INT	INT	If input is 0 ~ 32,767, convert it normally. Otherwise, the error occurs.
ULINT_TO_DINT	DINT	If input is 0 ~ 2^{31}-1, convert it normally. Otherwise, the error occurs.
ULINT_TO_LINT	LINT	If input is 0 ~ 2^{63}-1, convert it normally. Otherwise, the error occurs.
ULINT_TO_USINT	USINT	If input is 0 ~ 255, convert it normally. Otherwise, the error occurs.
ULINT_TO_UINT	UINT	If input is 0 ~ 65,535, convert it normally. Otherwise, the error occurs.
ULINT_TO_UDINT	UDINT	If input is 0 ~ 2^{32}-1, convert it normally. Otherwise, the error occurs.
ULINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
ULINT_TO_BYTE	BYTE	Convert lower 8 bit to BYTE type.
ULINT_TO_WORD	WORD	Convert lower 1 bit to WORD type.
ULINT_TO_DWORD	DWORD	Convert lower 32 bit to DWORD type.
ULINT_TO_LWORD	LWORD	Convert ULINT to LWORD type without conversion of internal bit array.
ULINT_TO_BCD	BCD	If input is 0 ~ 9,999,999,999,999,999, convert it normally. Otherwise, the error occurs.
ULINT_TO_REAL	REAL	Convert ULINT to REAL type. The tolerance may be generated during converting by the precision.
ULINT_TO_LREAL	LREAL	Convert ULINT to LREAL type. The tolerance may be generated during converting by the precision.

■ **Error**

If the error occurs, _ERR and _LER flags are set.

Note If the error occurs, outputs bits from lower bit of IN as many as the bit of output type without the conversion of internal bit array.

■ **Program example**

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M0
JMPN PP
LD IN_VAL
ULINT_TO_LINT
ST OUT_VAL
PP :</td></tr>
</table>

(1) If the execution condition(%M0) is On, ULINT_TO_LINT function is executed.

(2) IF the input variable IN_VAL(ULINT type) is 123,567,899, output variable OUT_VAL(LINT type) will be 123,567,899.

Input(IN1) : IN_VAL(ULINT) = 123,567,899

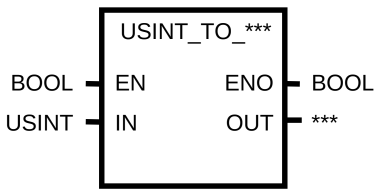
↓(ULINT_TO_LINT)

Output(OUT) : OUT_VAL(LINT) = 123,567,899

USINT_TO_***

USINT type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Unsigned Short Integer value to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

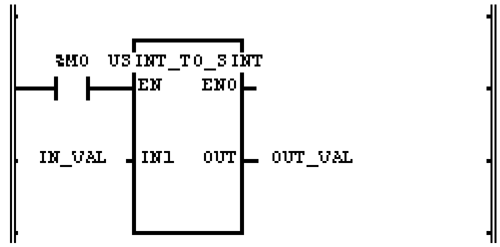
FUNCTION	Output type	Description
USINT_TO_SINT	SINT	If input is 0 ~ 127, convert it normally. Otherwise, the error occurs.
USINT_TO_INT	INT	Convert input to INT type.
USINT_TO_DINT	DINT	Convert USINT to DINT type normally.
USINT_TO_LINT	LINT	Convert input to LINT type.
USINT_TO_UINT	UINT	Convert input to UINT type.
USINT_TO_UDINT	UDINT	Convert input to UDINT type.
USINT_TO_ULINT	ULINT	Convert input to ULINT type.
USINT_TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
USINT_TO_BYTE	BYTE	Convert USINT to BYTE type without conversion of internal bit array.
USINT_TO_WORD	WORD	Convert upper bit to WORD type filled with 0.
USINT_TO_DWORD	DWORD	Convert upper bit to DWORD type filled with 0.
USINT_TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
USINT_TO_BCD	BCD	If input is 0 ~ 99, convert it normally. Otherwise, the error occurs.
USINT_TO_REAL	REAL	Convert USINT to REAL type.
USINT_TO_LREAL	LREAL	Convert USINT to LREAL type.

■ **Error**

If the conversion error occurs, _ERR and _LER flags are set.

Note If the error occurs, outputs bits from lower bit of IN as many as the bit of output type without the conversion of internal bit array.

■ **Program example**

<table>
<tr><th>LD</th><th>IL</th></tr>
<tr><td>
</td><td>LD %M0
JMPN LL
LD IN_VAL
USINT_TO_SINT
ST OUT_VAL
LL :</td></tr>
</table>

(1) If the execution condition(%M0) is On, ULINT_TO_SINT function is executed.
(2) IF the input variable IN_VAL(USINT type) is 123, output variable OUT_VAL(SINT type) will be 123.

Input(IN1) : IN_VAL(USINT) = 123(16#7B)

0	1	1	1	1	0	1	1

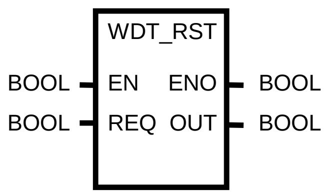
↓ (ULINT_TO_SINT)

Output(OUT) : OUT_VAL(SINT) = 123(16#7B)

0	1	1	1	1	0	1	0

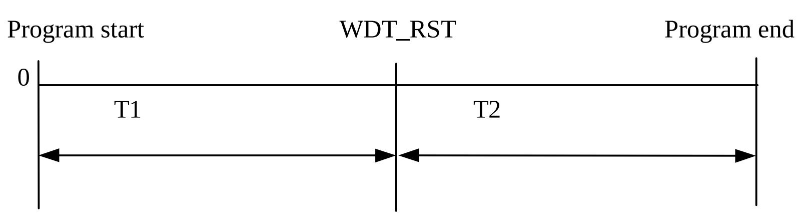
WDT_RST

Watch_Dog timer reset	Product	GM1	GM2	GM3	GM4	GM6
	Applicable	●	●	●	●	●

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Watch_Dog timer reset request **Output** ENO : Output EN value itself OUT : Output 1 after reset Watch_Dog timer

■ **Function**

- Reset Watch-Dog Timer in the program.
- The program uses WDT_RST, if the scan time may exceeds the defined Watch-Dog Time.
- When scan time exceeds Watch Dog Timer frequently, please change watch dog time set value of basic parameter section of GMWIN the programming software.
- Both T1 from 0 Line to WDT-RST function and T2 from WDT-RST function to end of program shall not exceed SCAN Watch_Dog time setting value.

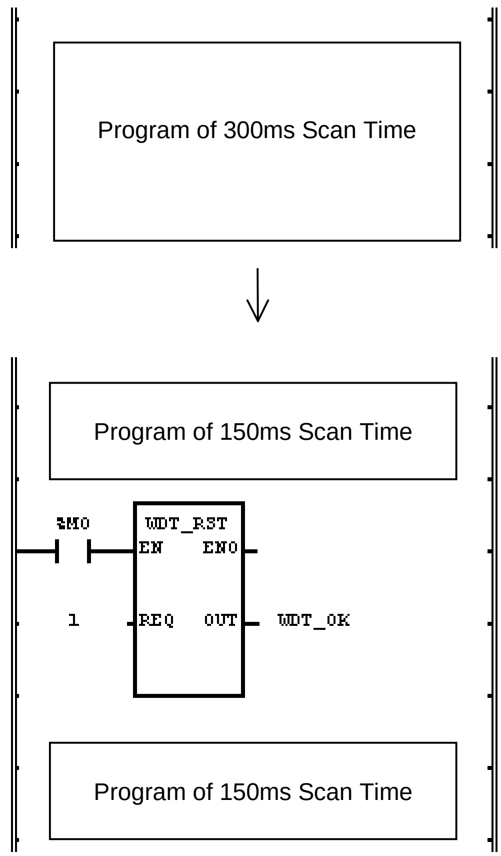


-WDT_RST function can be used several times at 1 scan.

■ **Program example**

Program that the program running time is 300ms according to the execution condition of 200ms Scan Watch_Dog time.

LD



IL

Program of 300ms Scan Time

↓

Program of 150ms Scan Time

```
LD        %M0
JMPN      FG
LD        1
WDT_RST
ST        WDT_OK
FG :
```

Program of 150ms Scan Time

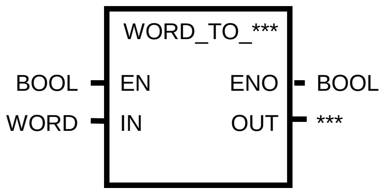
(1) If the execution condition(%M0) is On, WDT-RST(Watch Dog timer initialization) function is executed.

(2) IF WDT-RST function is executed, the program of 300ms Scan time can be run with no interference when watch dog time set value is 200ms.

WORD_TO_***

WORD type conversion

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

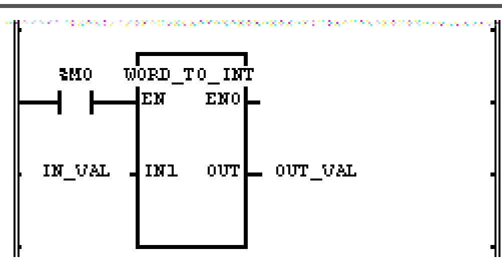
Function	Description
	**Input** EN : Execute the function in case of 1 IN : Bit array(16Bit) to be converted **Output** ENO : Output EN value itself OUT : Type converted data

■ **Function**

Convert IN to OUT data type.

FUNCTION	Output type	Description
WORD _TO_SINT	SINT	Convert lower 8 bit to SINT type.
WORD _TO_INT	INT	Convert WORD to INT type without conversion of internal bit array.
WORD _TO_DINT	DINT	Convert upper bit to DINT type filled with 0.
WORD _TO_LINT	LINT	Convert upper bit to LINT type filled with 0.
WORD _TO_USINT	USINT	Convert lower 8 bit to SINT type.
WORD _TO_UINT	UINT	Convert WORD to INT type without conversion of internal bit array.
WORD _TO_UDINT	UDINT	Convert upper bit to DINT type filled with 0.
WORD _TO_ULINT	ULINT	Convert upper bit to LINT type filled with 0.
WORD _TO_BOOL	BOOL	Convert lower 1 bit to BOOL type.
WORD _TO_BYTE	BYTE	Convert lower 8 bit to SINT type.
WORD _TO_DWORD	DWORD	Convert upper bit to DWORD type filled with 0.
WORD _TO_LWORD	LWORD	Convert upper bit to LWORD type filled with 0.
WORD _TO_DATE	DATE	Convert WORD to DATE type without conversion of internal bit array.
WORD _TO_STRING	STRING	Convert WORD to STRING type.

■ **Program example**

LD	IL
	LD %M0 JMPN P0 LD IN_VAL WORD_TO_INT ST OUT_VAL PO:

(1) If the execution condition(%M0) is On, WORD-TO-INT function is executed.

(2) IF the input variable IN_VAL(WORD type) is 2#0001_0001_0001_0001, output variable OUT_VAL(INT type) will be 4096 + 256 + 16 + 1 = 4,369.

Input(IN1) : IN_VAL(WORD) = 16#1111

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

(WORD_TO_INT)

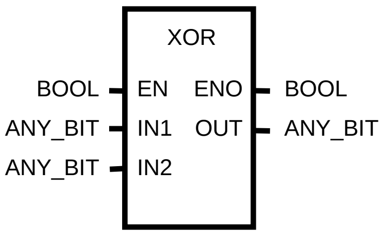
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Output(OUT) : OUT_VAL(INT) = 4,369(16#1111)

XOR

Logical XOR

Product	GM1	GM2	GM3	GM4	GM6
Applicable	●	●	●	●	●

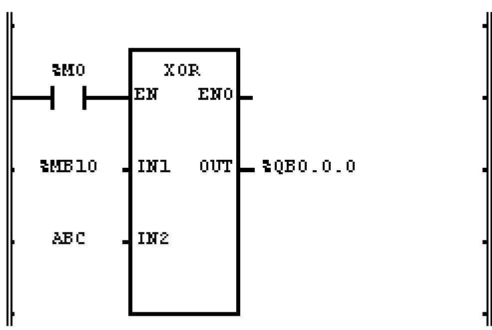
Function	Description
	**Input** EN : Execute the function in case of 1 IN1 : Value to be XOR IN2 : Value to be XOR Can be extended to 8 inputs. **Output** ENO : Output EN value itself OUT : XOR value IN1, IN2 and OUT shall be same type.

■ **Function**

Execute XOR of IN1 and IN2 and output the result to OUT.

```
IN1    1111 ..... 0000
XOR
IN2    1010 ..... 1010
OUT    0101 ..... 1010
```

■ **Program example**

LD	IL
	LD %M0 JMPN ZZ LD %MB10 XOR IN1:= CURRENT RESULT IN2:= ABC ST %QB0.0.0 ZZ :

(1) If the execution condition(%M0) is On, XOR(exclusive logical sum) function is executed.
(2) IF the input variable %MB10 is 11001100 and ABC is 11110000, XOR result of output variable %QB0.0.0 will be 00111100.

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

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

Output (OUT) : %QB0.0.0(BYTE) = 16#3C

1	1	0	0	1	1	0	0

(XOR)

1	1	1	1	0	0	0	0

0	0	1	1	1	1	0	0

8.3 MK(MASTER-K) function libraries

Each MK function library is described.

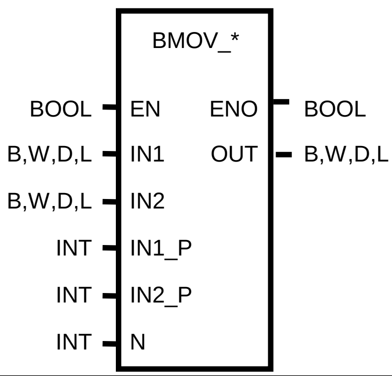
MK function libraries mean the librarized command used in Master-K series.

BMOV_B,W,D,L

Copy or Move the part of bit string

Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function



Description

Input
- EN : Execute 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

Output
- ENO : Output 1 in case of no error
- OUT : Combined bit string data output

■ Function

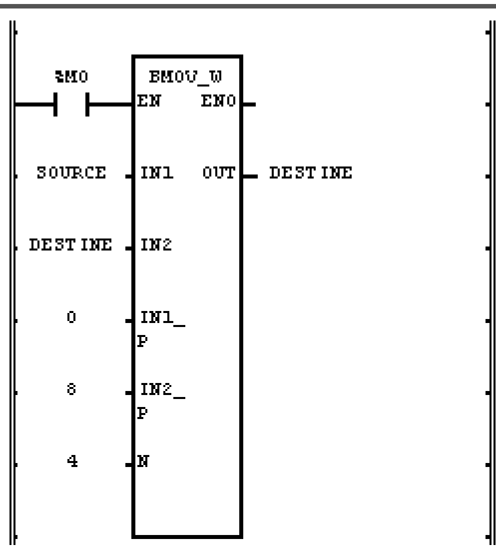
If EN is 1, take N bits from the bit position assigned by IN1_P to left direction among IN1 bit string and replace it to IN2 bit string, replaced position is assigned by IN2_P and replacement direction is from right to left.
If IN1=1111 0000 1111 0000 and IN2=0000 1010 1010 1111 and IN1_P=4 and IN2_P=8 and N=4, output data OUT=0000111110101111. The input can access to B(BYTE), W(WORD), D(DWORD) and L(LWORD) type data and L(LWORD) applies to GM1,2 only.
One of 'BMOV_B', 'BMOV _W', 'BMOV _D', 'BMOV _L' function can be selected 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

LD		%M0
JMPN		LSB
LD		SOURCE
BMOV_W	IN1:=	CURRENT RESULT
	IN2:=	DESTINE
	IN_1P:=	0
	IN_2P:=	8
	N:=	4
ST		DESTINE
LSB		

(1) If the execution condition(%M0) is On, BMOV_W function is executed.
(2) As input variable SOURCE=2#0101 1111 0000 1010 and DESTINE=2#0000 0000 0000 0000 and IN1_P=0 and IN2_P=8 and N=4, the operation result is 2#0000 1010 0000 0000 and DESTINE=2#0000 1010 0000 0000 since output is set to DESTINE.

Input (IN1) : SOURCE(WORD) = 16#5F0A

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

(IN2) : DESTINE(WORD) = 16#0000

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(IN1_P) = 0
(IN2_P) = 8
(N) = 4

(BMOV_W)

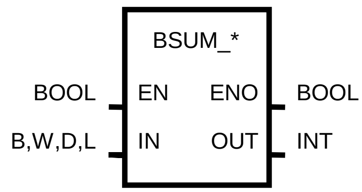
Output(OUT) : DESTINE(WORD) = 16#0A00

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

BSUM_B,W,D,L

Output ON bit number by digit

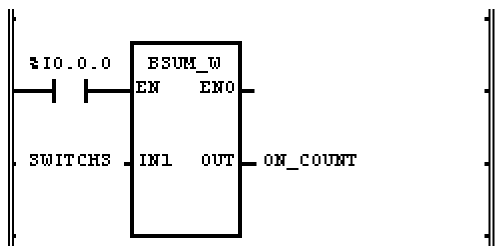
Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to detect ON bit **Output** ENO : Output EN value itself OUT : Result data sums of ON bit number

■ **Function**

If EN is 1, count Bit number of 1 among IN bit string and output the result to OUT.
The input can access to B(BYTE), W(WORD), D(DWORD) and L(LWORD) type data and L(LWORD) applies to GM1,2 only.
One of 'BSUM_B', 'BSUM_W', 'BSUM_D' and 'BSUM_L' function can be selected according to input data.

■ **Program example**

LD	IL
	LD %I0.0.0 JMPN AAA LD SWITCHS BSUM_W ST ON_COUNT AAA:

(1) If the execution condition(%I0.0.0) is On, BSUM_W function is executed.
(2) If input variable SWITCHS(WORD type)=2#0000 0100 0010 1000, output ON bit number, i.e. 3, and store integer value '3' to ON_COUNT(INT type).

Input(IN1) : SWITCHS(WORD) = 16#428

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

(BSUM_W)

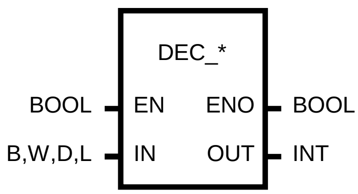
Output(OUT) : ON_COUNT(INT) = 16#3

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

DEC_B,W,D,L

Decrease IN data by 1

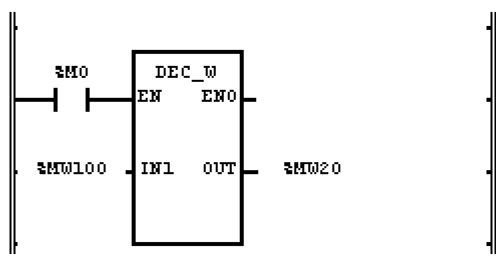
Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to be decreased **Output** ENO : Output EN value itself OUT : Result data decreased

■ **Function**

If EN is 1, decrease IN data by 1 and output the result to OUT.
Though the underflow occurs, the error is not generated and the result will be 16#FFFF if the IN is 16#0000.
The input can access to B(BYTE), W(WORD), D(DWORD) and L(LWORD) type data and L(LWORD) applies to GM1,2 only.
One of 'DEC_B', 'DEC_W', 'DEC_D' and 'DEC_L' function can be selected according to input data.

■ **Program example**

LD	IL
	LD %M0 JMPN KKK LD %MW100 DEC_W ST %MW20 KKK:

(1) If the execution condition(%M0) is On, DEC_W function is executed.
(2) If input variable %MW100=16#0007(2#0000 0000 0000 0111), %MW20=16#0006(2#0000 0000 0000 0110 after operation.

Input(IN1) : %MW100(WORD) = 16#0007

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

-1 (DEC_W)

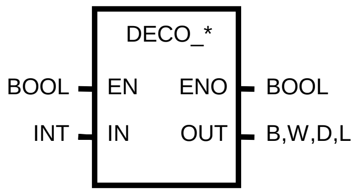
Output(OUT) : %MW20(WORD) = 16#0006

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

DECO_B,W,D,L

Set assigned bit position to ON

Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to be decoded **Output** ENO : Output 1 in case of no error OUT : Result data decoded

■ Function

If EN is 1, outputs the bit string data of OUT type with assigned bit position set to 1, the bit position is assigned by IN value.

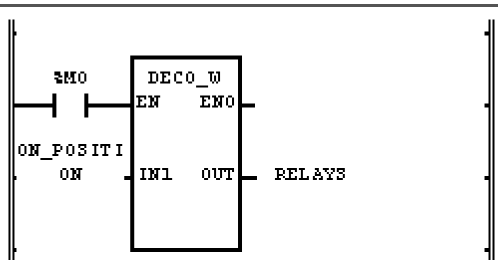
The input can access to B(BYTE), W(WORD), D(DWORD) and L(LWORD) type data and L(LWORD) applies to GM1,2 only.

One of 'DECO_B', 'DECO_W', 'DECO_D' and 'DECO_L' function can be selected according to input data.

■ Error

If input data is negative or bit location assign data exceed the bit limit of output type(over 16 in case of DECO_W), OUT will be 0 and _ERR and _LER flags are set.

■ Program example

LD	IL
	LD %M0 JMPN AAA LD ON_POSITION DECO_W ST RELAYS AAA:

(1) If the execution condition(%M0) is On, DECO_W function is executed.

(2) If input variable ON_POSITION(INT type) = 5, RELAYS(WORD type) = 2# 0000 0000 0010 0000 since only No. 5 bit of output is on.

Input(IN1) : ON_POSITION(INT) = 5(16#5)

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

↓(DECO_W)

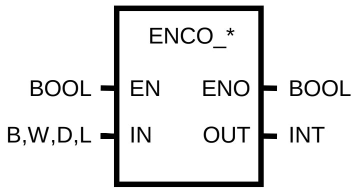
Output(OUT) : RELAYS(WORD) = 16#20

0	0	0	0	0	0	00	0	0	0	1	0	0	0	0	0

ENCO_B,W,D,L

Set assigned bit position to ON

Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to be incoded **Output** ENO : Output 1 in case of no error OUT : Result data decoded

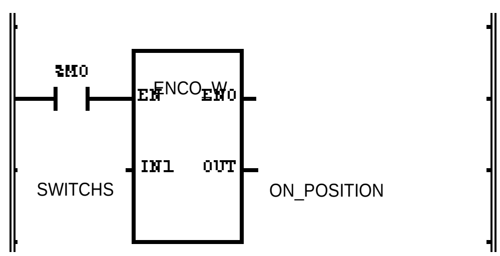
■ **Function**

If EN is 1, output to OUT the highest bit position of 1 among IN bit string. The input can access to B(BYTE), W(WORD), D(DWORD) and L(LWORD) type data and L(LWORD) applies to GM1, 2 only.
One of 'ENCO_B', 'ENCO_W', 'ENCO_D' and 'ENCO_L' function can be selected according to input data.

■ **Error**

If No bit is 1 among input data, Out turns 1, _ERR, _LER flag become three.

■ **Program example**

LD	IL
	LD %M0 JMPN AAA LD SWITCHS ENCO_W ST ON_POSITION AAA:

(1) If the execution condition(%M0) is On, ENCO_W function is executed.
(2) If SWITCHS(WORD type) = 2#000 1000 0010, output the position of 2 bits under On, in other words, '11' that is in upper position between '11' and '1', and store integer value '3' to ON_POSITION(INT type)

Input(IN1) : SWITCHS(WORD) = 16#802

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

Output(OUT) : ON_POSITION(INT) = 11(16#B)

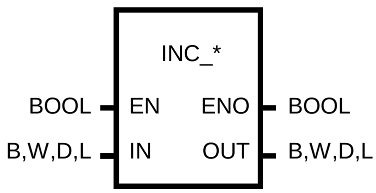
0	0	0	0	0	0	00	0	0	0	1	0	0	0	0	0

(DECO_W)

INC_B,W,D,L

Increase IN data by 1

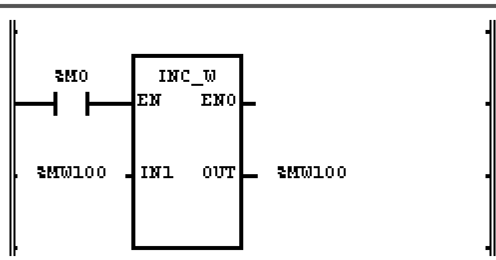
Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to be increased **Output** ENO : Output EN value itself OUT : Result data increased

■ **Function**

If EN is 1, increase IN data by 1 and output the result to OUT. There is no error when overflow occurs, the result will be 16#000 in case of 16#FFFF.
One of 'INC_B', 'INC _W', 'INC_D' and 'INC_L' function can be selected according to input data. Data types are B(BYTE), W(WORD), D(DWORD), L(LWORD), L(LWORD), is only in GM1, 2.

■ **Program example**

LD	IL
	LD %M0 JMPN BBB LD %MW100 INC_W ST %MW100 BBB:

(1) If the execution condition(%M0) is On, INC_W function is executed.
(2) If input variable %MW100=16#0007(2#0000 0000 0000 0111), %MW100=16#0008(2#0000 0000 0000 1000) after operation.

Input(IN1) : %MW100(WORD)=16#0007

Output(OUT) : %MW100(WORD)=16#0008

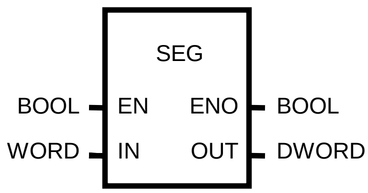
+1 (INC_W)

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

SEG

Convert BCD or HEX value to 7 segment display code

Product	GM1	GM2	GM3	GM4	GM5
Applicable	.	.	.	.	.

Function	Description
	**Input** EN : Execute the function in case of 1 IN : Input data to be converted to 7 segment code **Output** ENO : Output EN value itself OUT : Result data converted to 7 segment code

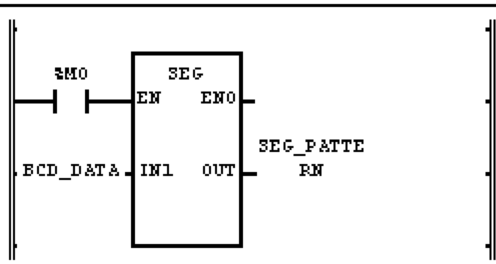
■ **Function**

If EN is 1, convert BCD or HEX(hexadecimal) digit to the code for 7 segment display as below table and output the result to OUT. The value from 0000 to 9999 can be displayed to four 7 segment in case of BCD input and the value from 0000 to FFFF can be displayed to four 7 segment in case of HEX input.

Display example

1) 4 position BCD -> 4 position 7 segment code: Use 'SEG' function
2) 4 position HEX -> 4 position 7 segment code: Use 'SEG' function
3) Integer -> 7 segment code of 4 position BCD type: Use 'SEG' function after 'INT_TO_BCD' function
4) Integer -> 7 segment code of 4 position HEX type: Use 'SEG' function after 'INT_TO_WORD' function
5) When to convert over 4 position digit,
   A) In case of BCD or HEX, split the digit by 4 position and apply 'SEG' to each 4 position fragment.
   B) Integer to 8 position BCD 7 segment code: Divide the integer by 10,000 and convert the quotient and remainder to BCD through 'INT_TO_BCD' function. After this, convert each BCD to lower 4 and upper 4 position 7 segment code.

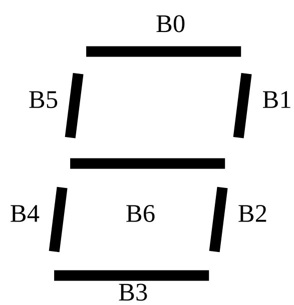
■ **Program example**

LD	IL
	LD %M0 JMPN BBB LD BCD_DATA SEG ST SEG_PATTERN BBB:

(1) If the execution condition(%M0) is On, SEC_W function is executed.
(2) If input variable BCD_DATA(WORD type) = 16#1234, output '2#00000110_01011011_01001111_01100110' of '1234' display in 7 segment display and store it in SEG_PATTERN(DWORD type).

Input(IN1) : BCD_DATA(WORD) = 16#1234	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0
↓ (SEG)																
Output(OUT) : SEG_PATTERN(DWORD) = 16#065B4F66 Upper	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	1
Lower	0	1	0	0	1	1	1	1	0	1	1	0	0	1	1	0

7 segment configuration



7 segment code conversion table

Input (BCD)	Input (Hexadecimal)	Integer value	Output B7	B6	B5	B4	B3	B2	B1	B0	Display data
0	0	0	0	0	1	1	1	1	1	1	***0***
1	1	1	0	0	0	0	0	1	1	0	***1***
2	2	2	0	1	0	1	1	0	1	1	***2***
3	3	3	0	1	0	0	1	1	1	1	***3***
4	4	4	0	1	1	0	0	1	1	0	***4***
5	5	5	0	1	1	0	1	1	0	1	***5***
6	6	6	0	1	1	1	1	1	0	1	***6***
7	7	7	0	0	1	0	0	1	1	1	***7***
8	8	8	0	1	1	1	1	1	1	1	***8***
9	9	9	0	1	1	0	1	1	1	1	***9***
	A	10	0	1	1	1	0	1	1	1	***A***
	B	11	0	1	1	1	1	1	0	0	***B***
	C	12	0	0	1	1	1	0	0	1	***C***
	D	13	0	1	0	1	1	1	1	0	***D***
	E	14	0	1	1	1	1	0	0	1	***E***
	F	15	0	1	1	1	0	0	0	1	***F***

Chapter 10 Communication function block libraries

10. Communication function block libraries

10.1. Communication function block libraries

Each communication function block library is described in this section.

Point Below function block is supported according to PLC type. (O : Available, X : Not available)

Data type(Bit size) / Product		GM1 GM2	GM3	GM4	GM5	Function block name (RD / WR)
Basic type (Max. 4)	BOOL(1)	O	O	O	O	RD(WR) BOOL
	BYTE(8)	O	O	O	O	RD(WR)BYTE
	WORD(16)	O	O	O	O	RD(WR)WORD
	DWORD(32)	O	O	O	O	RD(WR)DWORD
	LWORD(64)	O	X	X	X	RD(WR)LWORD
	USINT(8)	O	O	O	O	RD(WR)USINT
	UINT(16)	O	O	O	O	RD(WR)UINT
	UDINT(32)	O	O	O	O	RD(WR)UDINT
	ULINT(64)	O	X	X	X	RD(WR)ULINT
	SINT(8)	O	O	O	O	RD(WR)SINT
	INT(16)	O	O	O	O	RD(WR)INT
	DINT(32)	O	O	O	O	RD(WR)DINT
	LINT(64)	O	X	X	X	RD(WR)LINT
	REAL(32)	O	X	X	X	RD(WR)REAL
	LREAL(64)	O	X	X	X	RD(WR)LREAL
	TIME(16)	O	O	O	O	RD(WR)TIME
	DATE(48)	O	O	O	O	RD(WR)DATE
	TIME of DAY(48)	O	O	O	O	RD(WR)TOD
	DATE and TIME (48)	O	O	O	O	RD(WR)DT
Block (Max Fnet:120, Mnet:400 byte)		O	O	O	O	RD(WR) block

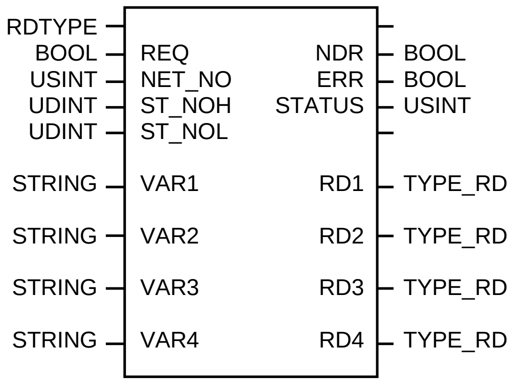
Data type(bit size) \ Product		GM1 GM2	GM3	GM4	GM5	Function block name (RD / WR)
Array Type (Within Max. 100 byte)	BOOL	O	O	O	O	RD(WR) Array
	BYTE	O	O	O	O	RD(WR)Array
	WORD	O	O	O	O	RD(WR) Array
	DWORD	O	O	O	O	RD(WR) Array
	LWORD	O	X	X	X	RD(WR) Array
	USINT	O	O	O	O	RD(WR) Array
	UINT	O	O	O	O	RD(WR) Array
	UDINT	O	O	O	O	RD(WR) Array
	ULINT	O	X	X	X	RD(WR) Array
	SINT	O	O	O	O	RD(WR) Array
	INT	O	O	O	O	RD(WR) Array
	DINT	O	O	O	O	RD(WR) Array
	LINT	O	X	X	X	RD(WR) Array
	REAL	O	X	X	X	RD(WR) Array
	LREAL	O	X	X	X	RD(WR) Array
	TIME	O	O	O	O	RD(WR) Array
	DATE	O	O	O	O	RD(WR) Array
	TIME of DAY	O	O	O	O	RD(WR) Array
	DATE and TIME	O	O	O	O	RD(WR) Array

RDTYPE(BOOL...DT)

Read data from another station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function



Description

Input
- REQ : Function block execution request at rising edge(0 1)
- NET_NO : Slot number installing the communication module for F/B transfer(0 ~ 7)
- ST_NOH : Fix to 0 during using Fnet (SAP with upper number of station communication module in remote station during using Mnet)
- ST_NOL : Station number of communication module installed in another station(Lower station number of another station during using Mnet)
- VAR1 – 4 : Direct address or variable identifier to read the data

Output
- NDR : On during receiving the data without error
- ERR : On when the error occurs after executing function block
- STATUS : Detailed code value for the error
- RD1 – 4 : Station area to store the received data from another station

■ Function and description

Read the data of another station through the communication module and store the data to specified location.
Use respective function block according to the data type to be processed.
Ex) Select "RDWORD" in function block list to process WORD(16Bit) type data.

■ ST_NOH/ST_NOL

Assign the station number of communication module.

- Fnet : ST_NOH=0(Fix), ST_NOL=Station number of another station(For example, Station 10 is 10 by decimal and 16#A by hexadecimal)
- Mnet(For Mini-MAP) : ST_NOH=SSAP(Station SAP)+DSAP(Another station SAP)+Upper station number of another communication module ST_NOL=Lower station number of another communication module

* SAP(Service Access Point) : Factor to define the service characteristics and connect the service according to upper application layer. 16#54,16#58, 16#5C,16#60 and 16#64 are supplied for the communication with GLOFA Mnet and 16#10 and 16#14 are for the communication between other company’s Mini-Map module and Mnet.

Ex 1) Communication with GLOFA Mnet
When A station reads the data of B station(remote station),
MAC address of A station : 16#00E091000000 , MAC address of B station : 16#00E091000003(Upper station No.: 00E0, Lower station No.: 91000003)
ST_NOH : 16#54 (GLOFA SAP) 58 (Another station SAP) 00E0 (Another station communication module's upper station number)
Therefore, ST_NOH=16#545800E0
ST_NOL=16#91000003 (Another station communication module's upper station number)
* MAC address is marked at the side of product.

Ex 2) Communication with other Mnet
When A station reads the data of B station(remote station)(providing Mini-Map module SAP=4E)
MAC address of A station : 16#00E091000000, Mac address of B station : 16#080070221C9A
ST_NOH : 16#10 (GLOFA SAP) 4E (Another station SAP) 0800 (Another station communication module's upper station number)
Therefore, ST_NOH=16#104E0800
ST_NOL=16#70221C9A (Another station communication module's upper station number)

■ VAR1 - VAR 4

Direct address or variable identifier of remote station to read the data.(Marked by STRING)
The data type shall be in accordance with the function block data type.(For example, VAR1 - VAR 4 data type uses WORD for "RDWORD" function block)

- Fnet :
  Direct address : Read the area of another station directly. BOOL, BYTE, WORD, DWORD and LWORD(GM1/2 only) are available.
  Ex 1) When read 100th bit area of remote station memory : '%MX100'
  Ex 2) When read input 16 point of second slot(2) of another station main base(0) : '%IW0.2.0'
  Variable identifier : The variable defined by another station(Access and resource global variable registration defining what data type process the variable identifier and where the data is assigned to) shall be used as the variable identifier to read the data of another station.
  Ex) 'PLC_1'

- Mnet
  Communication with GLOFA Mnet:
  The operation method of direct address and variable identifier is same as that of Fnet.
  Communication with other Mnet:
  Assign direct address used or supplied by other Mnet.
  (Variable identifier is not supplied.)

* Input string mark('') to the position that does not used in VAR1 ~VAR4.

■ RD1 - RD4

Assign the area to store received data from another station.
VAR1 input data is stored to RD1 and VAR2,3,4 are stored to respective RD2,3,4.
Data type shall be in accordance with that of function block.

■ NDR

Switch ON when the data are received normally after operating the function block and hold ON till the block is restarted.

■ ERR

Switch ON when the error occurs after operating the function block and hold ON till the block is restarted at next scanning. When the error occurs, the data is not received.

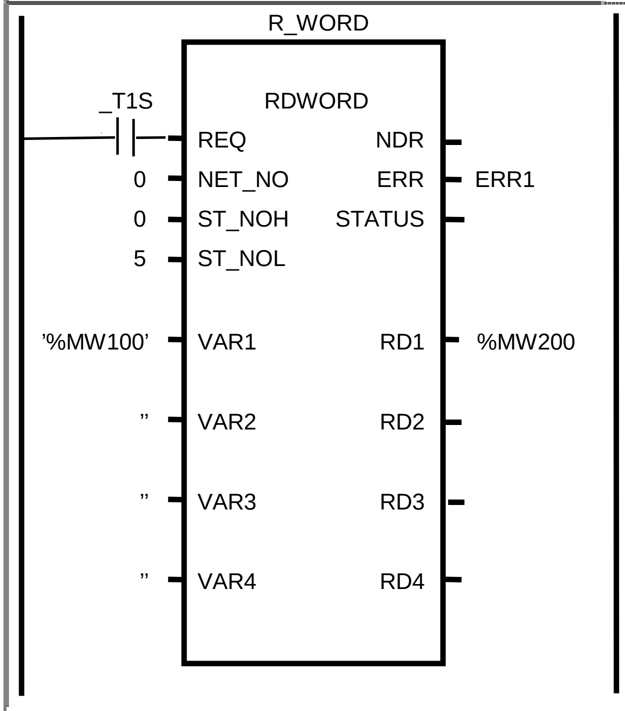
■ STATUS

Indicate the detailed code value on the error ON when the error occurs after operating the function block and hold ON till the block is restarted at next scanning. (Refer to 10-20 for code value)

■ Program example: Providing that Fnet module is installed at 0 slot)

In case that prefix number of another station is 5 and read and store %MW100 of another station area to %MW200 of local station.
(Use RDWORD function block and preset REQ condition every second)

LD

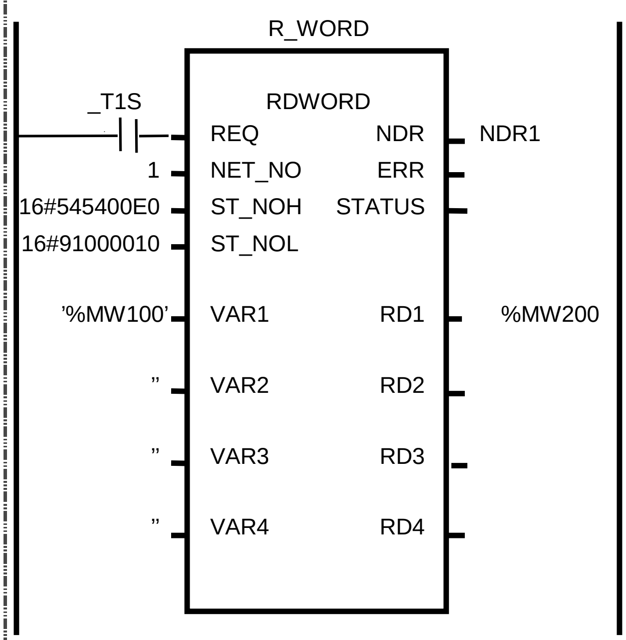


IL

```
CAL     RDWORD       R_WORD
        REQ :=       _T1S
        NET_NO :=    0
        ST_NOH :=    0
        ST_NOL :=    5
        VAR1 :=      '%MW100'
        VAR2 :=      "
        VAR3 :=      "
        VAR4 :=      "
LD                   R_WORD.ERR
ST                   ERR1
LD                   R_WORD.RD1
ST                   %MW200
```

■ **Program example : Providing that GLOFA Mini-MAP module is installed at No.1 slot.**
If MAC address of remote station is 16#00E091000010(marked at the side of Mini-MAP module) and read and store %MW100 of remote station area to %MW200 of local station.
(Use RDWORD function block and preset REQ condition every second)

LD



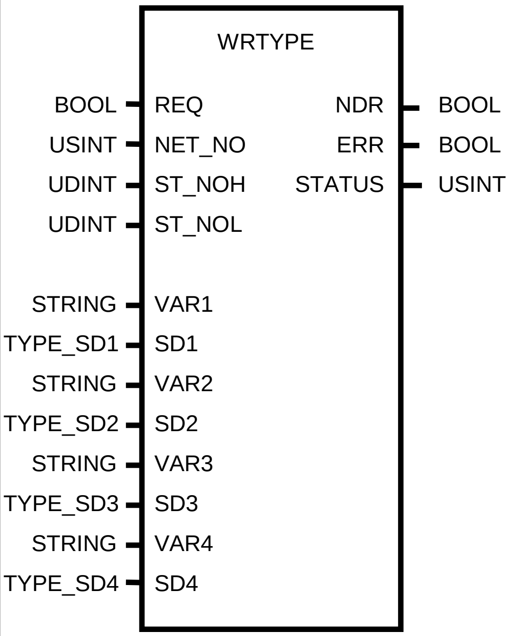
IL

```
CAL     RDWORD       R_WORD
        REQ :=       _T1S
        NET_NO :=    1
        ST_NOH :=    16#545400E0
        ST_NOL :=    16#91000010
        VAR1 :=      '%MW100'
        VAR2 :=      "
        VAR3 :=      "
        VAR4 :=      "
LD                   R_WORD.RD1
ST                   %MW200
LD                   R_WORD.NDR
ST                   NDR1
```

WRTYPE(BOOL...DT)

Write the data to remote station	

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** REQ: Function block execution request at rising edge(0 1) NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7) ST_NOH: Fix to 0 during using Fnet (with upper station number of communication module in remote station during using Mnet) ST_NOL: Station number of communication module installed in another station(Lower prefix module of remote station during using Mnet) VAR1 - 4: Direct address or variable identifier to read the data SD1-4: Data or master area which shall send to another station. **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error

■ **Function and description**

Transfer the data of station or specific data to station through the communication module of station.
Use respective function module according to the data type to be processed.
Ex) Select "WRBYTE" in function block list to process BYTE(8Bit) type data. **(Refer to "RDTYPE" function block for detailed input and output.)**

■ **ST_NOH/ST_NOL description**

Assign upper and lower station number(station number).

■ **VAR1 - VAR 4**

Direct address or variable identifier of another station to transfer the data.(Marked by STRING)
The data type shall be in accordance with the function block data type.(For example, VAR1 - VAR 4 data type shall use BYTE to use direct address in "WRBYTE" function block.)
When variable identifier is used, data type is automatically set and variable name defined by remote station(Access and resource global variable registration defining what data type process the variable identifier and where the data is assigned to) shall be used.

■ **SD1 - SD4**

Set the numeric to transfer to the remote station and the area of station. The data set to SD1 is transferred to the area of another station set to VAR1 and SD2,3,4 are transferred to respective VAR2,3,4.
Data type shall be in accordance with that of function block.

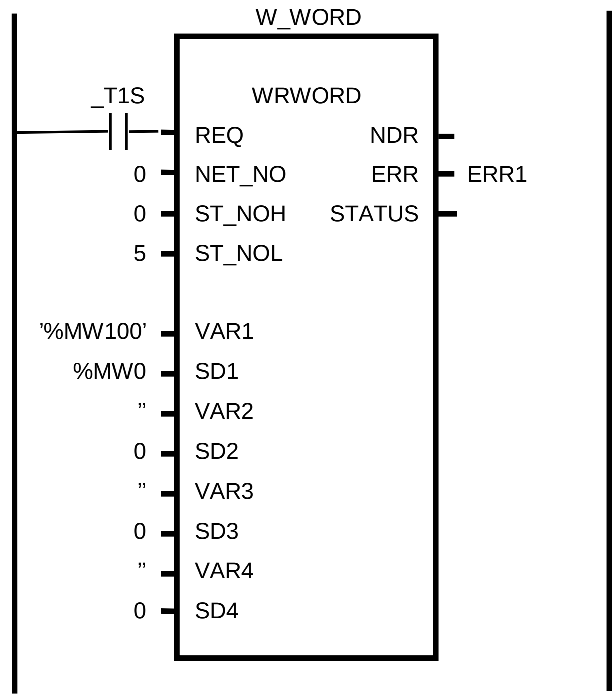
■ **NDR / ERR / STATUS**

Refer to "RDTYPE" function block.

■ **Program example : Providing that Fnet module is installed at No.0 slot.**

If %MW0 data of master station area is transferred to %MW100 of another 5 station.
(Use WRWORD function block and preset REQ condition every second)

LD



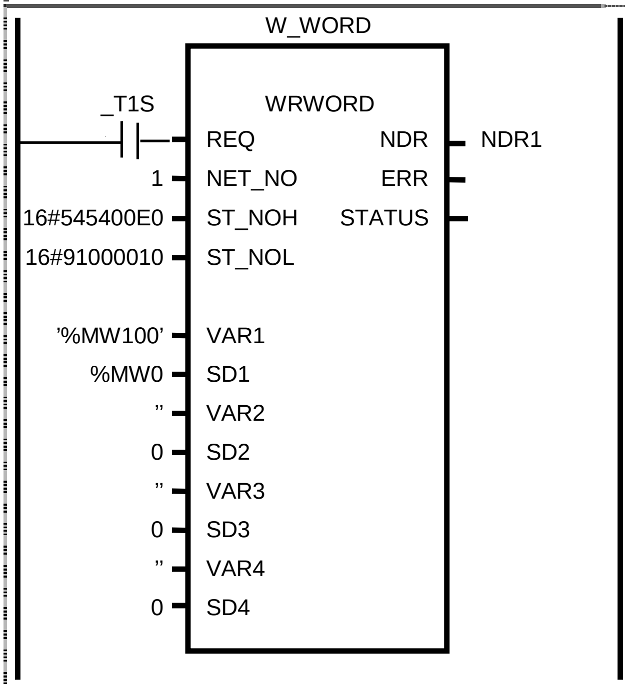
IL

```
CAL     WRWORD       W_WORD
        REQ :=       _T1S
        NET_NO :=    0
        ST_NOH :=    0
        ST_NOL :=    5
        VAR1 :=      '%MW100'
        SD1:=        %MW0
        VAR2 :=      "
        SD2:=        0
        VAR3 :=      "
        SD3:=        0
        VAR4 :=      "
        SD4:=        0
LD                   W_WORD.ERR
ST                   ERR1
```

■ **Program example : Providing that Mnet module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010 and read and transfer %MW0 of station area to %MW100 of another station.
(Use WRWORD function block and preset REQ condition every second)

LD



IL

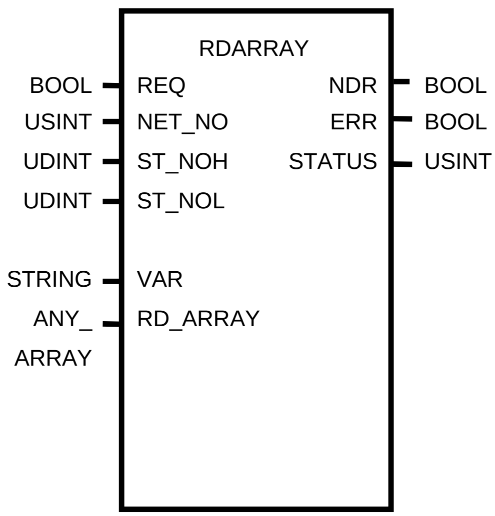
```
CAL     WRWORD       W_WORD
        REQ :=       _T1S
        NET_NO :=    1
        ST_NOH :=    16#545400E0
        ST_NOL :=    16#91000010
        VAR1 :=      '%MW100'
        SD1:=        %MW0
        VAR2 :=      "
        SD2:=        0
        VAR3 :=      "
        SD3:=        0
        VAR4 :=      "
        SD4:=        0
LD                   W_WORD.NDR
ST                   NDR1
```

RDARRAY

Read DATA of array type from another station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function



Description

Input

- REQ: Function block execution request at rising edge(0 1)
- NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7)
- ST_NOH: Fix to 0 during using Fnet (SAP with upper station number of communication module in remote station during using Mnet)
- ST_NOL: Station number of communication module installed in remote station(Lower station module of remote station during using Mnet)
- VAR: Variable identifier(defined by another station) to read the data(can not use direct address)
- RD_ARRAY: Local station area to store ARRAY data received from another station

Output

- NDR: On during receiving the data without error
- ERR: On when the error occurs after executing function block
- STATUS: Detailed code value for the error

■ **Function and description**

Function block to read the data of ARRAY type from another station.

RDARRAY can not use the direct variable of another station but can read the data by the variable name of another station. The variable name shall be assigned in the global variable list as array type. The data type shall be same to the array defined by remote station.

■ **ST_NOH/ST_NOL**

Communication module station number of another station(Refer to RDTYPE function block for details)

■ **VAR :**

Variable identifier to read from another station. Use the variable name defined by another station(Access and resource global variable registration defining what data type process the variable identifier and where the data is assigned to).

■ **NDR / ERR / STATUS**

Display the result of function block(Refer to RDTYPE function block for details).

■ **RDARRAY**

Station array area to store ARRAY data received from remote station.

The data type shall be same to the array defined by remote station.

■ **Program example : Providing that Fnet module is installed to No.0 slot.**

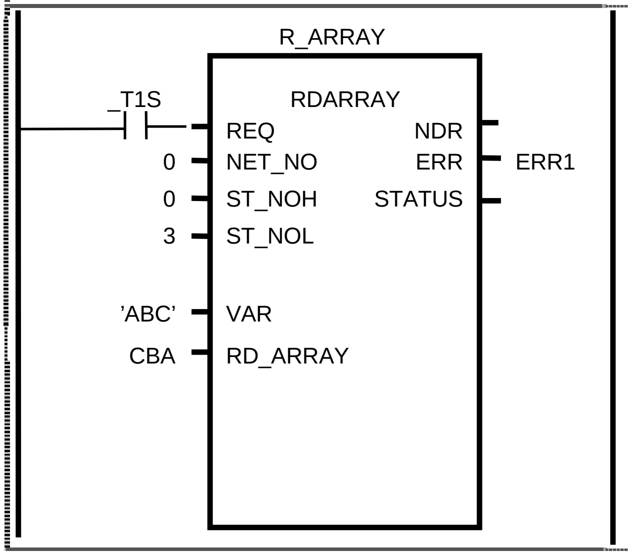
If the station number of another station is 3 and read ABC variable defined to ARRAY by another station and store it to CBA of local station array variable.
(Register the access and global variable for ABC variable at 3rd station and preset REQ condition every second)

- Variable registration example at 3rd station

Variable registration	Variable name	Access route	Description
Access variable registration	ABC	DEF(Example)	Set the access variable ABC to DEF
Resource global variable registration	RES1.DEF	-	Register DEF to global variable again(use resource 1).

* When the route name is assigned to direct variable(%I,%Q,%M) during access variable registration, resource global variable registration is not required.

LD



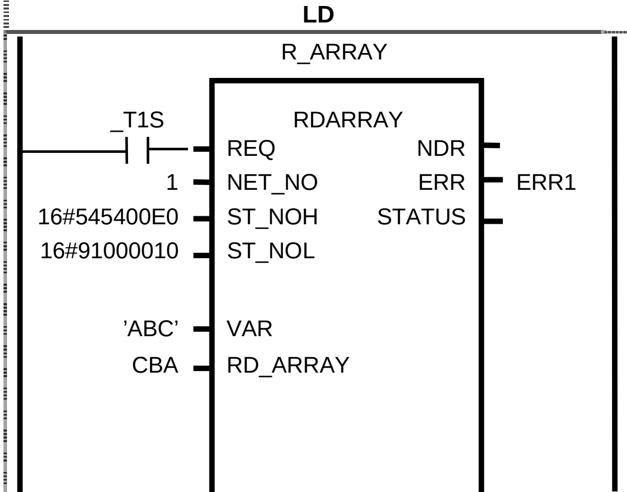
IL

```
CAL    RDARRAY        R_ARRAY
       REQ :=         _T1S
       NET_NO :=      0
       ST_NOH :=      0
       ST_NOL :=      3
       VAR :=         'ABC'
       RD_ARRAY :=    CBA

LD                    R_ARRAY.ERR
ST                    ERR1
```

■ **Program example : Providing that Mnet module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010 and read ABC variable defined as array by remote station and store it to CBA variable of station array.(Register the access and global variables for remote station ABC variable as Fnet example and preset REQ condition every second)

LD



IL

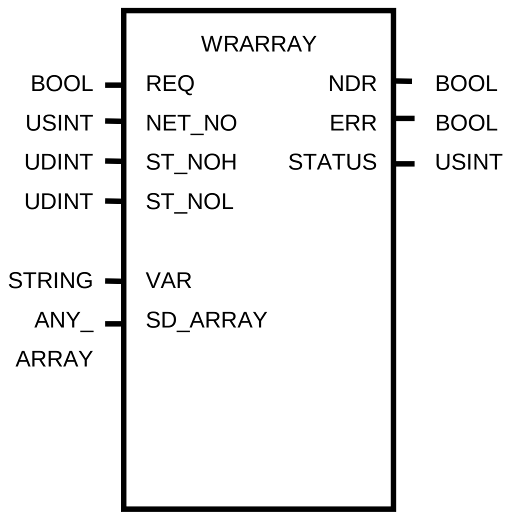
```
CAL    RDARRAY        R_ARRAY
       REQ :=         _T1S
       NET_NO :=      1
       ST_NOH :=      16#545400E0
       ST_NOL :=      16#91000010
       VAR :=         'ABC'
       RD_ARRAY :=    CBA

LD                    R_ARRAY.ERR
ST                    ERR1
```

WRARRAY

Write DATA of ARRAY type to another station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function



Description

Input
- REQ: Function block execution request at rising edge(0 1)
- NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7)
- ST_NOH: Fix to 0 during using Fnet (SAP, upper prefix number of communication module in another station during using Mnet)
- ST_NOL: Station number of communication module installed in remote station(Lower station module of another station during using Mnet)
- VAR: Variable identifier(defined by another station) to read the data(can not use direct address)
- SD_ARRAY: Local station area to store ARRAY data received from remote station

Output
- NDR: On during receiving the data without error
- ERR: On when the error occurs after executing function block
- STATUS: Detailed code value for the error

■ Function and description

Function block to transfer ARRAY data of station to the variable defined as ARRAY type by remote station. WRARRAY can not use the direct variable(e.g., %I, %Q, %M area) but can transfer the data of variable identifier used in another station. The variable name shall be assigned in the global variable list and defined as array type. The data type shall be same to the array defined by another station.(within total 100 byte)

■ ST_NOH/ST_NOL

Communication module station number of another station(Refer to RDTYPE function block for details)

■ VAR :

Variable identifier to transfer to another station. Use the variable name defined by another station(Access and resource global variable registration defining what data type process the variable identifier and where the data is assigned to).

■ SD_RDARRAY

Station array area having the data to transfer to another station.
(The data type shall be same to the array defined by another station.)

■ NDR / ERR / STATUS

Display the result of function block(Refer to RDTYPE function block for details).

■ **Program example : Providing that Fnet module is installed to No.0 slot.**

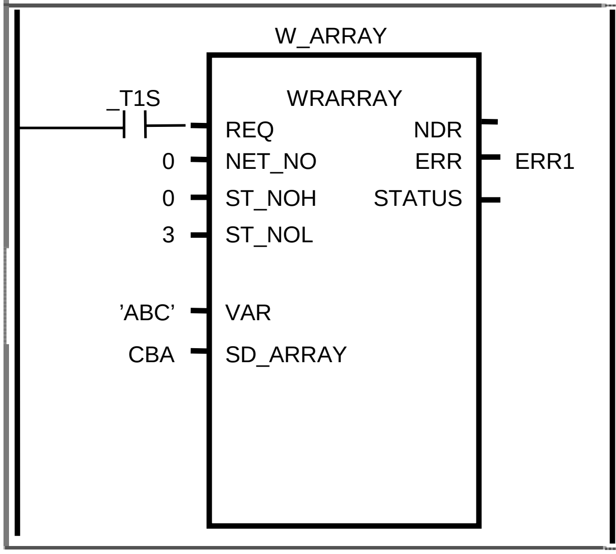
If the prefix number of another station is 3 and read ABC variable defined to ARRAY by another station and store it to CBA of station array variable.
(Register the access and global variable for ABC variable at 3rd station and preset REQ condition every second)

- Variable registration example at 3rd station

Variable registration	Variable name	Access route	Description
Access variable registration	ABC	DEF(Example)	Set the access route name of access variable ABC to DEF
Resource global variable registration	RES1.DEF	-	Register DEF to global variable again (use resource 1).

* When the route name is assigned to direct variable(%I,%Q,%M) during access variable registration, resource global variable registration is not required.

LD



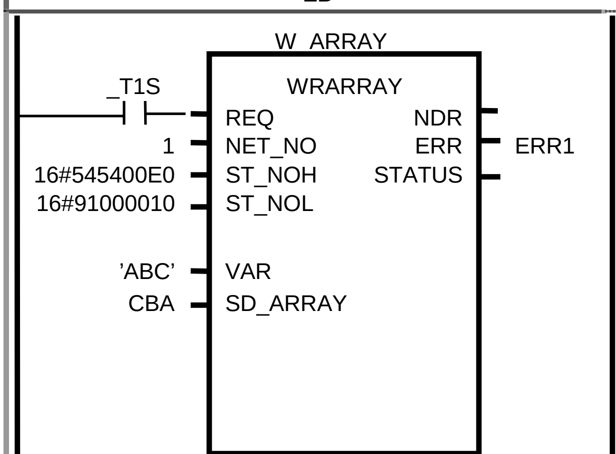
IL

```
CAL    WRARRAY        W_ARRAY
       REQ :=         _T1S
       NET_NO :=      0
       ST_NOH :=      0
       ST_NOL :=      3
       VAR :=         'ABC'
       SD_ARRAY :=    CBA

LD                    W_ARRAY.ERR
ST                    ERR1
```

■ **Program example : Providing that Mnet module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010 and transfer CBA variable of local station array to ABC variable defined as array by another station(Register the access and global variables for another station ABC variable as Fnet example and preset REQ condition every second)

LD



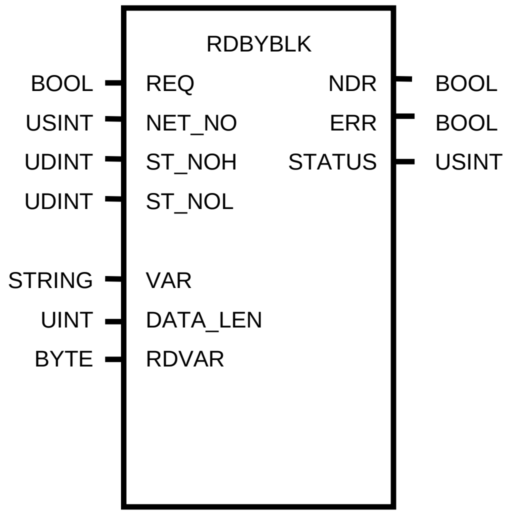
IL

```
CAL    WRARRAY        W_ARRAY
       REQ :=         _T1S
       NET_NO :=      1
       ST_NOH :=      16#545400E0
       ST_NOL:=       16#91000010
       VAR :=         'ABC'
       SD_ARRAY :=    CBA

LD                    W_ARRAY.ERR
ST                    ERR1
```

RDBYBLK

Read continuous data from station (Max. Fnet:120Byte, Mnet:400Byte)

Product	GM1	GM2	GM3	GM4	GM5
Applicable	●	●	●	●	●

Function	Description
	**Input** REQ: Function block execution request at rising edge(0 1) NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7) ST_NOH: Fix to 0 during using Fnet (SAP, upper station number of communication module in remote station during using Mnet) ST_NOL: Station number of communication module installed in another station(Lower station module of another station during using Mnet) VAR: Variable identifier(defined by another station) to read the data(can not use direct address) RDVAR: Local station block area to store the data received from another station DATA_LEN: Data to be read **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error

■ **Function and description**

Function block to read large data continuously from certain address from another station.
BYTE type can be used and only direct address(%IB, %QB, %MB) can be used for the variable.

■ **ST_NOH/ST_NOL**

Communication module station number of another station(Refer to RDTYPE function block for details)

■ **VAR :**

Only direct address can be used for start address and only BYTE type can be used.

Ex) ‘%MB100’ - From 100th byte of memory
‘%IB0.2.1’ - From 1st byte area among input area assigned to 2nd slot(2) of main base(0)
‘%QB0.3.1’ - From 1st byte area among output area assigned to 3rd slot(3) of main base(0)

■ **DATA_LEN**

Indicate data number to read from another station(Max. Fnet:120Byte, Mnet:400Byte).

■ **RDVAR**

Local station byte area to store the data read from another station.

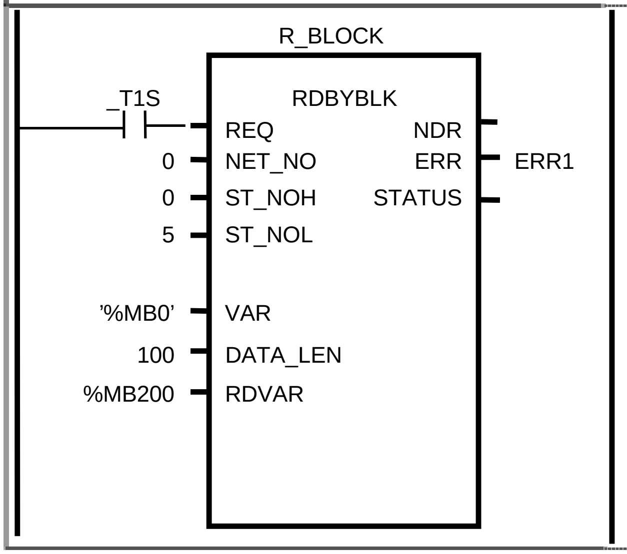
■ **NDR / ERR / STATUS**

Display the result of function block(Refer to RDTYPE function block for details).

■ **Program example : Providing that Fnet is installed to No.0 slot.**

If the station number of another station is 5 and read 100Byte from %MB0 of another station and store it to %MB200~%MB299 of local station area.
(Preset REQ condition every second)

LD



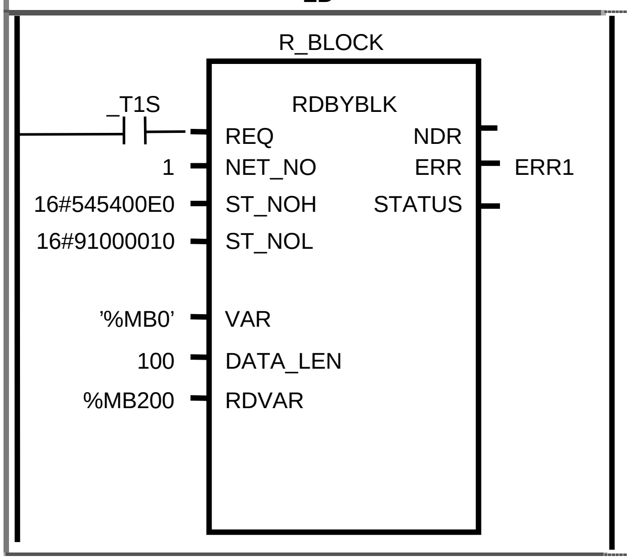
IL

```
CAL    RDBYBLK        R_BLOCK
       REQ :=         _T1S
       NET_NO :=      0
       ST_NOH :=      0
       ST_NOL :=      5
       VAR1 :=        '%MB0'
       RDVAR :=       %MB200
       DATA_LEN :=    100

LD                    R_BLOCK.ERR
ST                    ERR1
```

■ **Program example : Providing that Mnet is used and Mini-MAP module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010 and read 100Byte from %MW0 of another station and store it to %MB200~%MB299 of station area. (Preset REQ condition every second)

LD



IL

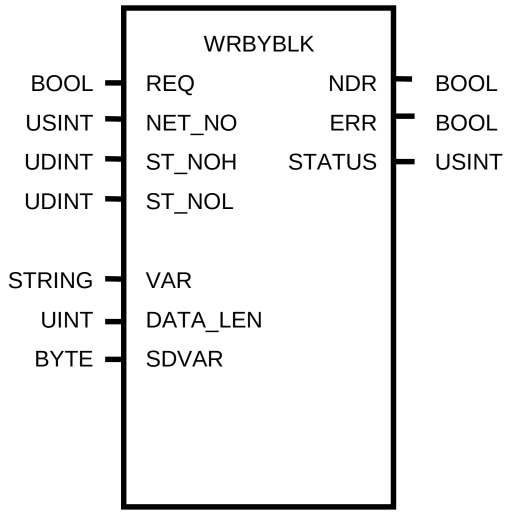
```
CAL    RDBYBLK        R_BLOCK
       REQ :=         _T1S
       NET_NO :=      1
       ST_NOH :=      16#545400E0
       ST_NOL :=      16#91000010
       VAR1 :=        '%MB0'
       RDVAR :=       %MB200
       DATA_LEN :=    100

LD                    R_BLOCK.ERR
ST                    ERR1
```

WRBYBLK

Write continuous data to remote station(Byte block) (Max. Fnet:120Byte, Mnet:400Byte)

Product	GM1	GM2	GM3	GM4	GM5
Applicable	●	●	●	●	●

Function



Description

Input
- REQ: Function block execution request at rising edge(0 1)
- NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7)
- ST_NOH: Fix to 0 during using Fnet (SAP, upper station number of communication module in remote station during using Mnet)
- ST_NOL: Station number of communication module installed in another station(Lower station module of another station during using Mnet)
- VAR: Variable identifier(defined by another station) to read the data(can not use direct address)
- SDVAR: Area that stores the data to be transferred to the another station
- DATA_LEN: Data number to be transferred

Output
- NDR: On during receiving the data without error
- ERR: On when the error occurs after executing function block
- STATUS: Detailed code value for the error

■ Function and description

Function block to write large data continuously from certain address from another station.
BYTE type can be used and only direct address(%IB, %QB, %MB) can be used for the variable.

■ ST_NOH/ST_NOL

Communication module station number of another station(Refer to RDTYPE function block for details)

■ VAR :

Only direct address can be used for start address and only BYTE type can be used.
Ex) '%MB100' - From 100th byte of memory
'%IB0.2.1' - From 1st byte area among input area assigned to 2nd slot(2) of main base(0)
'%QB0.3.1' - From 1st byte area among output area assigned to 3rd slot(3) of main base(0)

■ SDVAR

Station BYTE area that stores the data to be transferred to another station

■ DATA_LEN

Indicate data number to transfer to another station(Max. Fnet:120Byte, Mnet:400Byte).

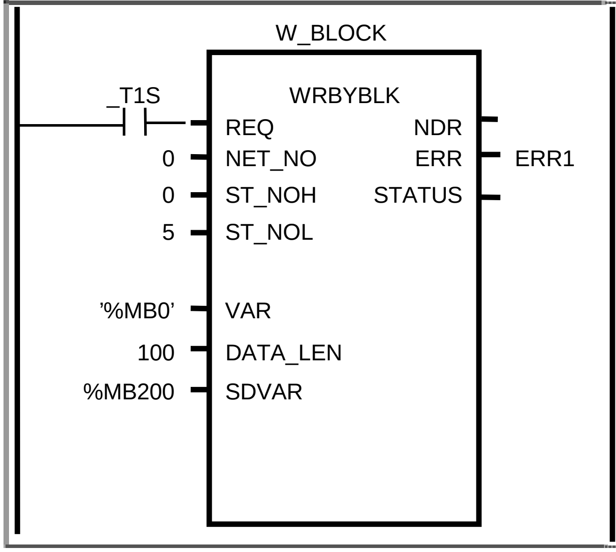
■ NDR / ERR / STATUS

Display the result of function block(Refer to RDTYPE function block for details).

■ **Program example : Providing that Fnet module is installed to No.0 slot.**

If the station number of another station is 5 and transfer the data from %MB200~%MB299 of station area to %MB0~%MB99 of remote station.
(Preset REQ condition every second)

LD



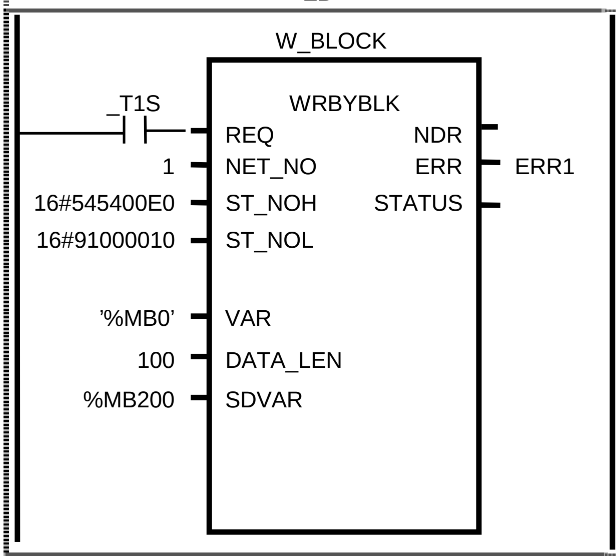
IL

```
CAL     WRBYBLK         W_BLOCK
        REQ :=          _T1S
        NET_NO :=       0
        ST_NOH :=       0
        ST_NOL :=       5
        VAR :=          '%MB0'
        SDVAR :=        %MB200
        DATA_LEN :=     100

LD                      W_BLOCK.ERR
ST                      ERR1
```

■ **Program example : Providing that Mnet module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010 and transfer the data from %MB200 to %MB299 of station area to %MB0~%MB99 of another station.
(Preset REQ condition every second).

LD



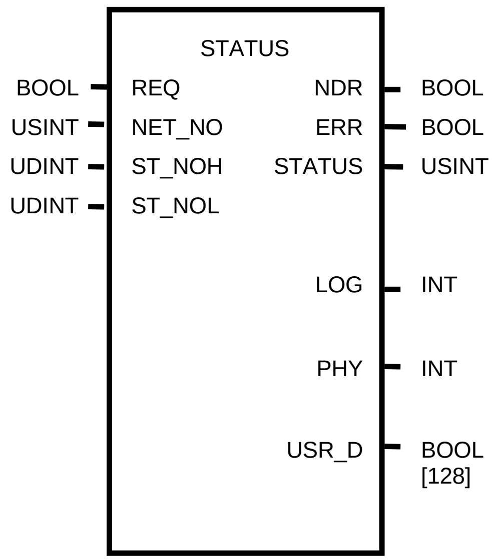
IL

```
CAL     WRBYBLK         W_BLOCK
        REQ :=          _T1S
        NET_NO :=       1
        ST_NOH :=       16#545400E0
        ST_NOL :=       16#91000010
        VAR :=          '%MB0'
        SDVAR :=        %MB200
        DATA_LEN :=     100

LD                      W_BLOCK.ERR
ST                      ERR1
```

STATUS

Read the status of remote station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	●	●	●	●	●

Function	Description
	**Input** REQ: Function block execution request at rising edge(0 1) NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7) ST_NOH: Fix to 0 during using Fnet (SAP, upper station number of communication module in another station during using Mnet) ST_NOL: Station number of communication module installed in another station(Lower station module of another station during using Mnet) **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error LOG: Function level to use for communication service PHY: H/W operation status of another station PLC USR_D: Total data of another station PLC status

■ Function and description

Function block to check the status of another station.

■ ST_NOH/ST_NOL

Communication module station number of another station(Refer to RDTYPE function block for details)

■ LOG

Indicate the function level to use for communication service(Logical State)

0 = State-Change-Allowed

■ PHY

Indicate H/W operation status of PLC by the physical state.

0 = Operational(in use)
1 = Partially-Operational-H/W(PLC and modules are not operated normally)
2 = Inoperable-H/W(Stop due to the error)
3 = Need-Commission-H/W(Unreliable data though it is used)

■ USR_D

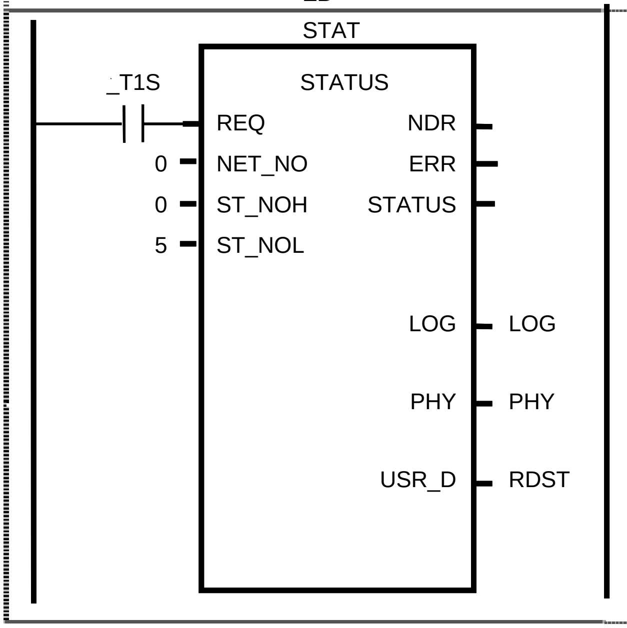
Supply 128Bit of Bit array for general status of another station PLC.
([0]~[64] are used and others are reserved.)

■ **Program example : Providing that Fnet module is installed to No.0 slot.**

In case of reading general information for remote station 5.

(Providing that REQ condition is set every second and RDST variable is declared as BOOL ARRAY[64])

LD



IL

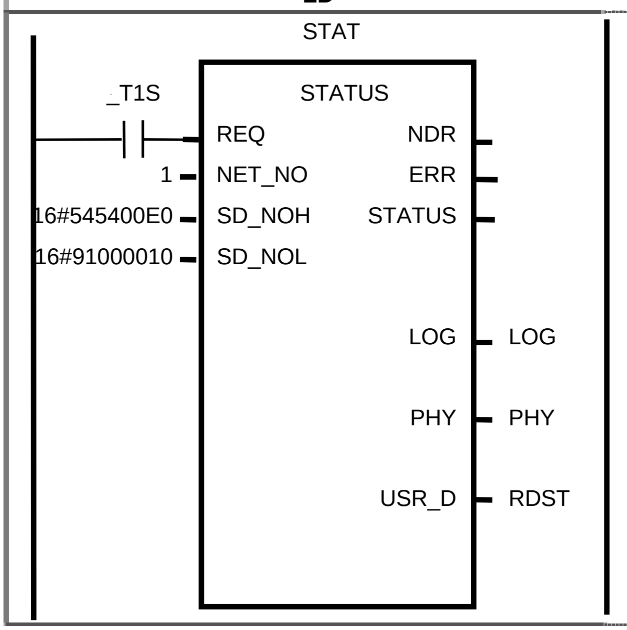
```
CAL     STATUS        STAT
        REQ :=        _T1S
        NET_NO :=     0
        ST_NOH :=     0
        ST_NOL :=     5
LD                    STAT.LOG
ST                    LOG
LD                    STAT.PHY
ST                    PHY
LD                    STAT.USR_D
ST                    RDST
```

■ **Program example : Providing that Mnet module is installed to No.1 slot.**

If MAC address of another station is 16#00E091000010(marked at the side of Mini-MAP module) and read general information of remote station.

(Providing that REQ condition is set every second and RDST variable is declared as BOOL ARRAY[64])

LD



IL

```
CAL     STATUS        STAT
        REQ :=        _T1S
        NET_NO :=     1
        ST_NOH :=     16#545400E0
        ST_NOL :=     16#91000010
LD                    STAT.LOG
ST                    LOG
LD                    STAT.PHY
ST                    PHY
LD                    STAT.USR_D
ST                    RDST
```

BIT array description stored to USR_D in STATUS function block: Declared from [0] to [64]

Bit number	Representat-ive content	Bit classification	Detailed content	Description
S[0]~S[7]	CPU_TYPE	0x00	GM1	Display CPU type by S[7]~S[0] value.
		0x01	GM2	
		0x02	GM3	
		0x03	GM4	
		0x04	GM5	
		0x05	GM3_FSM	
		0x06	GM4_FSM	
		0x07	SRU	
		0x08	FAM	
		0x09	PMU500	
		0x0A	PADT	
		0x22	GT3	
		0x23	GK4	
		0x24	GK5	
		0x25	GK3_FSM	
		0x26	GK4_FSM	

S[8]~ S[15]	_VERSION_ NO	S[8]~S[11]	_VERSION_NO Lower mark	Ex) If v3.1 is displayed,
		S[12]~S[15]	_VERSION_NO Upper mark	(1: S[11]~S[8] are displayed by decimal, 3: S[15]~S[12] are displayed by decimal)

S[16]	_SYS _STATE		Local control	Display whether operation mode can be changed by mode key or PADT.
S[17]			STOP	Display CPU operation status.
S[18]			RUN	Display CPU operation status.
S[19]			PAUSE	Display CPU operation status.
S[20]			DEBUG	Display CPU operation status.
S[21]			Operation mode change cause	Operation mode change by key
S[22]			Operation mode change cause	Operation mode change by PADT
S[23]			Operation mode change cause	Operation mode change by remote PADT
S[24]			Operation mode change cause	Operation mode change by communication
S[25]			STOP by STOP function	SCAN end and STOP by STOP function under RUN mode operation

Bit number	Representative content	Bit classification	Detailed mark	Description
S[26]	SYS_STATE		Forced input	Display forced ON/OFF to input contact
S[27]			Forced output	Display forced ON/OFF to output contact
S[28]			STOP by ESROP function	Prompt stop by ESTOP function under RUN mode operation
S[29]			No meaning	
S[30]			Monitoring	Display the external monitoring for the program and variable
S[31]			Remote mode ON	Display the operation under remote mode
S[32]	_PADT_CNF		Local PADT connection	Bit indicating status of local PADT connection
S[33]			Remote PADT connection	Bit indicating status of remote PADT connection
S[34]			Remote communication connection	Bit indicating contact status of remote communication
S[35]	_DOMAIN_ST		Abnormal main parameter	Flag that checks and displays the abnormal of main parameter
S[36]			Abnormal I/O parameter	Flag that checks and displays the abnormal of I/O configuration parameter
S[37]			Abnormal program	Flag that checks and displays the abnormal of user program
S[38]			Abnormal access variable	Flag that checks and displays the abnormal of access variable
S[39]			Abnormal high-speed link parameter	Flag that checks and displays the abnormal of high-speed link parameter
S[40]	_CPU_ER		CPU configuration error	Error flag generated when CPU module is not operated normally due to the self diagnosis of CPU module, wrong installation of Base CPU, multi CPU configuration (refer to _SYS_ERR for details).
S[41]	_IO_ER		Module type inconsistency error	Representative flag. Detects and displays that I/O configuration parameter of each slot is different from the installed module or certain module is installed in the slot that can not be installed. (Refer to _IO_TYER_N and _IOTYER[n]).
S[42]	_IO_TYER		Module installation error	Representative flag. Detects and displays that module configuration of each slot is changed during operation. (Refer to _IO_DEER_N and _IO_DEER[n]).
S[43]	_FUSE_ER		Fuse cut-off error	Representative flag. Detects and displays that the fuse installed in module of each slot is broken. (Refer to _FUSE_ER_N and _FUSE_ER[n]).
S[44]	_IO_RWER		Read/Write error of input/output (trouble)	Representative flag. Detects and displays the error that can not read or write the I/O module of each slot. (Refer to _IP_RWER_N and _IO_RWER[n]).

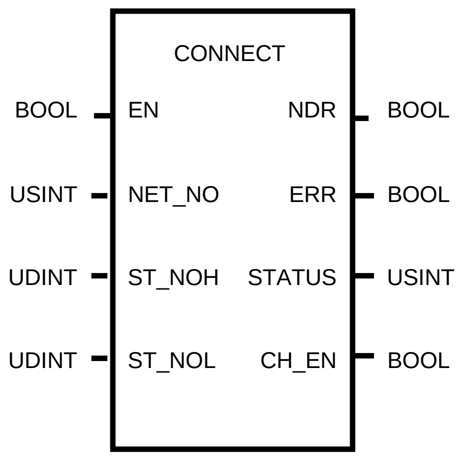
Bit number	Representative content	Bit classification	Detailed mark	Description
S[45]	_SP_IFER		Special/Communication module interface error(trouble)	Representative flag. Detects and displays the failure of special or communication module initialization of each slot or disable of normal interface due to malfunction of module. (Refer to _IP_IFER_N and _IP_IFER[n]).
S[46]	_ANNUN_ER		Trouble detection error of external device	Representative flag. Displays the trouble detection generation when the trouble of external device is recorded in _ANC_ERR[n] by user program.
S[47]	Not used			
S[48]	_WD_ER		SCAN WATCH-DOG error	Error generated when the scan time exceeds SCAN WATCH-DOG TIME set by parameter.
S[49]	_CODE_ER		Program code error	Error generated when the user program can not decode the command.
S[50]	_STACK_ER		STACK OVERFLOW error	Error generated when the program stack exceeds normal range during running the program.
S[51]	_P_BCK_ER		Program error	Error of the program memory damage or execution disable due to abnormal program.
S[52]	_RTC_ERR		System alarm RTC data abnormality	Flag displaying the data abnormality of RTC
S[53]	_D_BCK_ER		Data backup abnormality	If Hot/Warm restart program can not be executed since the data memory is damaged due to BACK_UP abnormality but Cold restart is executed, this flag indicates this problem and used in the initialization program. It will be automatically reset when initialization program is completed.
S[54]	_H_BCK_ER		Hot restart disable error	If restart program(Warm/Cold) is executed by the parameter since the recover of electricity exceeds hot restart time or the operation data for hot restart is not back-up normally, the flag indicates this problem and used in the initialization program. It will be automatically reset when initialization program is completed.
S[55]	_AB_SD_ER		Abnormal shutdown	When the program is stopped by the cutoff of power supply and restarted by warm restart, the flag indicates the operation error and used in the initialization program. It will be automatically reset when initialization program is completed. Indicates that the program is stopped by 'ESTOP' function.

Bit number	Representative content	Bit classification	Detailed mark	Description
S[56]	_TASK_ERR		Task conflict(Normal cycle, external task)	Flag indicates the task conflict when the tasks are duplicated during user program. (Refer to _TC_BMAP[n], _TC_CNT[n] for details.)
S[57]	_BAT_ERR		Battery trouble	Flag detects and indicates that the battery voltage is lower than rated value for user program and data memory back-up.
S[58]	_ANNUM_WR		Detect light trouble of external device	Representative flag indicates the trouble when the user program detect the alarm of external device and records it to _ANC_WB[n].
S[59]	Not used			
S[60]	Not used			
S[61]	_HSPMT1_ER		Over high-speed link parameter 1	Representative flag indicates high-speed link disable after checking the parameter of high-speed link. It will be reset during high-speed link disable.
S[62]	_HSPMT2_ER		Over high-speed link parameter 2	
S[63]	_HSPMT3_ER		Over high-speed link parameter 3	
S[64]	_HSPMT4_ER		Over high-speed link parameter 4	

CONNECT

Establish logical communication channel to another station (For connection with other Mnet)

Product	GM1	GM2	GM3	GM4	GM5
Applicable	●	●	●		

Function	Description
	**Input** REQ: Function block execution request at rising edge(0 1) NET_NO: Slot number installing the communication module for F/B transfer(0 ~ 7) ST_NOH: Fix to 0 during using Fnet (SAP, upper station number of communication module in another station during using Mnet) ST_NOL: Station number of communication module installed in forced station(Lower station module of another station during using Mnet) **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error CH_EN: Result of channel establishment

■ Function and description

The communication to another station in Mnet is executed after establishing communication channel and SAP is required for the connection. SAP is classified by SSAP and DSAP and the manufacturer of Mini-MAP communication module supplies SSAP to the user.
The type of SAP is divided by Association SAP, Associationless SAP, Unspec.SAP.

- Association SAP
  Connect after establishing the channel by channel service during communication(Initiate) service.
- Associationless SAP
  Execute the communication assuming the communication channel is established internally without communication channel(Initiate) service.
- Unspec. SAP
  Satisfy both of Association SAP and Associationless SAP function.

If SAP supplied by other company is Association SAP or the communication shall be executed by the communication channel(Initiate) service, the communication channel(INITIATE) shall be established by CONNECT function block. However, for the communication with our product, this function block is not required.
The function block is operated not the edge but the level. Therefore, if the channel is established, the channel is kept when EN input level is "1" and CH_EN bit is set as "1". When the channel is released by the request, CH_EN bit is cleared to "0" and the user can use CH_EN bit for other function block.

■ EN

EN shall be operated when level is "1" and holds "1" in service. (BOOL)

■ NET_NO

Slot location where communication module to transfer the data by this FB among communication module in main base is installed. (0~7)

■ **ST_NOH**

Upper prefix number of communication module installed in another station for channel establishment and SAP.
ST_NOH=SSAP+DSAP+Upper station number of another communication module

■ **ST_NOL**

Lower station number of communication module installed in another station for channel establishment.
ST_NOL=Lower station number of another station

Ex) Connection to other Mnet

When the communication channel is established from A station(our company) to B station(other company) (Providing other Mini-Map module SAP=4E)
MAC address of A station: 16#00E091000000(our company),
MAC address of B station: 16#080070221C9A(other company)
ST_NOH: 16#10 (SSAP) 4E (DSAP:) 0800 (Upper station number of another communicat-ion module)
Therefore, ST_NOH = 16#104E0800
ST_NOL = 16# 70221C9A (Lower station number of another communication module)

- SAP(SSAP) provided by GLOFA Mini-MAP module for the connection with other Mimi-MAP is divided by 16#10 and 16#14.

■ **NDR**

When the function block is operated and ended normally, NDR is on till this function block is operated at next scan.

■ **ERR**

When the error occurs after operating function block or the channel release request is received from remote station, ERR is on till this function block is operated at next scan.

■ **STATUS**

Indicate detailed code value on the error and holds the value till this function block is operated at next scan.

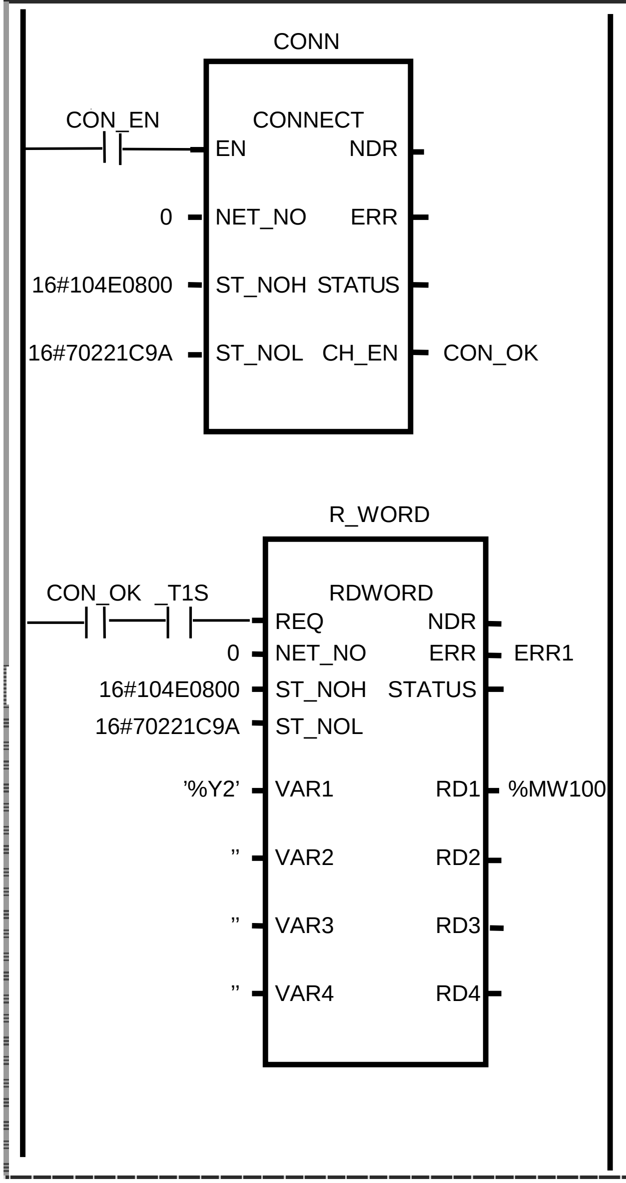
■ **CH_EN**

When the channel is established, CH_EN is "1" and when the channel is released, CHEN holds "0". When NDR is set to "1", CH_EN becomes "1" and holds "1" though NDR bit is cleared next. When ERR bit is set to "1", CH_EN bit is set to "0" and holds "0" till the channel is established.

■ **Program example : Providing that Mnet is used and communicated with other Mnet and Mini-MAP module is installed to No.0 slot.**

If MAC address of other company is 16#080070221C9A and read Y2 as 1 WORD(16Bit) after establishing the channel for other Mini-MAP and store it %MW100 area of station.

LD



IL

```
CAL     CONNECT      CONN
        EN :=        CON_EN
        NET_NO :=    0
        ST_NOH :=    16#104E0800
        ST_NOL :=    16#70221C9A
LD                   CONN.CH_EN
ST                   CON_OK

LD                   CON_OK
AND                  _T1S
ST                   RD_REQ
CAL     RDWORD       R_WORD
        REQ :=       RD_REQ
        NET_NO :=    0
        ST_NOH :=    16#104E0800
        ST_NOL :=    16#70221C9A
        VAR1 :=      '%Y2'
        VAR2 :=      ''
        VAR3 :=      ''
        VAR4 :=      ''
LD                   R_WORD.ERR
ST                   ERR1
LD                   R_WORD.RD1
ST                   %MW100
```

* CON_EN is variable of initial value 1.

STATUS code value of function block and description

1) Error received from communication module

Value (Decimal)	Description
0	– OK(Success : No Error)
1	– Physical Layer Error of LINK(Send/Receive disable) – Cause of station Error and power-off of another station, station number wrong description and trouble.
	Receive (-) response from another station. Classified by the error.
33	– Can not find variable identifier(Object Undefined). – Not defined in the access variable area.
34	– Invalid Address- Struct error described in the specification of communication module and out of range.
50	– Invalid Response – When the response is not received as requested or CPU trouble of another station
113	– Object Access Unsupported – VMD Specific, Symbolic Address – Maximum value out
114	– Access variable is not downloaded(Object non Existent) – Download disable in the access variable area
187	– Receive the error code except assigned code(Communication code value of other company)- Receive the code except defined error code
3	– Identifier of function block to be received does not exist in the communication channel. – Value not used in our company.
4	– Data Type Mismatch
5	– Receive reset from other station – Value not used in our company.
6	– Function block of another station is not ready.(Receiver Not Enabled) – Value not used in our company.
7	– Function block status of station is wrong. (Remote Device in Wrong State) – Value not used in our company.
8	– OBJECT requested by user can not be accessed. (Access Denied to Remote Object) – Value not used in our company.
9	– Receive disable due to excessive function block of another station (Receiver Overrun) – Value not used in our company.

Value (Decimal)	Description
10	– Response wait time out(Time out) – If the response is not received from another station in certain period
11	– Struct error
12	– Abort(Local/Remote) – Disconnect the connection due to serious error.
13	– Reject(Local/Remote) – Unsuitable format for MMS or error due to noise
14	– Communication channel establishment error(Connect/Disconnect) – Error relating logical communication channel establishment at the service on PI/DOMAIN/GEN and communication with other communication module.(For Mnet only)
15	– High-speed communication and connection service error

2) Error in CPU

Value (Decimal)	Description
16	If the location of computer communication module is wrong
18	Input parameter set error
20	If response frame not requested is received
21	If the response is not received from computer communication module

3) Error relating remote function block(FSM)

Value (Decimal)	Description
128	FSM power error
129	BASE(Rack) number error
130	Slot number error
131	Module information error
132	Data range error(Invalid Range)
133	Inconsistency of data type
134	IP module is not ready
135	Read/Write error of IP module
136	Access failure(Bus access error)
137	Error except assign code

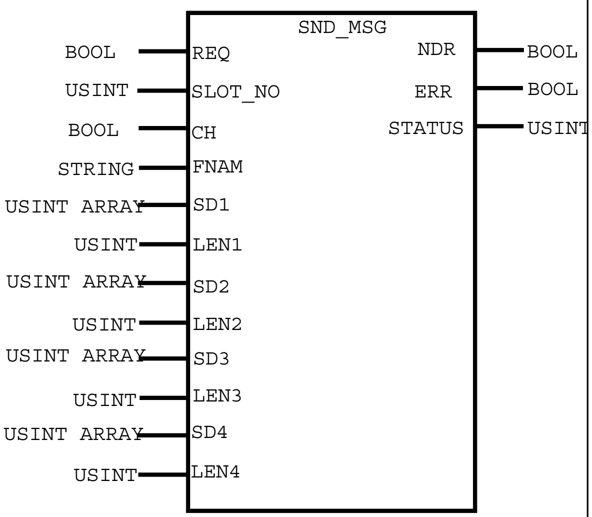
10.2 Computer link module function block libraries

Describes the function block that controls the frame, arranged by frame editor, in PLC Program example
Program that set the output contact %00.3.0 when No. 5 pulse inflows to input contact %I0.1.14.
Function block for CLM is SND_MSG and RCV_MSG.

SND_MSG(Send Message)

Send the data to another station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** REQ: Function block execution request at 1(rising edge) SLOT_NO: Slot number input stalling CLM CH: Channel number input to send the data 0: RS-232-C, 1: RS-422 FNAM: Frame name input to be sent SDx: Array variable name input to be sent LENx: Each array length input to be sent In case of no data to be sent, input 0. (x: 1,2,3,4) **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error

■ Function

Send the frame downloaded to computer link module(CLM) with input data of variable through the channel assigned to CLM.

Before executing this function block, download the frame of same name to the frame of 'FNAM' by using the frame editor. Further, input SDx of same number to array variable used in the frame. Refer to G3L-CUEA and G4L-CUEA technical document for frame download method.

■ Error

(1) Status value in CPU (Decimal)

STATUS value	Meaning	Treatment
16	CLM is located wrongly.	Input exact slot value.
20	1) Wrong library is used. 2) Response frame is wrong.	1) Check the data of library.(Check that [Communi.fb] file is dated after OCT, 1996) 2) Check again the receive frame of local station and send frame of remote station.
21	The response is not received from CLM.(Wait time out)	Check the operation mode of CLM. Check whether CLM is user defined communication status.

(2) Status value in CLM (Decimal)

STATUS value	Meaning	Treatment
64	Channel status of RS-232C/422 is not run.	Run the operation by frame editor. (Menu: On-line operation conversion)
65	The frame name used in frame editor does not match with that used in function block.	Make same the frame name used in frame editor to that used in function block.
66	Frame name can not be found due to CPU trouble.(For send)	1) Download again the frame. 2) Check the CPU.
67	The frame assigned to FNAM is not received from remote station.	1) Check the receive frame again. 2) Check the send frame of remote station.
68	The frame is not downloaded from the frame editor.	Download the frame.
69	ASCII <-> HEX conversion error	Check whether the receive data is ASCII or HEX.
70	Array size assigned by frame editor does not match with that (size assigned to LENx) used in function block.	Check the data size and fix it. (Data size is Byte value)
100	Array type assigned to SDx or RDx is different.	Set the array type to unsigned integer.
102	The frame name does not exist in FNAM.	1) Check the frame name again. 2) Download the frame again.
103	The frame definition is wrong.	1) Check respective frame content by frame editor. 2) Download the frame again.
104	The frame is not downloaded by the frame editor.	Download the frame
115	The operation mode is not the user defined communication mode.	Fix all switches exactly. User defined communication RS-232C: 0,2,4(0:Continuous mode) RS-422/485: 2,5,6

■ **Program example**

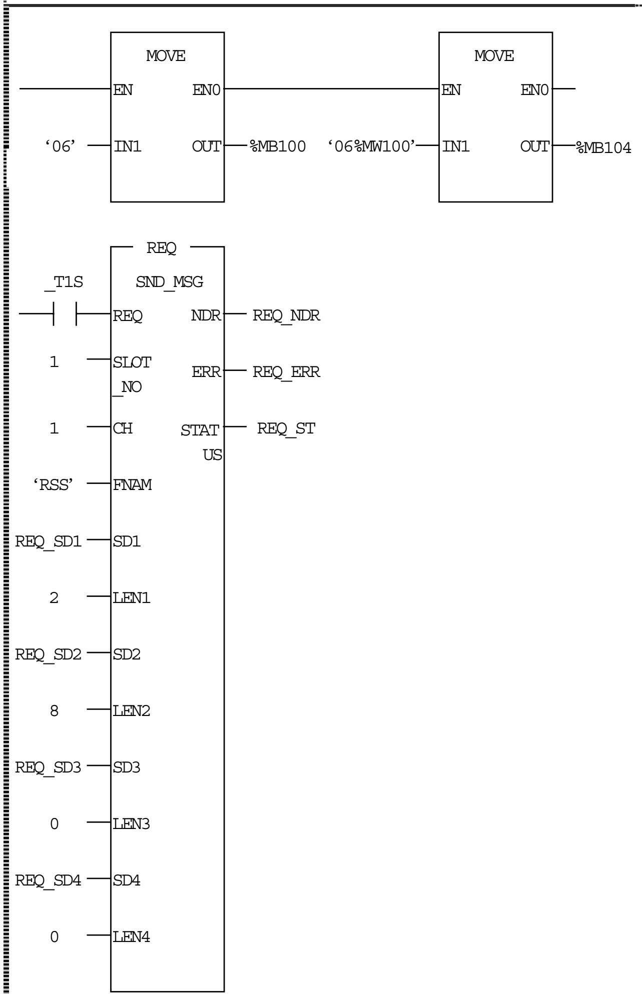
This example is prepared under below assumption.

1) Remote station: station number 06, RS-422, operation mode is GLOFA CLM
2) Frame named 'RSS' is download to CLM.

	Header	Segment1	Segment2	Segment3	Segment4 ~ Segment8	Tail
Type		ARRAY	CONSTANT	ARRAY	NONE	
Content	[ENQ]	SD1	PSS01	SD2		[EOT]
Type		2		8		
Remark	Set the header to ENQ(05H)	Set ARRAY variable to 'SD1' of 2Byte(used for another station set area in this example).	Fixed data set (Set 1 for Read command and block number in this example).	Set ARRAY variable name to 'SD2' of 8Byte(variable name size area 2Byte + Variable name area 6Byte)	Not used	Set tail to EOT (04T)

* Variable REQ_SD1 is set to %MB100 area and variable REQ_SD2 is set to %MB104 area.

LD



IL

```
LD                      '06'
MOVE
ST                      %MB100
LD                      '06%MW100'
MOVE
ST                      %MB104
CAL    SND_MSG          REQ
       REQ:=            _T1S
       SLOT_NO:=        1
       CH:=             1
       FNAM:=           'RSS'
       SD1:=            REQ_SD1
       LEN1:=           2
       SD2:=            REQ_SD2
       LEN2:=           8
       SD3:=            REQ_SD3
       LEN3:=           0
       SD4:=            REQ_SD4
       LEN4:=           0
LD                      REQ.NDR
ST                      REQ_NDR
LD                      REQ.ERR
ST                      REQ_ERR
LD                      REQ.STATUS
ST                      REQ_ST
```

(Description)

Assuming that the protocol receiving the data from another station through computer communication is as below.

Frame start + Station Number + Instruction + (Size + Data area) + Transmission end

Providing that prefix number of remote station is 06, command is RSS01, data area is sent to %MW100 and other except command is variable in the program, declare the variable as CONSTANT and ARRAY to input the data on the frame editor. When the data process compiles the send function block(SND_MSG) to frame name('RSS') and REQ_SD1 variable data to SD1 and REQ_SD2 variable data to SD2, send the data referencing 'RSS' frame name to ENQ + SD1(REQ_SD1) + RSS01 + SD2(REQ_SD2) + EOT type.
SND_MSG function block SD1~SD4 shall use Unsigned Short Integer(USINT:8Bit) and can not use the data like %MW100 for SD1~SD4 since USINT describes the data from 0 to 255. In this case, declare REQ_SD1 to 2 USINT array type(prefix '0' and '6' for send) and define the area to %MB100. As same method, declare REQ_SD2 to 8 USINT array type(06%MW100: for size 2, send data area 6) and define the area as %MB104. As SD3 and SD4 are not used, declare the area as USINT array type. Then, move string '6' to %MB100 and string '06%MW100' to %MB104, the data is sent to the required protocol sharing the data.

REQ_SD1 content

Memory address	String content
%MB100	0
%MB101	6

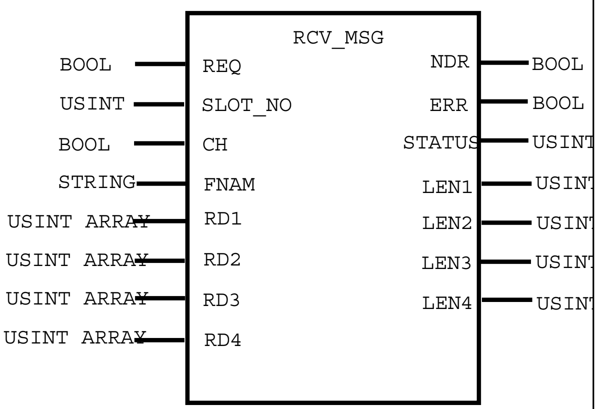
REQ_SD2 content

Memory address	String content
%MB104	0
%MB105	6
%MB106	%
%MB107	M
%MB108	W
%MB109	1
%MB110	0
%MB111	0

RCV_MSG(Receive Message)

Receive the data to remote station

Product	GM1	GM2	GM3	GM4	GM5
Applicable	◆	◆	◆	◆	◆

Function	Description
	**Input** REQ: Function block execution request at 1(rising edge) SLOT_NO : Slot number input stalling CLM CH: Channel number input to send the data 0: RS-232-C, 1: RS-422 FNAM: Frame name input to be sent RDx: Array variable name to store received data (x: 1,2,3,4) **Output** NDR: On during receiving the data without error ERR: On when the error occurs after executing function block STATUS: Detailed code value for the error LENx: Each array variable length received from remote station (x: 1,2,3,4)

■ Function

Function block that stores the data to RDx variable if receive frame downloaded to CLM is transferred from another station.

Before executing this function block, download the frame with same name of 'FNAM'. Refer to G3L-CUEA and G4L-CUEA technical document for frame download method.

■ Error

Refer to [Error] in SND_MSG function block.

■ Program example

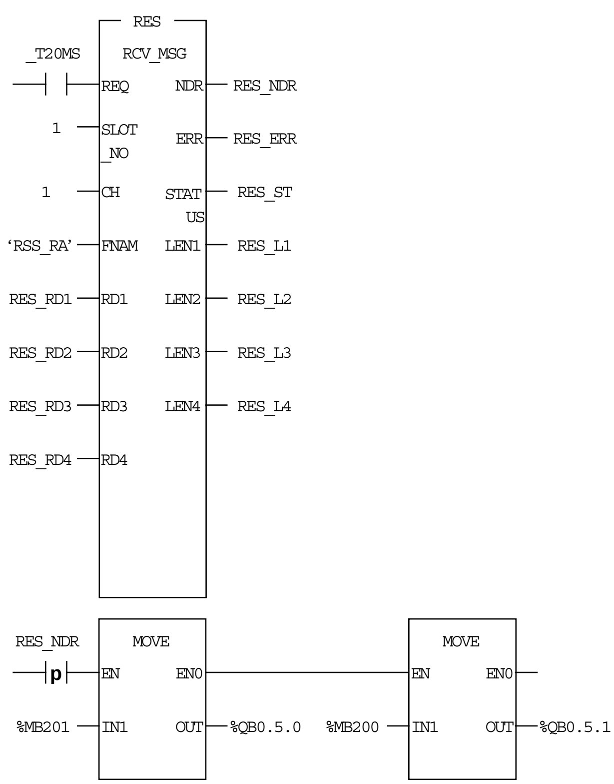
This example is prepared under below assumption.

1) Providing that the data of 06 station is received and RS-422 and operation mode is GLOFA PLC CLM.

2) Providing 'RSS_RA' frame is downloaded to CLM.

	Header	Segment1	Segment2	Segment3~Segment8	Tail
Type		CONSTANT	ARRAY	NONE	
Content	[ACK]	06RSS0102	RD1		[ETX]
Type			Hexa		
Size			2		
Remark	Set the header to ACK(06H)	Set the fixed receive data 01: Block number 02: Receive data size(2Byte)	Read 2Byte data and send it to RD1 of function block.	Not used.	Set tail to ETX (03H)

LD



IL

```
CAL     SND_MSG     RES
        REQ:=       _T20MS
        SLOT_NO:=   1
        CH:=        1
        FNAM:=      'RSS_RA'
        RD1:=       RES_RD1
        RD2:=       RES_RD2
        RD3:=       RES_RD3
        RD4:=       RES_RD4
LD                  RES.NDR
ST                  RES_NDR
LD                  RES.ERR
ST                  RES_ERR
LD                  RES.STATUS
ST                  RES_ST
LD                  RES.LEN1
ST                  RES_L1
LD                  RES.LEN2
ST                  RES_L2
LD                  RES.LEN3
ST                  RES_L3
LD                  RES.LEN4
ST                  RES_L4
LD                  RES_NDR
CAL     R_TRIG      M_EN
        CLK :=      RES_NDR
LD                  M_EN.Q
JMPN                JMP_1
LD                  %MB201
MOVE
ST                  %QB0.5.0
LD                  %MB200
MOVE
ST                  %QB0.5.1
JMP_1:
```

[Note] Providing that 'RES_RD1' memory is allocated to %MB200 in above example.

(Description)

Assuming that the protocol receiving the data from another station through computer communication is as below.

Frame start + Fixed data + Data + Transmission end

If send ACK to frame start and 06RSS0102 to fixed data and ETX to transmission end, the frame editor defines the header to [ACK], tail to [ETX], fixed data content to CONSTANT(06RSS0102), variable data to RD1. To process variable 2Byte in the function block, define RES_RD1 variable as 2 array of USINT data type and %MB200 of storage area and set this variable to RCV_MSG RD1. Output the receive data, which is input with the protocol defined at the frame editor 'RSS_RA' to %QB0.5.0 and %QB0.5.1.