e-Front runners

MICREX-SX series

SPH

USER'S MANUAL

Instructions

Preface

This User's Manual explains the system configuration, the memory and the language of SPH. Read this manual carefully to ensure correct operation.

When using modules or peripheral devices, be sure to read the corresponding user's manuals listed below.

Title	Manual No.	Contents
User's Manual Hardware, MICREX-SX series SPH	FEH201	Explains the system configuration, the specifications and operations of modules in the MICREX-SX series.
User's Manual P/PE-link modules, MICREX-SX series SPH	FEH203	Explains the communication specifications of P/PE-link, the specifications and operations of the modules.
User's Manual T-link master module / T-link interface module, MICREX-SX series SPH	FEH204	Explains the communication specifications of the T-link, the specifications and operations of the T-link master module / the T-link interface module.
User's Manual D300win <Introduction>, MICREX-SX series	FEH250	Explains the basic operations of D300win, the programming and monitoring for MICREX-SX series.
User's Manual Software PLC, MICREX-SX series SPH	FEH245	Explains the system configuration, the specifications, and operations (installation, etc.) of the software PLC for MICREX-SX series.
User's Manual D300winV2 <Reference>, MICREX-SX series	FEH254	Explains the menu and icon of D300win and all of the operations of D300winV2.

In addition to the above manuals, the following Fuji Electric FA Components & Systems Co., Ltd. site offers various manuals and technical documents associated with MICREX-SX.
URL http://www.fujielectric.co.jp/fcs/eng/

Notes

1. This manual may not be reproduced in whole or part in any form without prior written approval by the manufacturer.
2. The contents of this manual (including specifications) are subject to change without prior notice.
3. If you find any ambiguous or incorrect descriptions in this manual, please write them down (along with the manual No. shown on the cover) and contact FUJI.

Safety Precautions

Be sure to read the “Safety Precautions” thoroughly before using the module.

Here, the safety precaution items are classified into “Warning” and “Caution.”

Warning : Incorrect handling of the device may result in death or serious injury.

Caution : Incorrect handling of the device may result in minor injury or physical damage.

Even some items indicated by “Caution” may also result in a serious accident.

Both safety instruction categories provide important information. Be sure to strictly observe these instructions.

Warning

◊ Place the emergency stop circuit, interlock circuit or the like for safety outside the PC. A failure of PC might break or cause problems to the machine.

Caution

◊ Sufficiently make sure of safety before program change, forced output, starting, stopping or anything else during a run. The wrong operation might break or cause machine problems.

Revision

*Manual No. is shown on the cover.

Printed on	*Manual No.	Revision contents
Sep. 1998	FEH200	First edition
Sep. 2000	FEH200a	The tentative specifications and description of the processor bus access have been added. Standard CPU specification added. Duplex system and fail-soft specifications added. High-performance CPU NP1PS-74 specification added.
Jul. 2001	FEH200b	Software PLC specifications added. High-performance CPU NP1PS-117R specifications added.

Contents

Contents

Contents

Section 1 Specifications

Page

Section 1 Specifications
1-1 Specifications

1-1-1 High-performance CPU (SPH300)

Item			Specification		
Type			NP1PS-32	NP1PS-74	NP1PS-117R
Control system			Stored program, Cyclic scanning system (default task), periodic task, event task		
Input / Output connection method			Direct input / output (SX bus), remote input / output (T-link, OPCN-1 etc.)		
I/O control system			Via SX bus: Takt synchronous refresh Via T-link: 10ms periodic refresh		
CPU			32-bit OS processor, 32-bit execution processor		
Memory types			Program memory, Data memory, Temporary		
Programming language			IL language (Instruction List) ST language (Structured Text) LD language (Ladder Diagram) FBD language (Function Block Diagram) FSC elements (Sequential Function Chart) To IEC 61131-3		
Length of instructions			Variable length (depending on language)		
Instruction execution speed		Sequence instruction	20 to 520ns/instruction		
		Applied instruction	40ns or more/instruction		
Program memory capacity			32768 steps	75776 steps	119808 steps
Program steps in a POU			4096 steps		
No. of I/O points			512 words (Max. 8192 points)		
memory (Note 1)	I/O memory (I/Q)		512 words		
	General memory (M)		8192 words	32768 words	131072 words
	Retain memory (M)		4096 words	16384 words	32768 words
	Instance memory for User FB (M)		4096 words	16384 words	32768 words
	Instance memory for system FB (M)		16384 words	65536 words	65536 words
		Timer	512 words	2048 words	2048 words
		Integrating timer	128 words	512 words	512 words
		Counter	256 words	1024 words	1024 words
		Edge detection	1024 words	4096 words	4096 words
		Others	8192 words	32768 words	32768 words
	System memory (M)		512 words		
Temporary area			8192 words		
Available basic data type (Note 2)			BOOL, INT, DINT, UINT, UDINT, REAL, TIME, DATE, TOD, DT, STRING, WORD, DWORD		
Data type nesting			One level (array of arrays, structure of arrays, array of structures, structure of structures)		
No. of structure data type members			200		
No. of array data type elements			16-bit data type: 4096 32-bit data type: 2048		
No. of tasks			Default tasks (Cyclic scanning): 1 Periodic tasks: 4 Event tasks: 4 (Total of 4 tasks when Periodic task is used)		
Program instance (No. of POUs/resource)			256 (Max. no. of instances allowed in one task=128)		
No. of POUs in program			Earlier than D300win version V1.2: 1000 (including POUs in the library) D300win version 2.0 or later: 2000 (including POUs in the library)		

Note: 1) The area sizes of standard memory, retain memory, the instance memory for user FBs, and the instance memory for system FBs can freely be increased or decreased. Default values are shown in the above table.
2) This depends on each instruction.

Item		Specification
No. of user function blocks		512
Nesting depth of user function blocks (Note)		Earlier than D300win version V1.2: 124 levels D300win version 2.0 or later: 127 levels
No. of user functions		512
Nesting depth of user functions (Note)		Earlier than D300win version V1.2: 124 levels D300win version 2.0 or later: 127 levels
FB instance		620/POU (Up to 620 FBs allowed in one POU)
Variable	Global variable	Earlier than D300win version V1.2: 8000 D300win version 2.0 or later: 15000
	Local variable	Earlier than D300win version V1.2: 8000 (Total for all POUs) D300win version 2.0 or later: 15000/POU
No. of user FB terminals		VAR_INPUT : Up to 128 VAR_OUTPUT : Up to 128 128 in total
Library	No. of registered libraries	16/project
	Nesting depth	8 levels
Diagnostic function		Self-diagnosis (memory check, ROM sum check), System configuration supervising, Module fault monitoring
Security function		Password
Calendar		Up to 31 Dec. 2069 23:59:59 ±27sec/month (when active) When multi-CPU system is used, time is synchronized.
Battery backup		Data memory, calendar IC memory Primary lithium battery Switching time: within 5 min. (at 25° C) Durability (at 25° C): NP1PS-32 5 years NP1PS-74 1.3 years NP1PS-117R 1.3 years
Memory backup by flash ROM (contained in CPU module)		Saves application programs, system definitions, ZIP files in flash ROM of the CPU.
Memory backup by user ROM card (removable) (only for NP1PS-117R)		Application programs, system definitions, zip files and compressed projects can be saved in user ROM card (compact flash card).

Note: Total of nesting levels for user function blocks and user functions.

1-1-2 Standard CPU (SPH200)

Item			Specification	
Type			NP1PH-16	NP1PH-08
Control system			Stored program, Cyclic scanning system (default task), periodic task, event task	
Input / Output connection method			Direct input / output (SX bus), remote input / output (T-link, OPCN-1 etc.)	
I/O control system			Via SX bus: Takt synchronous refresh Via T-link: 10ms periodic refresh	
CPU			16-bit OS processor, 16-bit execution processor	
Memory types			Program memory, Data memory, Temporary	
Programming language			IL language (Instruction List) ST language (Structured Text) LD language (Ladder Diagram) FBD language (Function Block Diagram) FSC elements (Sequential Function Chart) To IEC 61131-3	
Length of instructions			Variable length (depending on language)	
Instruction execution speed		Sequence instruction	70ns or more/instruction	
		Applied instruction	140ns or more/instruction	
Program memory capacity			16384 steps	8192 steps
Program steps in a POU			4096 steps (Note 2)	
Data memory	I/O memory (I/Q)		512 words (Max. 8192 points)	
	General memory (M)		8192 words (default)	4096 words (default)
	Retain memory (M)		4096 words (default)	2048 words (default)
	Instance memory for User FB (M)		4096 words (default)	2048 words (default)
	Instance memory for System FB (M)		8192 words (default) Timer: 256 (8 words/point) Integrating timer: 64 (8 words/point) Counter: 128 (4 words/point) Edge detection: 512 (2 words/point) Others: 4096 words	4096 words (default) Timer: 128 (8 words/point) Integrating timer: 32 (8 words/point) Counter: 64 (4 words/point) Edge detection: 256 (2 words/point) Others: 2048 words
	System memory (M)		512 words	
Temporary area			4096 words	
Available basic data type (Note 1)			BOOL, INT, DINT, UINT, UDINT, REAL, TIME, DATE, TOD, DT, STRING, WORD, DWORD	
Data type nesting			One level (array of arrays, structure of arrays, array of structures, structure of structures)	
No. of structure data type members			200	
No. of array data type elements			The memory area size limits the number allowed. For example, when the standard memory size is 8192 words, up to 8192 words may be allowed.	
No. of tasks			Default tasks (Cyclic scanning): 1 Periodic tasks: 4 Event tasks: 4 (Total of 4 tasks when Periodic task is used)	
Program instance (No. of POUs/resource)			64 (Max. no. of instances allowed in one task=64)	
No. of POUs in program			Earlier than D300win version V1.2: 1000 (including POUs in the library) D300win version 2.0 or later: 2000 (including POUs in the library)	

Note:1) This depends on each instruction.
2) For CPUs earlier than Version **30 or D300win loaders earlier than Version 2.0, 2048 steps/POU

Item		Specification
No. of user function blocks		256
Nesting depth of user function blocks		64 levels
No. of user functions		256
Nesting depth of user functions		64 levels
FB instance		620/POU (Up to 620 FBs allowed in one POU)
Variable	Global variable	8000
	Local variable	8000
No. of user FB terminals		VAR_INPUT : Up to 128 VAR_OUTPUT : Up to 128 128 in total
Library	No. of registered libraries	16/project
	Nesting depth	8 levels
Diagnostic function		Self-diagnosis (memory check, ROM sum check), System configuration supervising, Module fault monitoring
Security function		Password
Calendar		Up to 31 Dec. 2069 23:59:59 ±27sec/month
Battery backup		Application programs, system definitions, ZIP files, Data memory, calendar IC memory Primary lithium battery Switching time: within 5 min. (at 25° C) Durability: 5 years (at 25° C)
User ROM (optional)		Saves application programs, system definitions, ZIP files in flash ROM of the CPU.

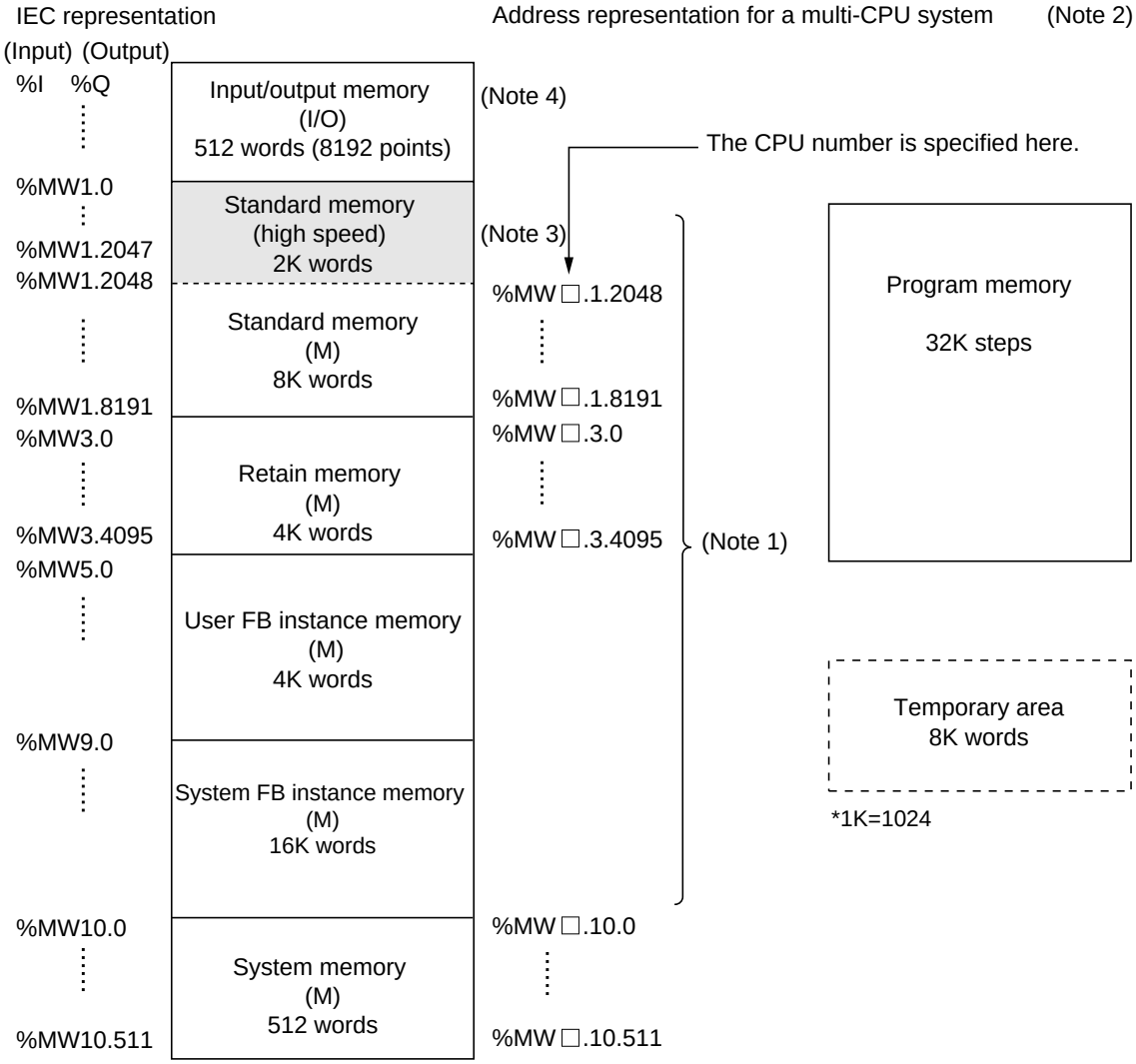
1-1-3 Software PLC (SPS)

Item			Specification
Control system			Stored program control system
Input / Output connection method			(ISA bus adapted SX bus master board is necessary.) (ISA bus adapted OPCN-1 bus master board is necessary.)
CPU			Depends on the computer you use. Pentium 233MHz or higher is recommended.
Memory types			Program memory (Stored in the main memory or hard disk of the computer) Data memory (Stored in the main memory (DRAM) of the computer) Temporary (Stored in the main memory (DRAM) of the computer)
Programming language			IL language (Instruction List) ST language (Structured Text) LD language (Ladder Diagram) FBD language (Function Block Diagram) FSC elements (Sequential Function Chart)
Instruction execution speed (Note 1)		Sequence instruction	200ns
		Applied instruction	200ns (when ADD Instruction executed)
Program memory capacity			768K steps (Stored in the main memory or hard disk of the computer)
Program steps in a POU			6K steps
Memory (Note 2)	I/O memory (I/Q)		512 words (8192 points)
	General memory (M)		256K words No high-speed memory is available. (Expandable to max. 4882K words)
	Retain memory (M)		32K words (Expandable to max. 3309K words) (Note 4)
	System memory (M)		512 words
Available basic data type (Note 3)			BOOL, INT, DINT, UINT, UDINT, REAL, TIME, STRING, WORD, DWORD
Data type nesting			One level (array of arrays, structure of arrays, array of structures, structure of structures)
No. of structure data type members			200
No. of array data type elements			16-bit data type: 4096 32-bit data type: 2048
No. of tasks			Default tasks: 1, Periodic tasks: 4
Program instance			128 (Max. No. of instances allowed in one task = 128)
No. of POUs in program			2000 (Including POUs in library)
No. of user function blocks			512 (Max. depth of nesting is 127 levels.)
No. of user functions			512 (Max. depth of nesting is 127 levels.)
FB instance			620/POU (Max. 620 FBs can be created for one POU.)
Variable	Global variable		15000
	Local variable		15000/POU
No. of user FB terminals		Input	128
		Output	128
		Input / Output	128 Total 128
Library	No. of registered libraries		16/project
	Nesting depth		8 levels
Diagnostic function			System configuration supervising, Module fault monitoring
Calendar function			None
Backup of data			Retain memory data are written in the hard disk while processing to close WindowsNT.

Note: 1) Depends on the performance of the CPU of the computer you use. Figures shown in this table are for Pentium 75 MHz processor.
2) With the software PLC, you can program without being conscious of the instance memory for FBs.
3) DATE type, DT type and TOD type are unsupported.
4) Written in the hard disk while processing to close WindowsNT.

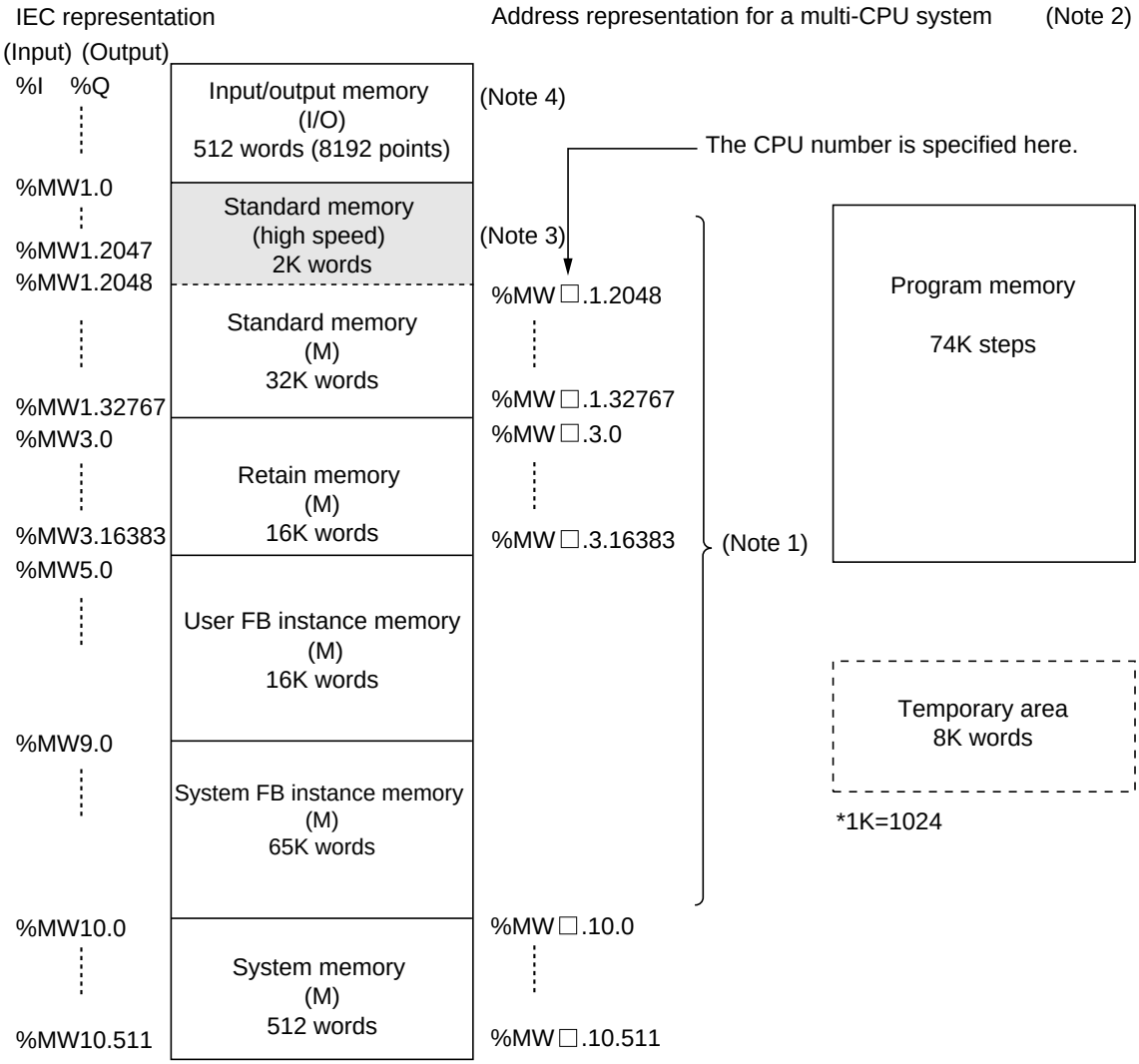
1-2-1 Memory map

(1) NP1PS-32 (High-performance CPU)



Note: 1) The size of standard memory (excluding high-speed memory), retain memory, user FB instance memory, and system FB instance memory may be increased or decreased by the loader settings. In these memories, default values have been stored. For details refer to “3-5-2 Defining the CPU memory sizes.”
2) When another CPU memory is accessed in a multi-CPU system, the CPU number is specified in the □. This is not necessary to access own memory.
3) The first 2K words of the standard memory (M) are made up of high-speed memory which is accessed at a higher speed. Another CPU can not access this area as global memory. Its size cannot be altered.
4) For the input/output address assignments, refer to “1-3 Input/output Address Assignments.”

(2) NP1PS-74 (High-performance CPU)



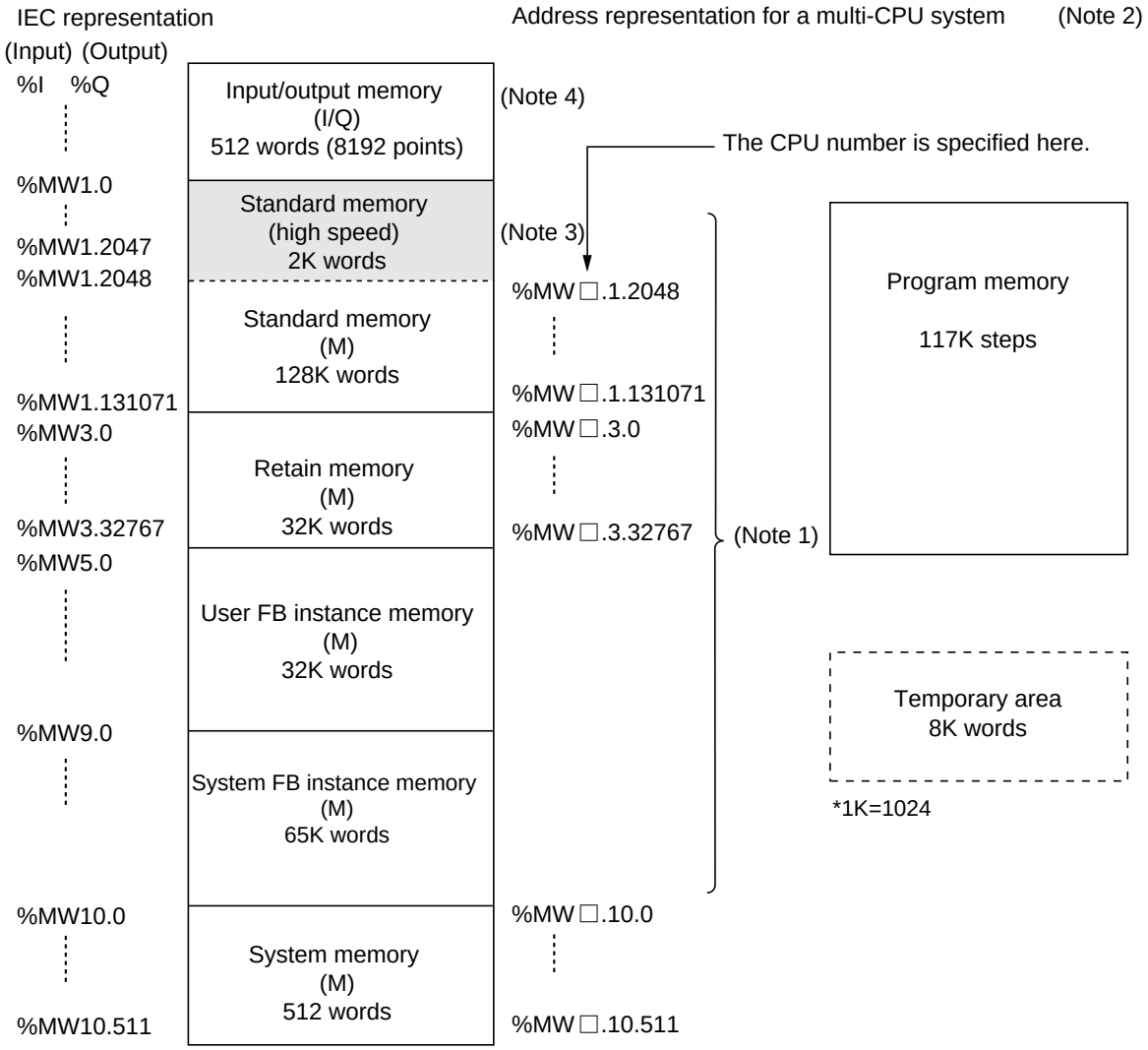
Note: 1) The size of standard memory (excluding high-speed memory), retain memory, user FB instance memory, and system FB instance memory may be increased or decreased by the loader settings. In these memories, default values have been stored. For details refer to “3-5-2 Defining the CPU memory sizes.”

2) When another CPU memory is accessed in a multi-CPU system, the CPU number is specified in the □. This is not necessary to access own memory.

3) The first 2K words of the standard memory (M) are made up of high-speed memory which is accessed at a higher speed. Another CPU can not access this area as global memory. Its size cannot be altered.

4) For the input/output address assignments, refer to “1-3 Input/output Address Assignments.”

(3) NP1PS-117R (High-performance CPU)



Note: 1) The size of standard memory (excluding high-speed memory), retain memory, user FB instance memory, and system FB instance memory may be increased or decreased by the loader settings. In these memories, default values have been stored. For details refer to “3-5-2 Defining the CPU memory sizes.”

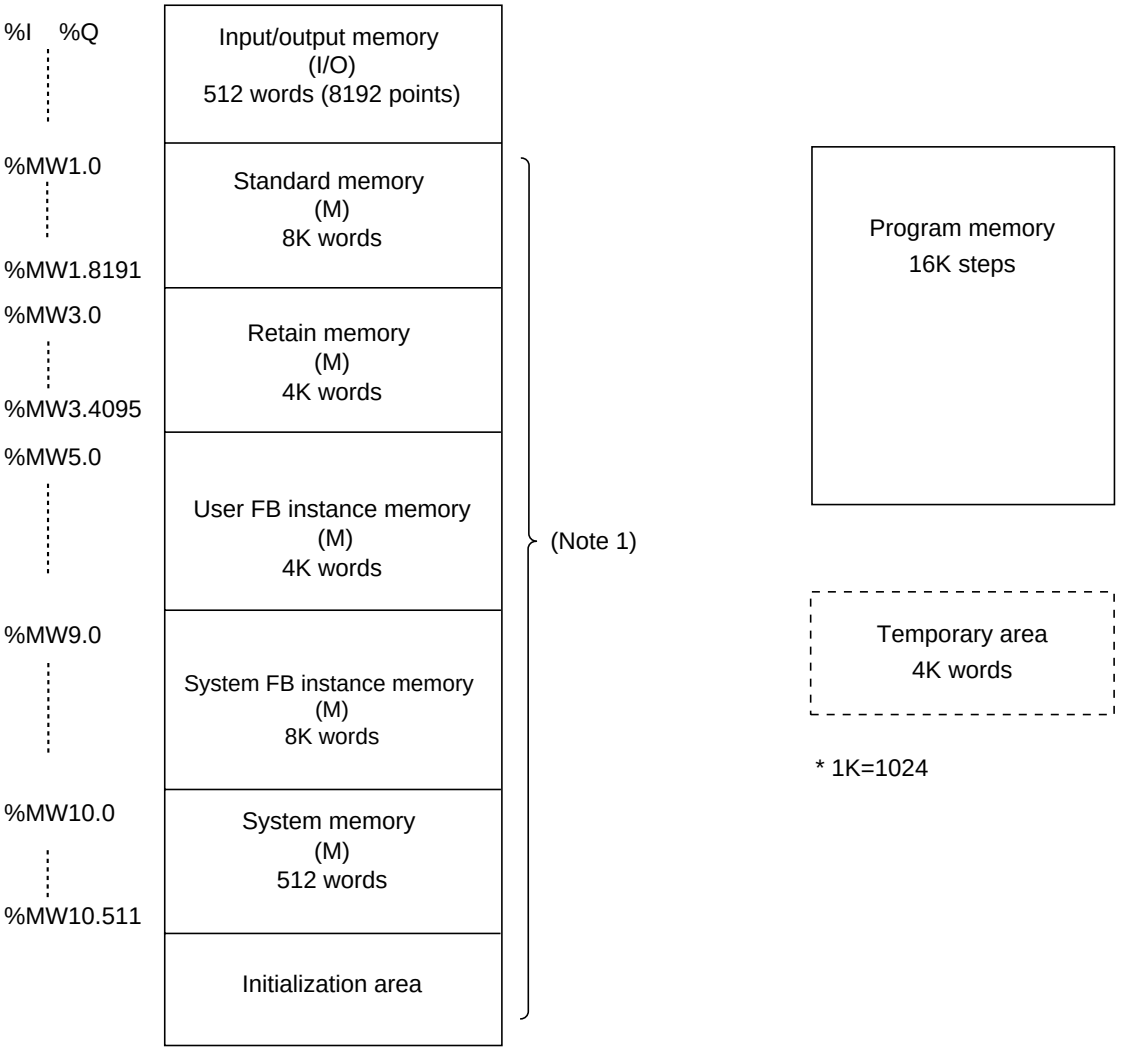
2) When another CPU memory is accessed in a multi-CPU system, the CPU number is specified in the ☐. This is not necessary to access own memory.

3) The first 2K words of the standard memory (M) are made up of high-speed memory which is accessed at a higher speed. Another CPU can not access this area as global memory. Its size cannot be altered.

4) For the input/output address assignments, refer to “1-3 Input/output Address Assignments.”

(4) NP1PH-16 (Standard CPU)

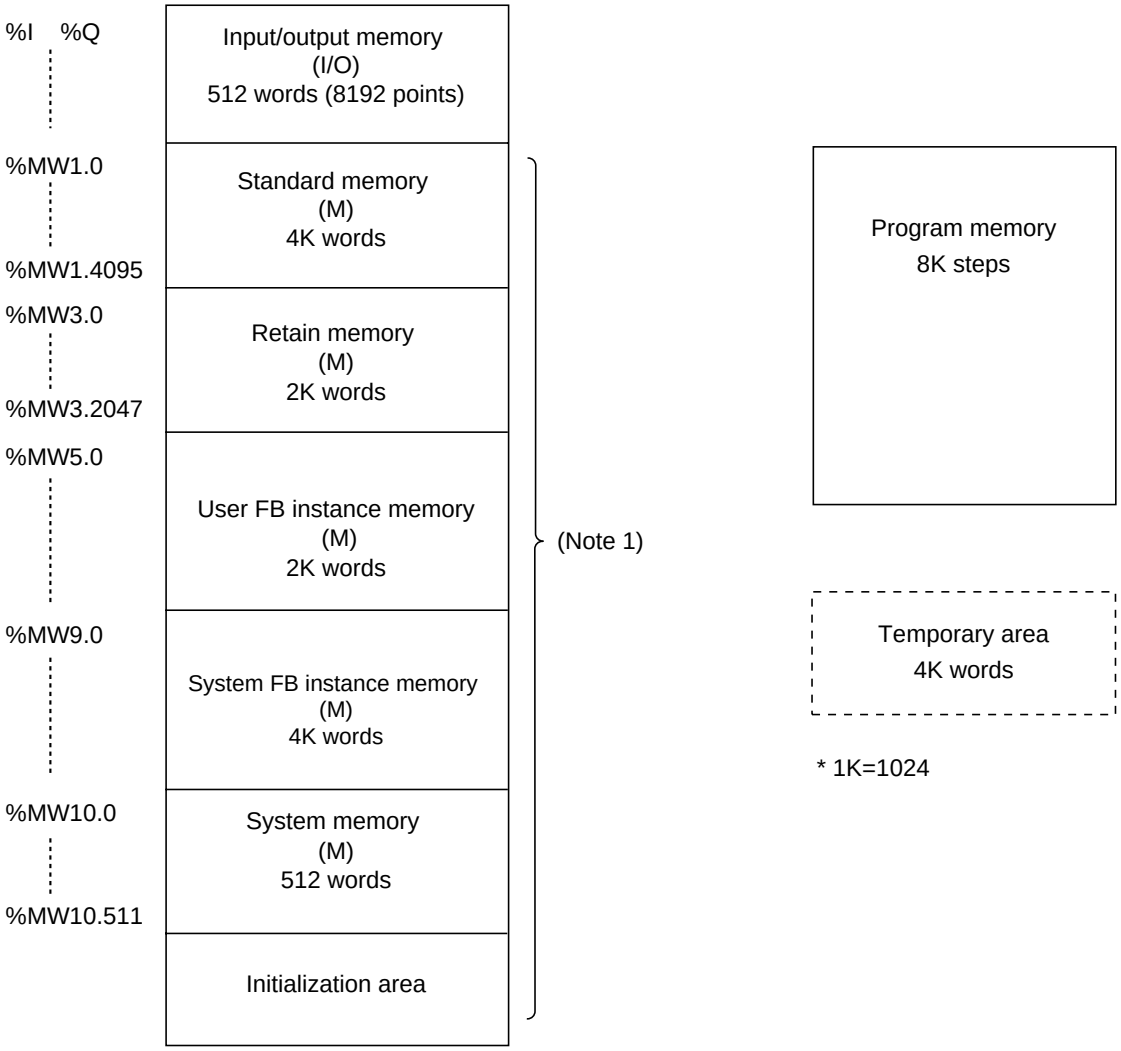
IEC representation



Note: 1) The size of standard memory, retain memory, user FB instance memory, system FB instance memory, and Initialization area may be increased or decreased by the loader settings. In these memories, default values have been stored. For details refer to “3-5-2 Defining the CPU memory sizes.”
2) No multi-system can be built up using a standard CPU module.
3) For the input/output address assignments, refer to “1-3 Input/output Address Assignments.”

(5) NP1PH-08 (Standard CPU)

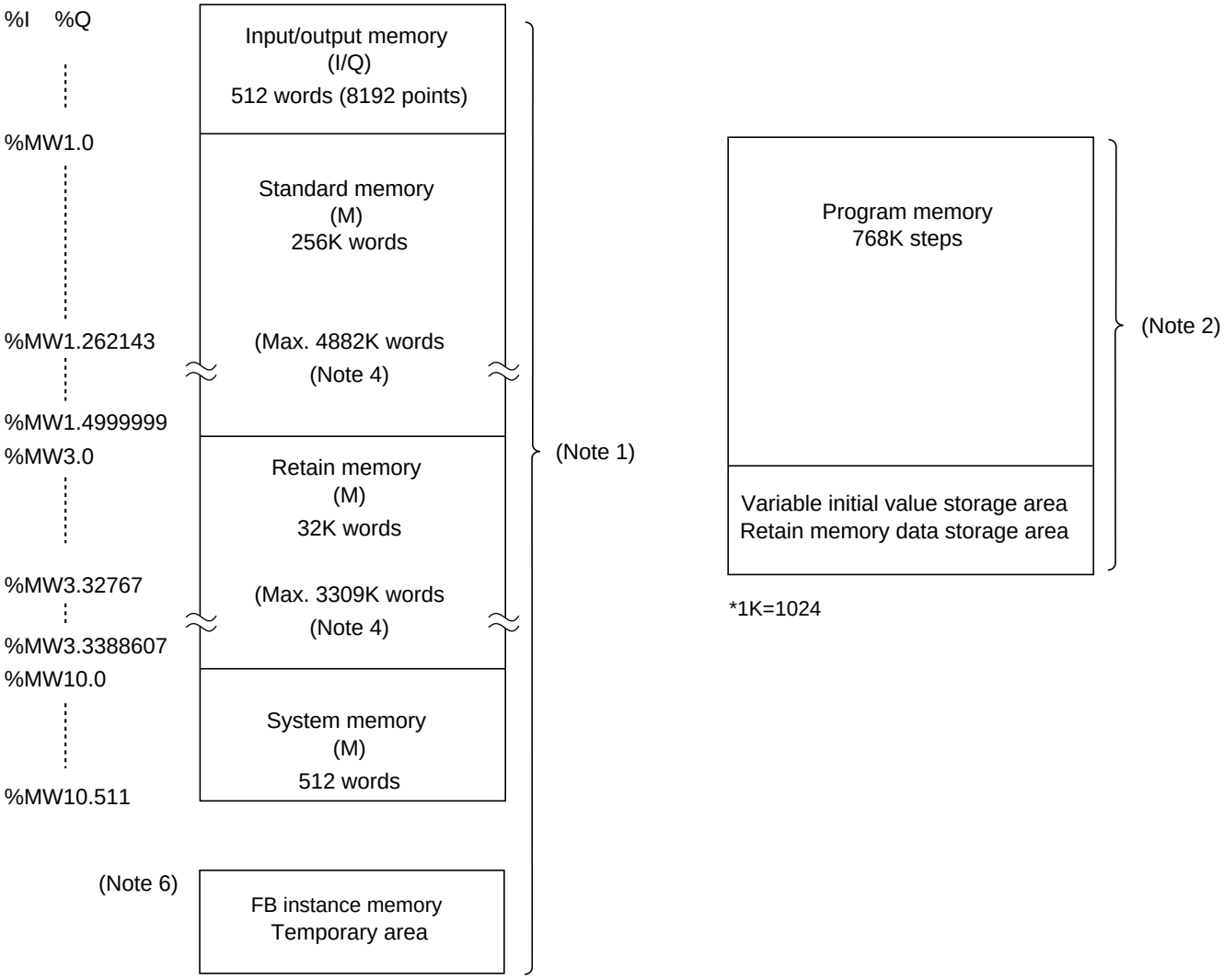
IEC representation



Note: 1) The size of standard memory, retain memory, user FB instance memory, system FB instance memory, and Initialization area may be increased or decreased by the loader settings. In these memories, default values have been stored. For details refer to “3-5-2 Defining the CPU memory sizes.”
2) No multi-system can be built up using a standard CPU module.
3) For the input/output address assignments, refer to “1-3 Input/output Address Assignments.”

(6) NP4P-SPS (Software PLC)

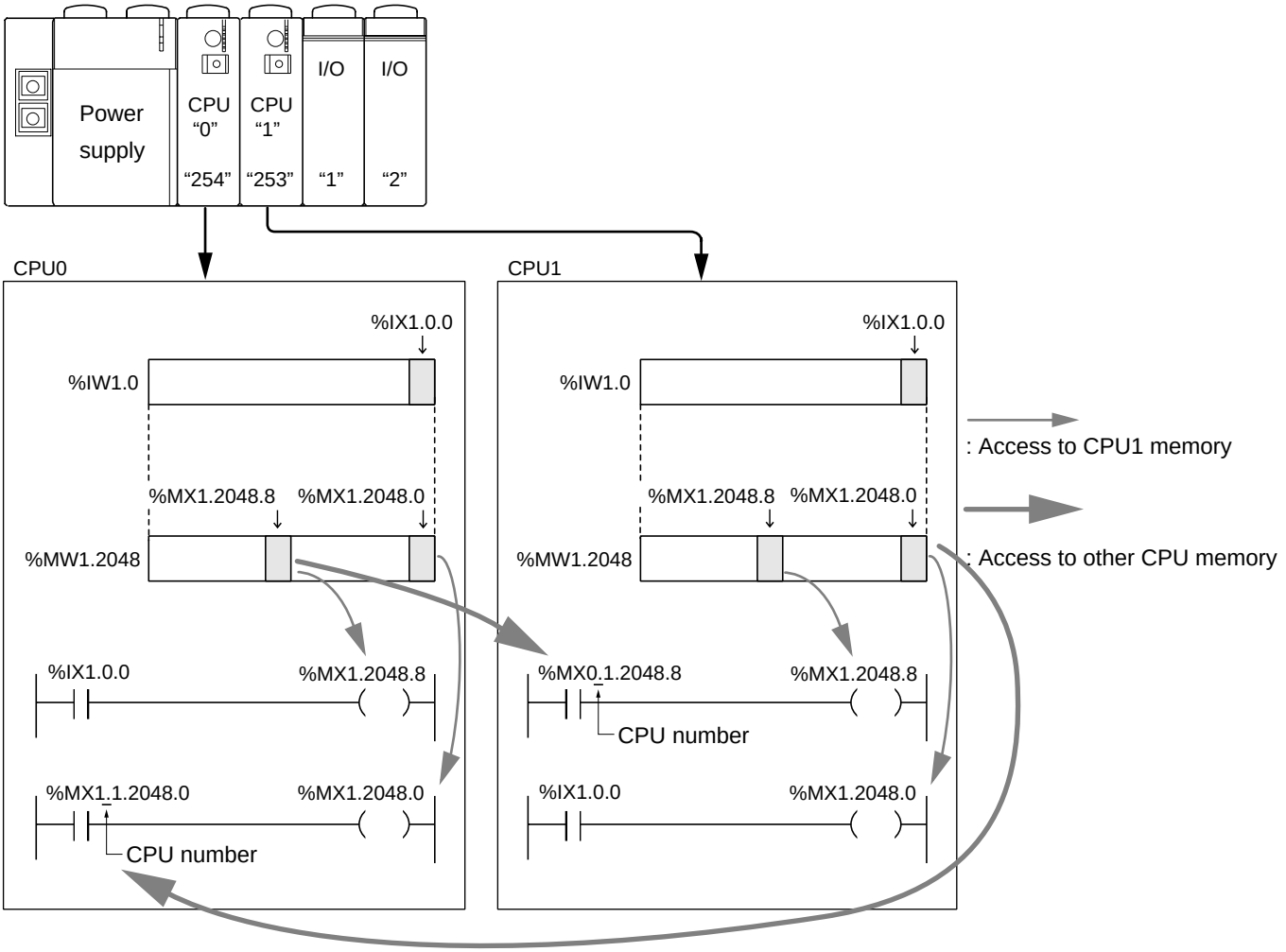
IEC representation



Note: 1) Input/output memory, standard memory, retain memory, system memory, FB instance memory, and temporary memory areas are allocated in the main memory of the computer.
2) Program memory, variable memory data storage area, and retain memory data storage area are allocated in the hard disk.
3) You can program without being conscious of the capacity of FB instance memory or temporary memory.
4) When there is enough margin in the capacity of the main memory of the personal computer, standard memory and retain memory can be expanded to maximum 4882K words and maximum 3309K words, respectively.
5) For software PLC, standard memory has no high-speed memory area.
6) For software PLC, all memories are volatile, including FB instance memory. Therefore, some FBs may function differently with software PLC from with hardware PLC (SPH300, SPH200). For more information, refer to the explanation of FB.

<Address representation for a multi-CPU system>

When CPU modules are installed on the same processor bus in a multi-CPU system, access to another CPU is represented as follows:



- When another CPU memory is accessed, the CPU number must be specified as shown in the figure above.
- Representation of the I/O area (I, Q) remains the same for both CPU0 and CPU1, as the I/O areas are common to all the CPUs.

Note: The above address representations are valid only when the CPU modules are installed on the single processor bus.

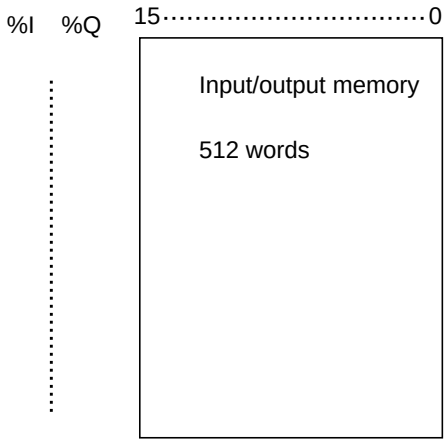
1-2-2 Input/output memory area (512 words)

The input/output memory area is a window through which data is exchanged between the CPU and external devices. It is used by input devices such as pushbuttons, switches, and sensors which send data to the CPU and by output devices such as relays, solenoids, and indicators which show the results of program executions.

Key-point

(1) Input is represented by the %I (prefix), and output by the %Q (prefix). When actually assigning an I/O address, these prefixes are followed by a size and address in the variable declaration. For details, refer to “1-3 Input/output Address Assignments” and “1-4-4 Variable declarations.”
(2) This area is also used to control I/O that is connected directly to the SX bus and remote I/O such as T-links.
(3) Both input and output cannot exist in the same word.

IEC representation

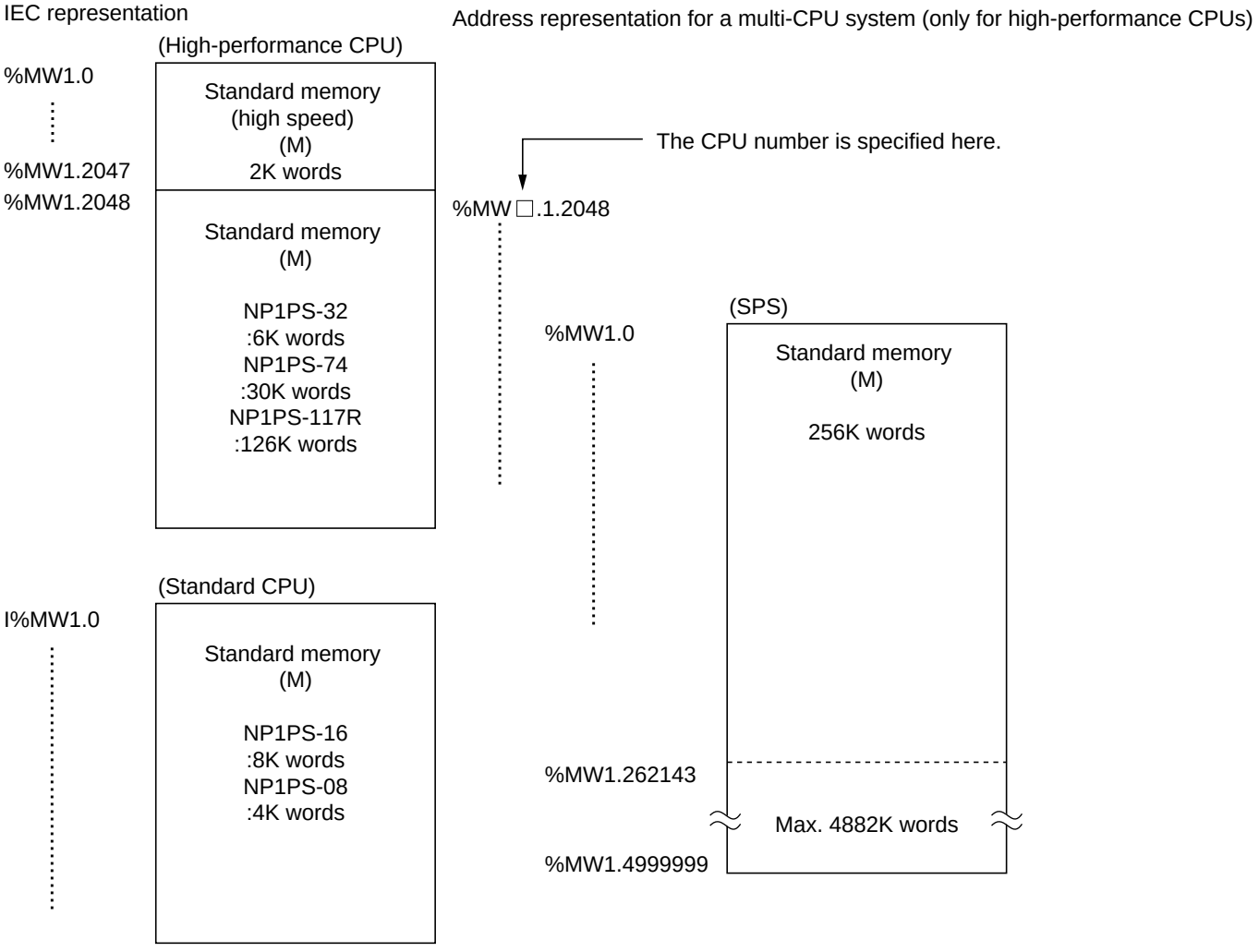


1-2-3 Standard memory area

The standard memory area is used for auxiliary relays that are used internally in the PC.

Key-point:

1) In the direct addressing mode, this memory area is specified in he format of %M □1. ..... (replace □ with any of X, W, or D). Usually, since a variable declaration is used to assign memory to an application program, you may neglect addresses in programming. Refer to "1-4 variables" for details.
2) The specified memory area is reset to (0) zero when self-PC is started.
3) For high-performance CPU, 2K words from the top of the standard memory is the area where data access in the CPU is processed at high speed. On the other hand, when data is accessed from external device such as POD, single access operation requires 81 Takts.
4) In the multi-CPU system, the other areas can be accessed as global memory areas from any other CPU (only for high-performance CPUs and SPS).
5) The sizes of the standard memory areas can be modified taking those of other areas into account. Note that the size of a fast access memory area is 2K-word fixed in the high-performance CPU and cannot be modified (only for SPH). Refer to "3-5-2 CPU Memory Size Definition" for modifying memory sizes.
6) For high-performance CPUs, no continuous access is allowed to the fast access memory area and the boundaries between other areas. For example, an array or structure cannot lie across the boundary.



1-2-4 Retain memory area

The retain memory area is used for the auxiliary relays used internally in the PC.

Key-point:

(1) In the direct addressing mode, this memory area is specified in the format of %M□1. ..... (replace□ with any of X, W, or D). Usually, since a variable declaration is used to assign memory to an application program, you may neglect addresses in programming. Refer to "1-4 variables" for details.

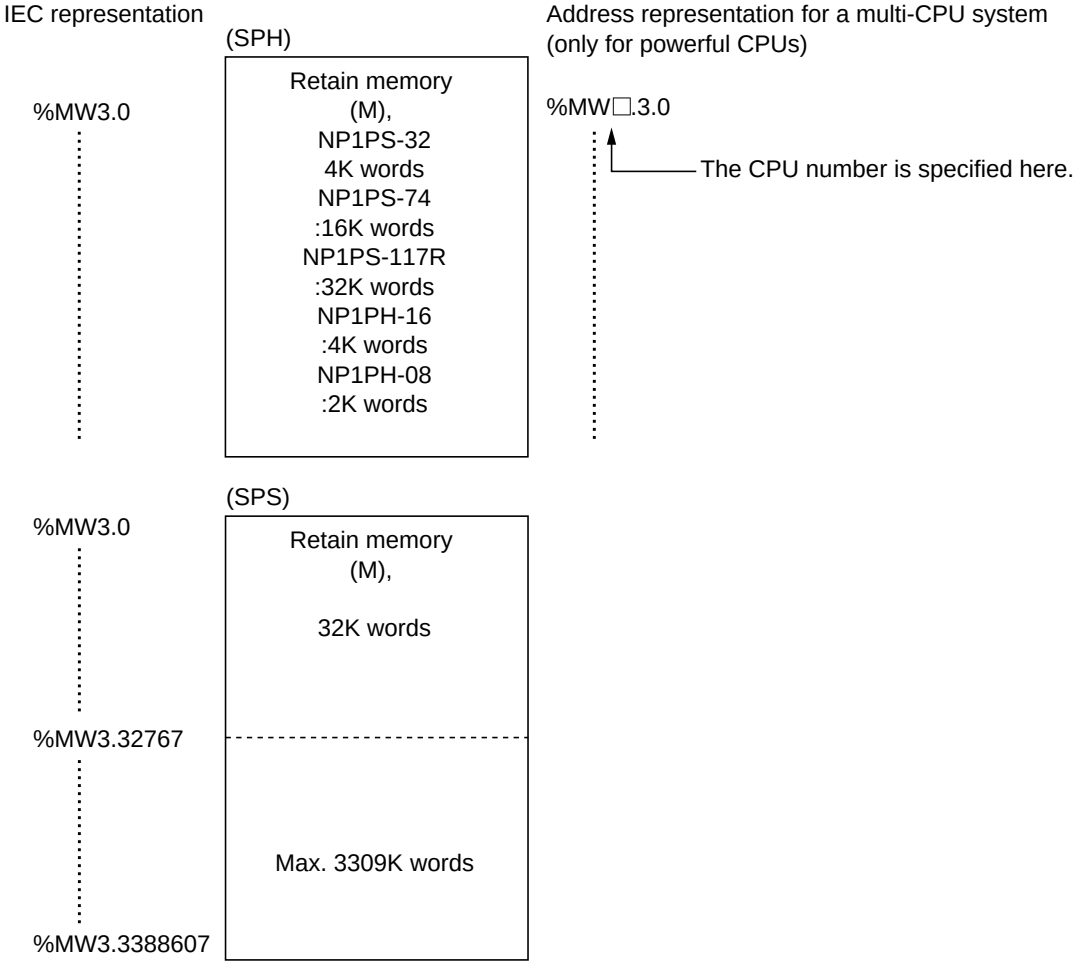
(2) The following processes are performed at cold or warm start.

	Cold	Warm
Retain memory	Reset to 0 (zero)	Retains old values
Initialized retain memory	Writes specified initial values	Retains old values

(3) When a project is downloaded, you have an option for selecting whether the area is to be cleared at project download. If you select "clear," the system cold-starts and If "not clear," the system warm-starts.

(4) In the multi-CPU system, the retain memory area can be accessed as a global memory area from any other CPU (only for high-performance CPUs).

(5) The size of the retain memory area can be modified taking those of other areas into account. Refer to "3-5-2 CPU Memory Size Definition" for modifying memory sizes (only for SPH).



Note: The system cold-starts when initially starting from the D300win while it warm-starts at power-on or when starting from the D300win.

1-2-5 User FB instance memory area

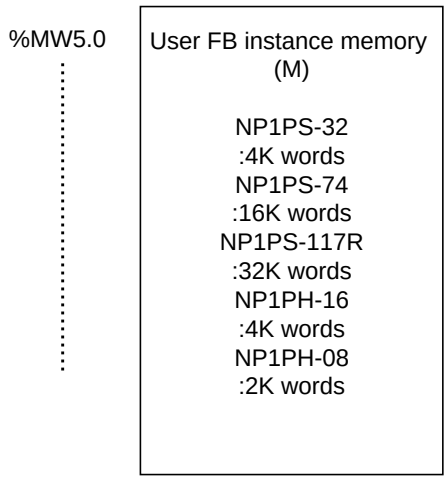
The user FB instance memory area is an instance memory area unique to each user FB used internally in the PC.

Key-point:

(1) Do not read out from and write data into the instance memory area from an application program or the D300win.
If you neglect this advice, the user FB may not operate correctly.

(2) The size of the user FB instance memory area can be modified taking those sizes of other areas into account.
Refer to "3-5-2 CPU Memory Size Definition" for modifying memory sizes.

IEC representation



Note: With SPS, you can program without being conscious of this area. However, the declaration of FB instance in the local variable worksheet is necessary.

1-2-6 System FB instance memory area

The system FB instance memory area is an instance memory area that is unique to each of the system FBs, such as timers, counters, and differential instructions, which are used internally in the PC.

Key-point:

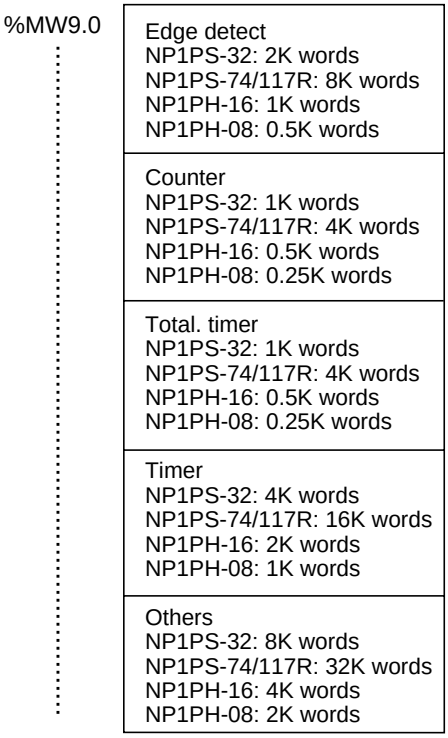
(1) Do not read out from and write data into the instance memory area from an application program or the D300win. If you neglect this advice, the user FB may not operate correctly.
(2) At PC startup, predefined initialization is performed. (the old values are retained or reset to 0s (zeros). Note that the values, even in the area where old values are usually retained, are reset to 0s (zeros) whenever a project is downloaded.
Example) The current values for the counters and totalizing timers and the old value for the edge detect instruction counter are retained, while the current values for the timers (excluding totalizing) are reset to 0s (zeros).
(3) Eight words/timer, four words/counter, and two words/edge detect instruction counter are used.
(4) The size of the system FB instance memory area can be modified taking those of other areas into account. Refer to "3-5-2 CPU Memory Size Definition" for modifying memory sizes.
(5) By default, the numbers of timers, totalizing timers, counters, and edge detect instruction counters have been preset per CPU. If necessary, these numbers may be modified.

	Timer	Total. timer	Counter	Edge detect	Others
NP1PS-32	512	128	256	1024	8192 words
NP1PS-74/117R	2048	512	1024	4096	32768 words
NP1PH-16	256	64	128	512	4096 words
NP1PH-08	128	32	64	256	2048 words

To set the timer, totalizing timer, counter, edge detect, and other system FB areas, the following conditions should be met.

(No. of timers) x words + (No. of counters) 4 words + (No. of edge detect counters) x words + others
≤ Preset size of system FB instance memory area

IEC representation



Note: 1) The numbers of words are default values in the memory map on the left side.
2) With SPS, you can program without being conscious of this area. However, the declaration of FB instance in the local variable worksheet is necessary.

1-2-7 Initialization area

The initialization area assigned only in the standard CPU variables.
stores the initial values for the user function block (FB) and

Key-point:

(1) The sizes of the storing areas are calculated by the following expression.
(Initialization area) = (No. of words in user FB instance area) x 9/8
+ (No. of variables for which initial values are set) x 5 (words)

(2) The default values have been preset as shown below.

	Initialization area (entire)	Initial user FB value storing area	Initial variable value storing area
NP1PH-16	7K words	4608 words	2560 words
NP1PH-08	3K words	2304 words	768 words

From the table shown above, the numbers (default value) of variables for which initial values can be set are calculated by the expressions below.
NP1PH-16: 2560 / 5 =512 (Any digits under the decimal point are truncated) 512
NP1PH-08: 768 / 5 =153 (Any digits under the decimal point are truncated) 153

(3) The size of a user FB initialization area requires: No. of words of the preset user FB instance area x 9/8

Initialization area

Initial user FB value storing area

NP1PH-16:
4608 words
(default)
NP1PH-08:
2304 words
(default)

Initial variable value storing area

NP1PH-16:
2560 words
(default)
NP1PH-08:
768 words
(default)

1-2-8 System memory area (512 words)

The system memory area is allocated to flags which indicate the operating status or the error status of the MICREX-SX series. This area is for exclusive use.

A resource is defined as one CPU system that configured by one CPU module and two or more I/O modules.

(1) System memory

%MW10.0	Resource operating status
%MW10.1	Resource switch information/User ROM state
%MW10.2	Resource fatal fault factor
%MW10.3	Not used
%MW10.4	Resource nonfatal fault factor
%MW10.5	Not used
%MW10.6	CPU error factor
%MW10.7	Not used
%MW10.8, 9	Memory error factor
%MW10.10 %MW10.11	SX bus error factor
%MW10.12	Application error factor (fatal fault)
%MW10.13	Application error factor (nonfatal fault)
%MW10.14 | %MW10.16	User fatal fault factor 0 - factor 47
%MW10.17	Not used
%MW10.18 | %MW10.20	User nonfatal fault factor 0 - factor 47
%MW10.21	Not used
%MW10.22 | %MW10.29	System definition error factor
%MW10.30 | %MW10.37	Not used
%MW10.38 %MW10.39	Application program error factor
%MW10.40 %MW10.41	Not used
%MW10.42 %MW10.43	Annunciator relay
%MW10.44 %MW10.45	Not used
%MW10.46	Duplex master annunciator
%MW10.47	Duplex operation mode
%MW10.48, 49	Resource operation/running information
%MW10.50, 51	Resource configuration/fault information
%MW10.52 | %MW10.67	SX bus configuration information (configuration information)
%MW10.68 | %MW10.83	SX bus fault information (configuration information)
%MW10.84 | %MW10.99	SX bus-connected module fail-soft mode information
%MW10.100 | %MW10.127	Not used
%MW10.128 | %MW10.135	Remote I/O master 0 I/O module configuration information
%MW10.136 | %MW10.143	Remote I/O master 0 I/O module fault information
%MW10.144 | %MW10.151	Remote I/O master 1 I/O module configuration information
%MW10.152 | %MW10.159	Remote I/O master 1 I/O module fault information
%MW10.160 | %MW10.167	Remote I/O master 2 I/O module configuration information
%MW10.168 | %MW10.175	Remote I/O master 2 I/O module fault information
%MW10.176 | %MW10.183	Remote I/O master 3 I/O module configuration information
%MW10.184 | %MW10.191	Remote I/O master 3 I/O module fault information
%MW10.192 | %MW10.199	Remote I/O master 4 I/O module configuration information
%MW10.200 | %MW10.207	Remote I/O master 4 I/O module fault information
%MW10.208 | %MW10.215	Remote I/O master 5 I/O module configuration information
%MW10.216 | %MW10.223	Remote I/O master 5 I/O module fault information
%MW10.224 | %MW10.231	Remote I/O master 6 I/O module configuration information
%MW10.232 | %MW10.239	Remote I/O master 6 I/O module fault information
%MW10.240 | %MW10.247	Remote I/O master 7 I/O module configuration information
%MW10.248 | %MW10.255	Remote I/O master 7 I/O module fault information
%MW10.256 | %MW10.299	Not used

(Continued on next page)

%MW10.300 | %MW10.315	Configuration information of all module in the resource (only for SPS)
%MW10.316 | %MW10.331	Fault information of all module in the resource (only for SPS)
%MW10.332 | %MW10.359	Not used
%MW10.360 | %MW10.367	Remote I/O master board 0 I/O module configuration information (only for SPS)
%MW10.368 | %MW10.375	Remote I/O master board 0 I/O module fault information (only for SPS)
%MW10.376 | %MW10.383	Remote I/O master board 1 I/O module configuration information (only for SPS)
%MW10.384 | %MW10.391	Remote I/O master board 1 I/O module fault information (only for SPS)
%MW10.392 | %MW10.399	Remote I/O master board 2 I/O module configuration information (only for SPS)
%MW10.400 | %MW10.407	Remote I/O master board 2 I/O module fault information (only for SPS)
%MW10.408 | %MW10.415	Remote I/O master board 3 I/O module configuration information (only for SPS)
%MW10.416 | %MW10.423	Remote I/O master board 3 I/O module fault information (only for SPS)
%MW10.424 | %MW10.431	Remote I/O master board 4 I/O module configuration information (only for SPS)
%MW10.432 | %MW10.439	Remote I/O master board 4 I/O module fault information (only for SPS)
%MW10.440 | %MW10.507	Not used

%MW10.508 | %MW10.511	SX bus transmission error rate information

(2) Resource operating status %MW10.0 (Read only)
The table given below shows the resource (CPU module) operating status and operating modes.

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.0.0	Run	Set to "ON" while the CPU is running.	O	O	O
%MX10.0.1	Stop	Set to "ON" while the CPU is down.	O	O	O
%MX10.0.2	Fatal fault	Set to "ON" when a fatal resource error has occurred.	O	O	O
%MX10.0.3	Nonfatal fault	Set to "ON" when a non-fatal resource error has occurred.	O	O	O
%MX10.0.4	Duplex operating station	Set to "ON" when an operating CPU is running in the duplex mode.	O	-	-
%MX10.0.5	Duplex waiting station	Set to "ON" when a waiting CPU is running in the duplex mode.	O	-	-
%MX10.0.6	1:1 duplex	Set to "ON" when the system is in the 1 to 1 duplex mode.	O	-	-
%MX10.0.7	N:1 duplex	Set to "ON" when the system is in the N to 1 duplex mode.	O	-	-
%MX10.0.8	Non-automatic operation mode	Set to "ON" while in the non-automatic operation mode.	O	O	O
%MX10.0.9	Automatic operation mode	Set to "ON" while in the automatic operation mode.	O	O	O
%MX10.0.10	Preceding state mode	Set to "ON" while in the preceding state mode.	O	O	-
%MX10.0.11	Battery-less operation mode	Set to "ON" while in the battery-less mode.	O	O	-
%MX10.0.12	Not used		-	-	-
%MX10.0.13 Note)	SX bus-connected module fail-soft mode	Set to "ON" when fail-soft may be performed for all the modulesconnected to the SX bus or individually reset	O	O	O
%MX10.0.14	Processor bus master	Set to "ON" when the CPU module is controlling the processor bus.	O	O	-
%MX10.0.15	SX bus master	Set to "ON" when the CPU module is controlling the SX bus.	O	O	O

O: Supported, -: Not supported

Non-automatic operation mode
The mode in which the CPU will not start operation when the system power is turned on with the key switch on the CPU module front panel set to “RUN” or “TERM.” This is set by the system definition for the CPU module.
With SPS, this mode is activated when there is no boot project.

Automatic operation mode
The mode in which the CPU will start operation when the system power is turned on with the key switch on the CPU module front panel set to “RUN” or “TERM.” This is set by the system definition for the CPU module. The mode is enabled in the system resource configuration at power-on. (The automatic mode is on by default.)
With SPS, this mode is activated when there is a boot project.

Preceding state mode
The mode in which the CPU will start operation when system power is turned on with the key switch on the CPU module front panel set to “RUN”; when system power is turned on with the key switch set to “TERM,” the CPU will enter the preceding state (running or stopped) that was established when power was turned off in the preceding run.

Battery-less operation mode
At system power-on, the entire memory is initialized to initial values or all zeros. Note that neither battery connection check norvoltage check is done. The mode is enabled in the system configuration at power-on. When the preceding state mode is turned on in the battery-less mode, the CPU enters the automatic operation mode. For the standard CPU, this mode is not enabled unless a user ROM card is inserted.

Note: <When %MX10.0.13=0 (Fail-soft has been disabled for the modules connected to the SX bus)>
For common modules, fail-soft is not performed, and for the I/O modules connected to the SX bus, it is not performed even if enabled by using the loader (D300win).
<When %MX10.0.13=1 (Fail-soft has been enabled for the modules connected to the SX bus)>
For common modules, fail-soft is performed, and for the I/O modules connected to the SX bus, it is performed because it has been enabled by using the loader (D300win).

(3) Resource switch / User ROM state %MW10.1 (Read only)
This area indicates the state of the CPU module switches that control the resource.

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.1.0 | %MX10.1.3	CPU number	Indicates the 4-bit number (0-F) set using the CPU number setting switches on the CPU module front panel. A range of numbers 0-7 is allowed.	O	-	O
%MX10.1.4 %MX10.1.5	Not used		-	-	-
%MX10.1.6	User ROM card connection state	1: connected 0: unconnected	O Note 1	O	-
%MX10.1.7	User ROM card write protect	1:write-protected 0:write-permitted (enabled when %MX10.16=1)	O Note 1	O	-
%MX10.1.8	STOP position	Set to "ON" when the key switch is in the STOP position.	O	O	-
%MX10.1.9	TERM position (bottom)	Set to "ON" when the key switch is in the TERM position (bottom).	O	O	-
%MX10.1.10	TERM position (top) (Note 2, 3)	Set to "ON" when the key switch is in the TERM position (top).	O	O	-
%MX10.1.11	RUN position	Set to "ON" when the key switch is in the RUN position.	O	O	-
%MX10.1.12 | %MX10.1.15	Not used		-	-	-

Note: 1) User ROM card (compact flash card) adapted models only.
2) The TERM position flag also turns on when the key switch is in an unknown state.
3) With user ROM card adapted high-performance CPU module, this is set to “ON” when the key switch is set to UR8M_TERM position.

(4) Resource fatal fault factor %MW10.2 (Read only)
This area indicates the factors of fatal faults that will stop the resource (one-CPU system).

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.2.0	CPU error	Set to "ON" when a fatal fault has occurred in the CPU module.	O	O	-
%MX10.2.1	Power supply fault	Set to "ON" when a power-off condition has occurred.	O	O	-
%MX10.2.2	Memory error	Set to "ON" when an error has occurred in the memory in the CPU module.	O	O	-
%MX10.2.3	SX bus error	Set to "ON" when SX bus error occurs, for example, the disconnection of cable or loop-back plug.	O	O	O
%MX10.2.4	Application error	Set to "ON" when an error has been found in an application program or system definition.	O	O	O
%MX10.2.5	I/O module error	Set to "ON" when a fault has occurred in any of the I/O modules controlled by the self-CPU module and fail-soft has been "disabled." When fail-soft has been "enabled," the entire system continues operating normally even if a fault has occurred in an I/O module.	O	O	O
%MX10.2.6	Common module error	Set to "ON" when a fault has occurred in any of the common modules (excluding self-module) connected to the SX bus.	O	O	O
%MX10.2.7	Relay-switching error	Set to "ON" when relay-switching cannot be performed in the duplex operation mode.	O	-	-
%MX10.2.8 %MX10.2.9	Not used		-	-	-
%MX10.2.10	Remote I/O module error on remote I/O master board	Set to "ON" when the system is down due to an error occurred on the remote I/O unit or module.	-	-	O
%MX10.2.11	Driver error		-	-	O
%MX10.2.12	Not used		-	-	-
%MX10.2.13	Other hardware error	Set to "ON" when an error has occurred in a CPU number selection switch.	O	O	-
%MX10.2.14	Not used		-	-	-
%MX10.2.15	User fatal fault	Set to "ON" when the user program turns on one of the user fatal fault flags (%MX10.14.0 to %MX10.16.15).	O	O	O

(5) Resource nonfatal fault factor %MW10.4 (Read only)
This area indicates the factors of faults that allow the resource to continue processing.

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.4.0 | %MX10.4.2	Not used		-	-	-
%MX10.4.2	Memory error	Set to "ON" when an error has occurred in the memory of the self-CPU module.	O	O	-
%MX10.4.3	SX bus error	Set to "ON" when an error has occurred in the SX bus.	O	O	O
%MX10.4.4	Application error	Set to "ON" when an error has been found in an application program or system definition.	O	O	O
%MX10.4.5	I/O module error	Set to "ON" when a fault has occurred in any of the I/O modules controlled by the self-CPU module and fail-soft has been enabled. Note)	O	O	O
%MX10.4.6	Common module error (Note *)	Set to "ON" when a fault has occurred in any of the I/O modules (excluding self-CPU) connected to the SX bus.	O	O	O
%MX10.4.7 | %MX10.4.9	Not used		-	-	-
%MX10.4.10	Remote I/O module error	Set to "ON" when an error has occurred on the remote I/O unit or module. With SPS, the system is stopped because fail-soft mode is not supported.	-	-	O
%MX10.4.11	Not used		-	-	-
%MX10.4.12	User ROM card - CPU mismatch	Set to "ON" when the content of user ROM card does not coincide with that of the CPU. Verification targets are system definition, project and password.	O Note 1	-	-
%MX10.4.13	Other hardware error	Set to "ON" when a fault in any of the key switches or loader/general-purpose communication selection switch occurs. The CPU module operates by assuming that "TERM" has been enabled if a fault has been detected in the key switch. It assumes that the loader side has been selected if a fault is detected in the loader/general-purpose communication selection switch.	O	O	-
%MX10.4.14	Battery error	Set to "ON" when the voltage of the data backup battery falls below the threshold level or the battery is dead.	O	O	-
%MX10.4.15	User nonfatal fault	Set to "ON" when the user program turns on one of the user non-fatal fault flags (%MX10.18.~%MX10.20.15).	O	O	O

Note: 1) User ROM card (compact flash card) adapted models only
* The common modules are those connected to the SX bus without occupying an I/O area (for example, a CPU module, communication module, etc.).

(6) CPU error factor %MW10.6 (Read only) (Not supported by SPS)

Address	Name	Description
%MX10.6.0	Arithmetic processor error	Hardware error in the arithmetic LSI in the CPU module
%MX10.6.1	OS processor error	Hardware error in the OS control LSI in the CPU module
%MX10.6.2 | %MX10.6.15	Not used	

(7) Memory error factor %MW10.8, %MW10.9 (Read only) (Not supported by SPS)

Address	Name	Description	Level
%MX10.8.0	System ROM error	Set to "ON" when an error has occurred in the system ROM in the CPU module.	Fatal fault
%MX10.8.1	System RAM error	Set to "ON" when an error has occurred in the system RAM in the CPU module.	Fatal fault
%MX10.8.2	Application ROM error	Set to "ON" when an error has occurred in the user program ROM in the CPU module.	Fatal fault (Note 1)
%MX10.8.3	Application RAM error	Set to "ON" when an error has occurred in the user program RAM in the CPU module.	Fatal fault
%MX10.8.4 | %MX10.8.14	Not used		
%MX10.8.15	Memory backup error	Set to "ON" when no power-failure-time data is retained.	Fatal fault (Note 2)
%MX10.9.0 | %MX10.9.14	Not used		
%MX10.9.15	Memory backup error	Set to "ON" when no power-failure-time data is retained. May clear the error condition by using an application program.	Nonfatal fault (Note 2)

Note: 1) Set to “ON” when an error has occurred in the user ROM card.
2) For a high-performance CPU, the bit set to “ON” when a memory backup error has occurred depends on the module version.
Earlier than V**.25: %MX10.8.15 V10.30 or later: %MX10.9.15

System operation after a memory error has occurred
Any memory backup error resets the entire user memory area to 0 (zero). Note that in most cases, since %MX10.8.~3 are set to “ON” when a hardware fault has occurred, cycling the power source may cause a memory error to be repeated.

(8) SX bus error factor %MW10.10, %MW10.11

Address	Name	Description	Level	SPH 300	SPH 200	SPS
%MX10.10.0	SX bus LSI error	Set to "ON" when an error has occurred in the LSI controlling the SX bus.	Fatal fault	O	O	O
%MX10.10.1	Station number double-assignment	Set to "ON" when the same SX bus station number has been assigned to more than one module in the configuration.	Fatal fault	O	O	O
%MX10.10.2	Module count exceeded	Set to "ON" when the number of modules connected to the SX bus exceeds 254.	Fatal fault	O	O	O
%MX10.10.3 | %MX10.10.12	Not used			-	-	-
%MX10.10.13	SX bus transmission error	Set to "ON" when an error has occurred in transmission via the SX bus.	Fatal fault	O	O	O
%MX10.10.14	Processor bus access error	Set to "ON" when an error has occurred in accessing the processor bus.	Fatal fault	O	O	-
%MX10.10.15	I/O refresh slow-down	Set to "ON" when the input/output data has not been updated on the SX bus for longer than 128ms.	Fatal fault	O	O	O
%MX10.11.0 | %MX10.11.13	Not used			-	-	-
%MX10.11.14	Processor bus access error	Set to "ON" when an access error has occurred on the processor bus (the cause of the error is attributable to the accessed module). May clear the error condition by using an application program	Fatal fault	O	O	-
%MX10.11.15	Not used			-	-	-

(9) Application error factor %MW10.12, %MW10.13 (Read only)

Address	Name	Description	Level	SPH 300	SPH 200	SPS
%MX10.12.0	System definition error	Set to "ON" when an error has been found in the system definition.	Fatal fault	O	O	O
%MX10.12.1	Application program error	Set to "ON" when an error has been found in the Application program.	Fatal fault	O	O	O
%MX10.12.2 | %MX10.12.15	Not used			-	-	-
%MX10.13.0	Not used			-	-	-
%MX10.13.1	Application program error	Set to "ON" when an error has been found in the Application program.	Nonfatal fault	O	O	-
%MX10.13.2 | %MX10.13.15	Not used			-	-	-

(10) User fatal fault %MW10.14 to %MW10.16

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.14.0	User fatal fault factor 0	A fatal error has occurred and the CPU stops when either one of these bits is set to "ON" by an application program.	O		
|	|				
%MX10.14.15	User fatal fault factor 15				
%MX10.15.0	User fatal fault factor 16				
|	|				
%MX10.15.15	User fatal fault factor 31				
%MX10.16.0	User fatal fault factor 32				
|	|				
%MX10.16.15	User fatal fault factor 47				

(11) User nonfatal fault %MW10.18 to %MW10.20

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.18.0	User nonfatal fault factor 0	A non-fatal error has occurred and the CPU continues running when either one of these bits is set to "ON" by an application program. Changing the bit set to "ON" to "OFF" by an application program lets the system recover from the non-fatal error state.	O		
|	|				
%MX10.18.15	User nonfatal fault factor 15				
%MX10.19.0	User nonfatal fault factor 16				
|	|				
%MX10.19.15	User nonfatal fault factor 31				
%MX10.20.0	User nonfatal fault factor 32				
|	|				
%MX10.20.15	User nonfatal fault factor 47				

(12) System definition error factor %MW10.22 to %MW10.29 (Read only)

Address	Name	Description	Level	SPH 300	SPH 200	SPS
%MX10.22.0	Not used			-	-	-
%MX10.22.1	System definition error	Set to "ON" when the contents of the system configuration definition in the CPU module do not match the actual system configuration.	Fatal fault	O	O	O
%MX10.22.2	System operation definition error	Set to "ON" when the Takt period is set to 0.5ms in a system in which two or more common modules are connected in one configuration.	Fatal fault	O	O	O
%MX10.22.3	System DO selection error	Set to "ON" when the SX bus direct-connect module defined in system DO (output) is not a digital output module.	Fatal fault	O	O	O
%MX10.22.4	Duplex selection error	Set to "ON" when an error is found in the equivalent value range specification.	Fatal fault	O	-	-
%MX10.22.5	Fail-soft startup selection error	Set to "ON" when fail-soft startup is enabled if any module is not applicable to fail-soft in the system.	Fatal fault	O	O	-
%MX10.22.6 | %MX10.22.9	Not used			-	-	-
%MX10.22.10	CPU operation definition error	Set to "ON" when the switch setting in the CPU module is different from the CPU number set in the system definition.	Fatal fault	O	O	O
%MX10.22.11	CPU memory boundary definition error	Set to "ON" when the memory space used by an application program exceeds the total memory capacity.	Fatal fault	O	O	-
%MX10.22.12 | %MX10.22.15	Not used			-	-	-
%MX10.23.0	CPU I/O group definition error (for default tasks)	Set to "ON" when an input module is defined for an output module.	Fatal fault	O	O	O
%MX10.23.1	CPU I/O group definition error (for level 0 tasks)					
%MX10.23.2	CPU I/O group definition error (for level 1 tasks)					
%MX10.23.3	CPU I/O group definition error (for level 2 tasks)					
%MX10.23.4	CPU I/O group definition error (for level 3 tasks)					
%MX10.23.5	Direct I/O connect fail-soft definition error	Set to "ON" when fail-soft is set for a module other than the input/output modules.	Fatal fault	O	O	O
%MX10.23.6	Remote I/O master 0 fail-soft definition error	Set to "ON" when an error is found in the fail-soft definition.	Fatal fault	O	O	O
%MX10.23.7	Remote I/O master 1 fail-soft definition error					
%MX10.23.8	Remote I/O master 2 fail-soft definition error					
%MX10.23.9	Remote I/O master 3 fail-soft definition error					
%MX10.23.10	Remote I/O master 4 fail-soft definition error					
%MX10.23.11	Remote I/O master 5 fail-soft definition error					
%MX10.23.12	Remote I/O master 6 fail-soft definition error					
%MX10.23.13	Remote I/O master 7 fail-soft definition error					
%MX10.23.14 %MX10.24.15	Not used			-	-	-

%MX10.24.0	I/O module hold definition error	Set to "ON" when hold is defined for a module other than output modules or the output module set for the system DO.	Fatal fault	O	O	O
%MX10.24.1	I/O initialization error	Set to "ON" when an error is found in the operation setting for the module connected to the SX bus.	Fatal fault	O	O	O
%MX10.24.2 | %MX10.24.15	Not used			-	-	-
%MX10.25.0	Remote I/O master 0 initialization error	Set to "ON" when an error is found in a remote I/O master initialization.	Fatal fault	O	O	O
%MX10.25.1	Remote I/O master 1 initialization error					
%MX10.25.2	Remote I/O master 2 initialization error					
%MX10.25.3	Remote I/O master 3 initialization error					
%MX10.25.4	Remote I/O master 4 initialization error					
%MX10.25.5	Remote I/O master 5 initialization error					
%MX10.25.6	Remote I/O master 6 initialization error					
%MX10.25.7	Remote I/O master 7 initialization error					
%MX10.25.8 | %MX10.25.15	Not used			-	-	-
%MX10.26.0	Processor-link 0 initialization error	Set to "ON" when an error is found in a P/PE-link/FL-net initialization. Processor-link 1 corresponds to the module of link number "9," processor-link 0 corresponds to the module of line number "8."	Fatal fault	O	O	O
%MX10.26.1	Processor-link 1 initialization error					
%MX10.26.2 | %MX10.29.15	Not used			-	-	-

Note: The system definition error factor includes errors that do not occur during normal operation.

(13) Application program error factor %MW10.38, %MW10.39

Address	Name	Description	Level	SPH 300	SPH 200	SPS
%MX10.38.0	Application WDT error	Set to "ON" when the run time for a default task exceeds the preset value of a watchdog timer.	Fatal fault	O	O	O
%MX10.38.1	Application execution error	Set to "ON" when an error has occurred during user program execution that causes "temporary size-over."	Fatal fault	O	O	O
%MX10.38.2 | %MX10.38.10	Not used			-	-	-
%MX10.38.11	FB instance setup error	Set to "ON" when the specified storage address is not found.	Fatal fault	O	O	-
%MX10.38.12	Initial setup value error	Set to "ON" when the preset initial value exceeds the defined range of a storage area.	Fatal fault	O	O	-
%MX10.38.13	SFM boundary definition error	Set to "ON" when a size greater than the maximum SFM capacity value has been defined.	Fatal fault	O	O	-
%MX10.38.14	POU instruction error	Set to "ON" when an error has been found in the POU instruction.	Fatal fault	O	O	-
%MX10.38.15	Task registration error	Set to "ON" when a task registration error has been found.	Fatal fault	O	O	-
%MX10.39.0	Missing level 0 task	Set to "ON" when a task is missing. May clear the error condition by using an application program.	Nonfatal fault	O	O	O
%MX10.39.1	Missing level 1 task					
%MX10.39.2	Missing level 2 task					
%MX10.39.3	Missing level 3 task					
%MX10.39.4	Level 0 task slow-down	Set to "ON" when program execution is deferred and the predefined periodic time is not maintained.	Nonfatal fault (Fatal fault, in the SPS)	O	O	O
%MX10.39.5	Level 1 task slow-down					
%MX10.39.6	Level 2 task slow-down					
%MX10.39.7	Level 3 task slow-down					
%MX10.39.8 | %MX10.39.14	Not used			-	-	-
%MX10.39.15	Takt period monitoring error	Set to "ON" when the Takt period does not match the system definition. May clear the error condition by using an application program.	Nonfatal fault	O	O	-

Note: The system definition error factor includes errors that do not occur during normal operation because D300win suppresses them from occurring (for example, compile check).

(14) Annunciator relay %MW10.42, %MW10.43

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.42.0	Initial flag	Set to "ON" at the first startup after program download and at initial startup (cold start). This flag never is set to "OFF" during operation.	O	O	O
%MX10.42.1	Power-off flag	Set to "ON" when a power-off condition has occurred in the preceding session.	O	O	-
%MX10.42.2 | %MX10.42.13	Not used		-	-	-
%MX10.42.14	Dummy module flag	Set to "ON" when more than one dummy module has been installed in one system.	O	O	-
%MX10.42.15	Processor bus access disable flag	Set to "ON" when the processor bus is disabled.	O	O	O
%MX10.43.0	Level 0 start flag	Set to "ON" during the first execution of level 0 task.	O	O	-
%MX10.43.1	Level 1 start flag	Set to "ON" during the first execution of level 1 task.	O	O	-
%MX10.43.2	Level 2 start flag	Set to "ON" during the first execution of level 2 task.	O	O	-
%MX10.43.3	Level 3 start flag	Set to "ON" during the first execution of level 3 task.	O	O	-
%MX10.43.4 | %MX10.43.14	Not used		-	-	-
%MX10.43.15	Default task start flag	Set to "ON" during the first execution of default task.	O	O	-

(15) Duplex annunciator relay %MW10.46, Duplex operation mode %MW10.47 (Read only)
(Not supported by SPH200, and SPS)

Address	Name	Description
%MX10.46.0	Duplex continuation start flag	Set to "ON" during operation in the duplex mode and when the operating system is switched to a waiting one. (The CPU that is changed over from waiting to running mode)
%MX10.46.1 | %MX10.46.15	Not used	
%MX10.47.0 | %MX10.47.3	Duplex logical CPU number	Indicates a 4-bit CPU logical number in the duplex operation mode (0-7). Allows you to recognize the default operating CPU taken over by the waiting CPU. It is undefined in the mode other than as "duplex."
%MX10.47.4 | %MX10.47.7	Not used	
%MX10.47.8	Duplex annunciator relay mode 0	Set to "ON" when an annunciator relay has been enabled for a pair of CPUs 0 and 1 during operation in the duplex mode.
%MX10.47.9	Duplex annunciator relay mode 1	Set to "ON" when an annunciator relay has been enabled for a pair of CPUs 2 and 3 during operation in the duplex mode.
%MX10.47.10	Duplex annunciator relay mode 2	Set to "ON" when an annunciator relay has been enabled for a pair of CPUs 4 and 5 during operation in the duplex mode.
%MX10.47.11	Duplex annunciator relay mode 3	Set to "ON" when an annunciator relay has been enabled for a pair of CPUs 6 and 7 during operation in the duplex mode.
%MX10.47.12 | %MX10.47.15	Not used	

(16) Resource configuration/operation information
%MW10.48, %MW10.49 (Read only)
(only for SPH300)

The information can be used to recognize the current status of system (CPU module) operation in the duplex or single mode. Resource configuration information can be used only in the duplex mode.The types of status listed below are valid when the associated bits (%MW10.50, 51) with resource operation/fault information have been set to “ON.”

< In the duplex mode >

Resource operation information	Resource running information	Resource status
OFF	OFF	Waiting CPU stopped
ON	OFF	Operating CPU stopped
ON	ON	Operating CPU running
OFF	ON	Waiting CPU running

< Resource operation information >

Address	Name	Description
%MX10.48.0	Operating CPU 0 running	Set to "ON" when the operating CPU is running in the duplex mode (in a mode other than "duplex," undefined).
%MX10.48.1	Operating CPU 1 running	
%MX10.48.2	Operating CPU 2 running	
%MX10.48.3	Operating CPU 3 running	
%MX10.48.4	Operating CPU 4 running	
%MX10.48.5	Operating CPU 5 running	
%MX10.48.6	Operating CPU 6 running	
%MX10.48.7	Operating CPU 7 running	
%MX10.48.8 | %MX10.48.15	Not used	

< Resource runnig information >

Address	Name	Description
%MX10.49.0	CPU 0 running	Set to "ON" when the associated CPU module connected to the SX bus is running.
%MX10.49.1	CPU 1 running	
%MX10.49.2	CPU 2 running	
%MX10.49.3	CPU 3 running	
%MX10.49.4	CPU 4 running	
%MX10.49.5	CPU 5 running	
%MX10.49.6	CPU 6 running	
%MX10.49.7	CPU 7 running	
%MX10.49.8 | %MX10.49.15	Not used	

(17) Resource configuration/fault information
%MW10.50, %MW10.51 (Read only)

The information can be used to recognize the status of other resources (CPU module) by using an application program.

Resource configuration information	Resource fault information	Resource status
OFF	OFF	Nonexistent
ON	OFF	Normal (running or stopped)
ON	ON	Nonfatal fault (running or stopped)
OFF	ON	Fatal fault (stopped or dropped)

< Resourse configuration information >

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.50.0	CPU 0 configuration	Set to "ON" when the associated CPU module connected to the SX bus has been found and its resource operation status is "normal" or "Non-fatal fault." Note: For SPH200, only CPU0 is the target.	O	O	O
%MX10.50.1	CPU 1 configuration				
%MX10.50.2	CPU 2 configuration				
%MX10.50.3	CPU 3 configuration				
%MX10.50.4	CPU 4 configuration				
%MX10.50.5	CPU 5 configuration				
%MX10.50.6	CPU 6 configuration				
%MX10.50.7	CPU 7 configuration				
%MX10.50.8 | %MX10.50.15	Not used		-	-	-

< Resourse fauit information >

Address	Name	Description	SPH 300	SPH 200	SPS
%MX10.51.0	CPU 0 error	Set to "ON" when the associated CPU module connected to the SX bus has been found and its resource operation status is "normal" or "Non-fatal fault." Note: For SPH200, only CPU0 is the target.	O	O	O
%MX10.51.1	CPU 1 error				
%MX10.51.2	CPU 2 error				
%MX10.51.3	CPU 3 error				
%MX10.51.4	CPU 4 error				
%MX10.51.5	CPU 5 error				
%MX10.51.6	CPU 6 error				
%MX10.51.7	CPU 7 error				
%MX10.51.8 | %MX10.51.15	Not used		-	-	-

(18) SX bus configuration information %MW10.52 to %MW10.67 (Read only)

When a module exists on the SX bus and it is running normally or with a nonfatal fault, the SX station number bit for the module is set to “ON.”

Whether the module is normal or in a nonfatal fault is identified by the combination of the configuration error information items.

SX bus configuration information	SX bus fault information	Module status
OFF	OFF	Nonexistent
ON	OFF	Normal
ON	ON	Nonfatal fault
OFF	ON	Fatal fault or dropped

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bit Address
%MW10.52	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		← Not used
%MW10.53	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
%MW10.54	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
%MW10.55	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	
%MW10.56	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	
%MW10.57	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	
%MW10.58	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	
%MW10.59	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
%MW10.60	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	
%MW10.61	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	
%MW10.62	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	
%MW10.63	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	
%MW10.64	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	
%MW10.65	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	
%MW10.66	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	
%MW10.67		254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	

(19) SX bus fault information %MW10.68 to %MW10.83 (Read only)

When there is a module on the SX bus and it is subject to a fatal or nonfatal fault, the bit corresponding to the SX bus station number of the module is set to “ON.”

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bit Address
%MW10.68	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		← Not used
%MW10.69	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
%MW10.70	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
%MW10.71	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	
%MW10.72	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	
%MW10.73	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	
%MW10.74	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	
%MW10.75	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
%MW10.76	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	
%MW10.77	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	
%MW10.78	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	
%MW10.79	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	
%MW10.80	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	
%MW10.81	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	
%MW10.82	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	
%MW10.83		254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	

(20) SX bus-connected module fail-soft information %MW10.84 to %MW10.99 (Read only)

When fail-soft or individual reset cannot be done for any of the modules connected to the SX bus, the SX station number bit for the module is set to “ON.”

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bit Address
%MW10.84	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		← Not used
%MW10.85	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
%MW10.86	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
%MW10.87	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	
%MW10.88	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	
%MW10.89	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	
%MW10.90	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	
%MW10.91	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
%MW10.92	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	
%MW10.93	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	
%MW10.94	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	
%MW10.95	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	
%MW10.96	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	
%MW10.97	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	
%MW10.98	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	
%MW10.99		254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	

(21) Remote I/O master 0 I/O module configuration/fault information %MW10.128 to %MW10.143 (Read only)

When there is a module that is under remote I/O master 0 control and it is normal or in a nonfatal fault, the SX station number bit for the pertinent module is set to “ON.”

Remote configuration information	Remote fault information	Module status
OFF	OFF	Does not exist
ON	OFF	Normal
ON	ON	Nonfatal fault
OFF	ON	Fatal fault or disconnected

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.128	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.129	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.130	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.131	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.132	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.133	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.134	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.135	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

When there is a module that is under remote I/O master 0 control and it is in a fatal or nonfatal fault, the bit corresponding to the remote station number of the module is set to “ON.”

<Fault information>

%MW10.136	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.137	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.138	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.139	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.140	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.141	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.142	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.143	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

The interpretation of paragraphs (22) through (28) is identical to that of paragraph (21).

(22) Remote I/O master 1 I/O module configuration/fault information %MW10.144 to %MW10.159 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.144	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.145	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.146	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.147	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.148	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.149	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.150	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.151	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
<Fault information>																
%MW10.152	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.153	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.154	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.155	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.156	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.157	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.158	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.159	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(23) Remote I/O master 2 I/O module configuration/fault information %MW10.160 to %MW10.175 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.160	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.161	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.162	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.163	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.164	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.165	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.166	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.167	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
<Fault information>																
%MW10.168	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.169	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.170	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.171	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.172	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.173	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.174	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.175	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(24) Remote I/O master 3 I/O module configuration/fault information %MW10.176 to %MW10.191 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.176	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.177	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.178	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.179	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.180	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.181	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.182	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.183	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.184	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.185	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.186	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.187	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.188	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.189	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.190	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.191	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(25) Remote I/O master 4 I/O module configuration/fault information %MW10.192 to %MW10.207 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.192	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.193	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.194	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.195	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.196	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.197	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.198	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.199	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.200	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.201	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.202	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.203	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.204	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.205	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.206	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.207	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(26) Remote I/O master 5 I/O module configuration/fault information %MW10.208 to %MW10.223 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.208	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.209	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.210	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.211	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.212	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.213	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.214	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.215	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.216	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.217	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.218	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.219	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.220	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.221	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.222	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.223	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(27) Remote I/O master 6 I/O module configuration/fault information %MW10.224 to %MW10.239 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.224	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.225	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.226	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.227	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.228	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.229	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.230	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.231	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.232	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.233	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.234	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.235	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.236	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.237	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.238	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.239	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(28) Remote I/O master 7 I/O module configuration/fault information %MW10.240 to %MW10.255 (Read only)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.240	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.241	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.242	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.243	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.244	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.245	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.246	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.247	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.248	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.249	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.250	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.251	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.252	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.253	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.254	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.255	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(29) Configuration information of all module in the resource %MW10.300 to %MW10.315 (Read only)
(only for SPS)

When a module exists on the SX bus and is operating in Normal or Nonfatal Fault condition, the bit that corresponds to the SX station number of this module is set to “ON.”
The judgment of “Normal” or “Nonfatal error” condition is made by combining with the following error information of all modules in the resource.

Configuration information of all modules in a resource	Fault information of all modules in a resource	Module status
OFF	OFF	Does not exist
ON	OFF	Normal
ON	ON	Nonfatal fault
OFF	ON	Fatal fault or disconnected

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bit Address
%MW10.300	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		← Not used
%MW10.301	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
%MW10.302	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
%MW10.303	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	
%MW10.304	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	
%MW10.305	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	
%MW10.306	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	
%MW10.307	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
%MW10.308	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	
%MW10.309	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	
%MW10.310	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	
%MW10.311	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	
%MW10.312	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	
%MW10.313	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	
%MW10.314	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	
%MW10.315		254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	

(30) Fault information of all module in the resource (Read only) %MW10.316 to %MW10.331 (Read only)
(only for SPS)

When a module exists on the SX bus and is in Fatal Fault or Nonfatal Fault condition, the bit that corresponds to the SX station number of this module is set to “ON.”

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	← Bit Address
%MW10.316	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		← Not used
%MW10.317	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
%MW10.318	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	
%MW10.319	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	
%MW10.320	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	
%MW10.321	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	
%MW10.322	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	
%MW10.323	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	
%MW10.324	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	
%MW10.325	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	
%MW10.326	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	
%MW10.327	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	
%MW10.328	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	
%MW10.329	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	
%MW10.330	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	
%MW10.331		254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	

(31) Remote I/O master board 0- I/O module configuration / fault information %MW10.360 to %MW10.375 (Read only)
(only for SPS)

When a remote I/O module is under control of remote I/O mater board 0 and is in Normal or Nonfatal Fault condition, the bit that corresponds to the remote station number of this module is set to “ON.”

Remote I/O configuration information	Remote I/O fault information	Module status
OFF	OFF	Does not exist
ON	OFF	Normal
ON	ON	Nonfatal fault
OFF	ON	Fatal fault or disconnected

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.360	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.361	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.362	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.363	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.364	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.365	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.366	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.367	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

When a remote I/O module is under control of remote I/O mater board 0 and is in Fatal or Nonfatal Fault condition, the bit that corresponds to the remote station number of this module is set to “ON.”

<Fault information>

%MW10.368	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.369	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.370	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.371	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.372	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.373	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.374	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.375	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

How to see the information of (31) to (34) is the same as of (21)

(32) Remote I/O master board 1- I/O module configuration / fault information %MW10.376 to %MW10.391 (Read only)
(only for SPS)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.376	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.377	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.378	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.379	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.380	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.381	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.382	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.383	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.384	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.385	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.386	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.387	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.388	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.389	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.390	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.391	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(33) Remote I/O master board 2- I/O module configuration / fault information %MW10.392 to %MW10.407 (Read only)
(only for SPS)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.392	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.393	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.394	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.395	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.396	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.397	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.398	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.399	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.400	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.401	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.402	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.403	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.404	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.405	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.406	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.407	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(34) Remote I/O master board 3- I/O module configuration / fault information %MW10.408 to %MW10.423 (Read only)
(only for SPS)

<Configuration information>

Word Address ↓	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0 ← Bit Address
%MW10.408	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.409	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.410	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.411	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.412	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.413	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.414	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.415	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

<Fault information>

%MW10.416	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
%MW10.417	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
%MW10.418	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
%MW10.419	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
%MW10.420	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
%MW10.421	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
%MW10.422	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
%MW10.423	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112

(35) SX bus transmission error rate information %MW10.508 to %MW10.511 (Read only)
Executing 100,000 Takts, the number of the Takts where SX bus error occurred is expressed by the ppm. If, of executed Takts, even one Takt is erroneous, the value becomes “10.” These data are updated every 100,000 Takts.

Address	Name	Description
%MX10.508 %MX10.509	Maximum value (lower word) Maximum value (upper word)	Set to the maximum of the error rate values for the SX bus detected by self-CPU module.
%MX10.510 %MX10.511	Current value (lower word) Current value (upper word)	Set to the current of the error rate values for the SX bus detected by self-CPU module.

Note: Various types of system flag information for system memory areas may be referenced from within an application program. Be sure not to use the information for “event variables,” which start the event tasks of an application program (otherwise, some variables may not start the associated task).

1-2-9 Temporary areas

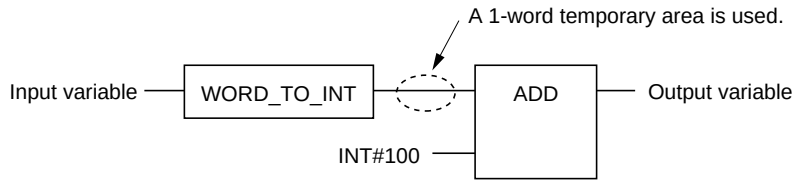
(1) Using temporary areas

Temporary areas, which are automatically generated by D300win when a program is compiled, are used in the manner described below (the area provides 8K words for a high-performance CPU and 4K words for a standard CPU).

1) Connecting function (FCT) terminals

As shown in the figure below, the same number of temporary areas as that of terminal data are used for connecting function terminals. No temporary area is used for connecting function block (FB) terminals and between function (FCT) and function block (FB) terminals because instance areas are used instead.

(Example)



2) Description of polynomials in the ST language

The temporary areas are used for passing the results of calculations.
An expression Y: = (X1 + X2) * X3 is given as a example. In this expression, the temporary area is used to output the result of summation as an input to multiplication. The amount of the temporary area to be used depends on the variable data to be used for a mathematical operation. (Example: INT type = one word, DINT type = two words)

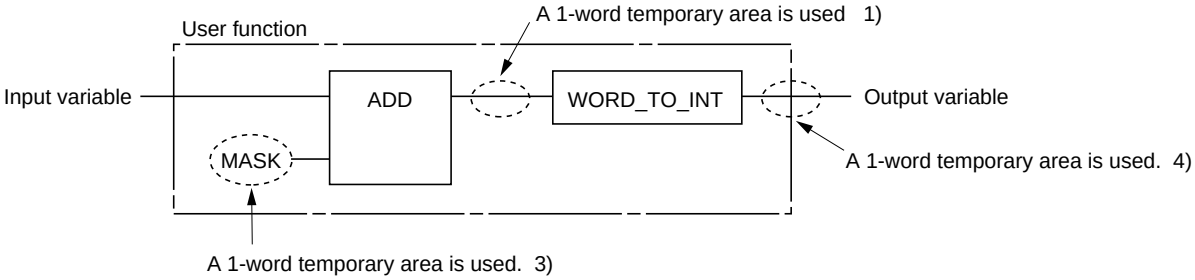
3) Variables in user functions (FCTs)

Every variable used in the user function (those declared in VAR~END_VAR) uses the same number of words in the temporary area as that of the variable itself. The variable includes multi-element variables such as arrays and structures. Declaring, for example, a 1K-word array for a variable allows a 1K word-temporary area to be used.

4) Output variables for user functions (FCTs)

The same number of words in the temporary area is used for the output variable for a function such as that of the variable itself. Declaring, for example, a 1K-word array for a variable allows a 1K-word temporary area to be used.

(Example)



5) Return values from called user functions (FCTs)

When the output variables for user functions are multi-element types such as arrays and structures, the program, which returns the user functions, uses a temporary area. The use of a 1K-word array variable allows a 1K-word temporary area to be used.

(2) Restrictions on a temporary size

Up to 4096 words are allowed for one POU. Up to 8192-word and 4096-word temporary areas can be used simultaneously in one high-performance CPU and standard CPU, respectively. When the amount of the temporary area used by the running CPU exceeds the predefined upper limit, an application execution error (%MX10.38.1 set to "ON") occurs, resulting in CPU shutdown with a fatal fault (%MX10.2.4 set to "ON").

Key-point:

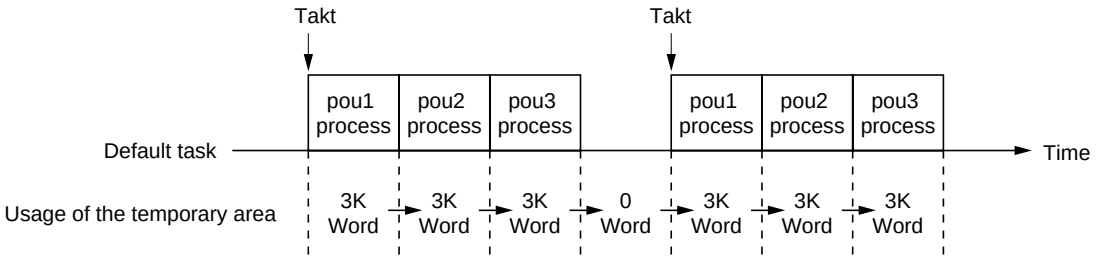
To save the usage of a temporary area:
- Avoid the use of large-size multi-element variables in the user function (FCT).
- Avoid the user functions (FCTs), from which outputs are large-size multi-element variables.
- When a large-size multi-element variable is required, use a function block and process it by VAR_IN_OUT.

(3) Using a temporary area

The temporary area is released for the next POU when a POU has been processed. The next POU uses the temporary area from the top address. If another POU process starts before the current POU process is completed, the temporary area is not released and the remaining area is used from its top. The temporary area is used in the manner described below.

1) When POUs have been assigned only to default tasks

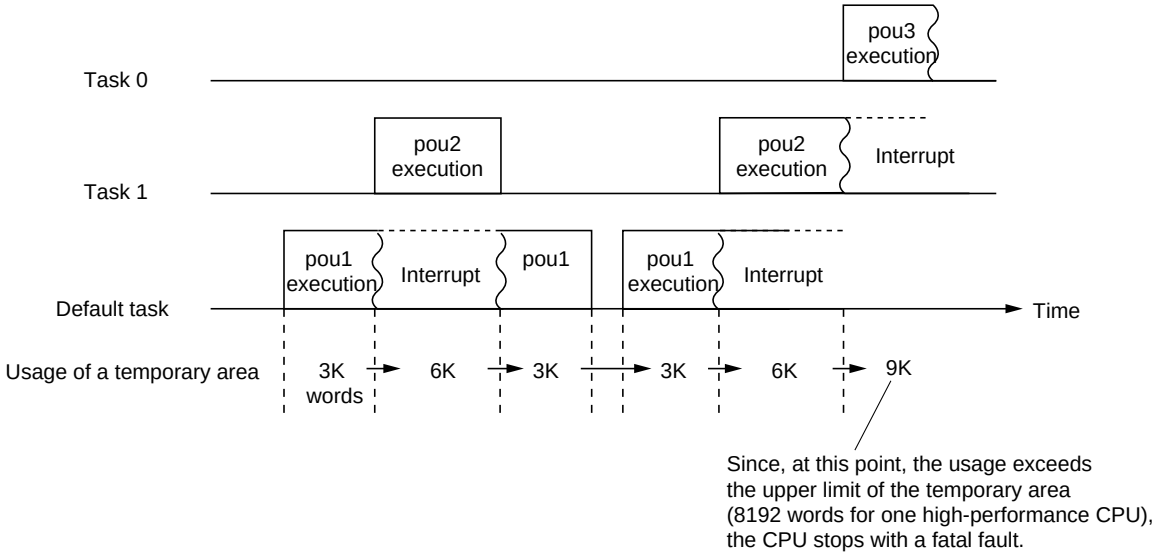
Suppose that three POUs (pou1, pou2, pou3), which have been assigned to their associated default tasks, use a 3K-word temporary area. Since pou1, pou2, and pou3 are executed sequentially as shown in the figure below, the maximum amount of temporary area to be used is 3Kwords or less.



2) When POUs have been assigned to different tasks

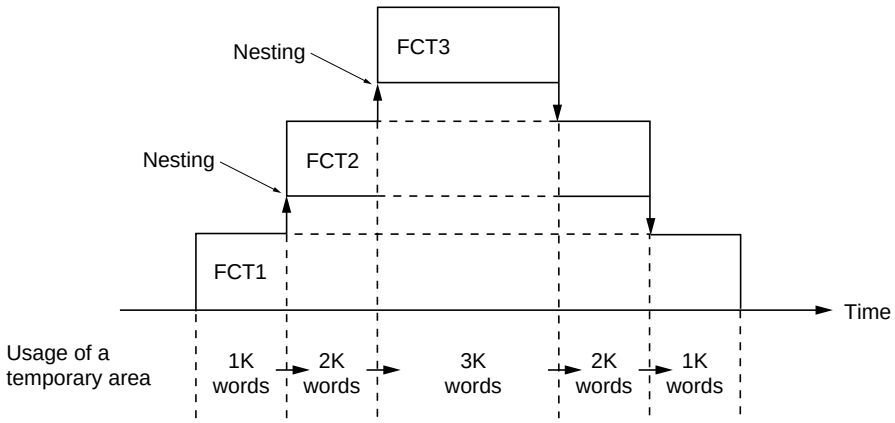
Suppose that three POUs, which have been assigned to the default task (pou1), periodic task 1 (pou2) and 2 (pou3), use a 3K-word temporary area. Since the temporary area is not released if pou2 and pou3 processes are interrupted during pou1 process execution, the total amount of temporary area to be used will be increased.

(Example of a powerful CPU)



3) When the user function process has a nesting structure

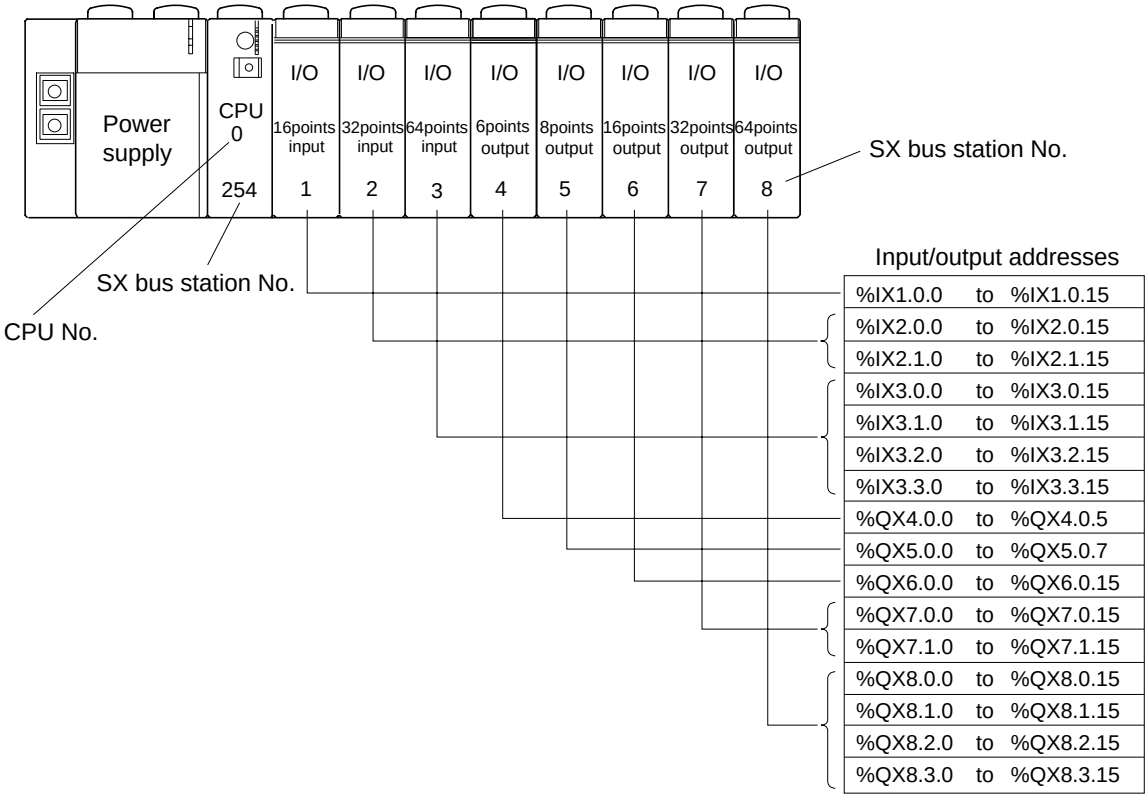
Suppose that three user functions (fct1, fct2, fct3) use a 1K-word temporary area. When fct1 calls fct2 and fct2 calls fct3 (a nesting structure), the total amount of temporary area used is increased.



1-3 Input/output Address Assignment

1-3-1 Address assignment example

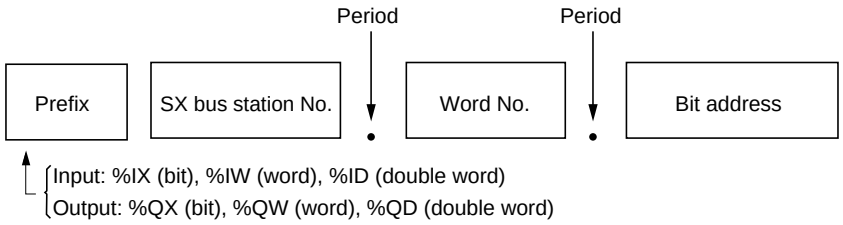
This subsection shows an example of address assignment.
The sample system configuration is illustrated below.



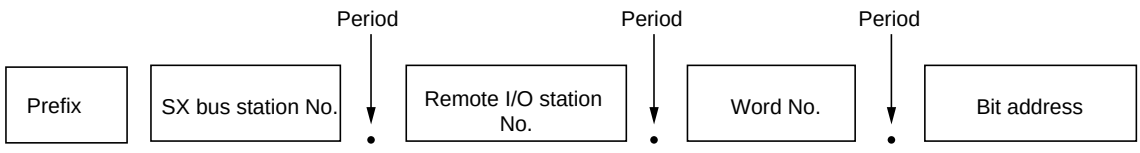
1-3-2 Address assignment conventions

The following conventions are used to assign input/output addresses to the MICREX-SX series CPU modules:

1) SX bus direct-connect I/O



2) Remote I/O



1-3-3 Assigning input/output addresses to an application program

In the MICREX-SX series, input/output addresses are assigned to variables in the form of variable declarations.

<Example of address assignments>

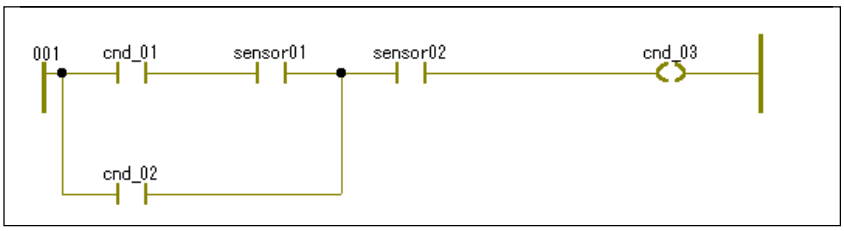
```
VAR
   input    AT %IX1.0.0 :  BOOL;
   output   AT %QX4.0.0 :  BOOL;
END_VAR
```

*Refer to “1-4 Variables” for details.

1-4-1 What is a variable?

To implement application programs in the MICREX-SX Series, it is necessary to assign variables to program code bodies and assign real CPU memory addresses to the variables associated with the instruction codes manually by the user or automatically by the loader.
The use of variables enables you to write application programs with a high-reusability software structure.

<Example of a program code body>



<Variable declarations for the above sample code body>

```
VAR
    sensor01    AT %IX1.0.0 :   BOOL; (*I/O area*)
    sensor02    AT %IX1.0.1 :   BOOL; (*I/O area*)
    cnd_01  :   BOOL;
    cnd_02  :   BOOL;
    cnd_03  :   BOOL;
END_VAR
```

Key-point:

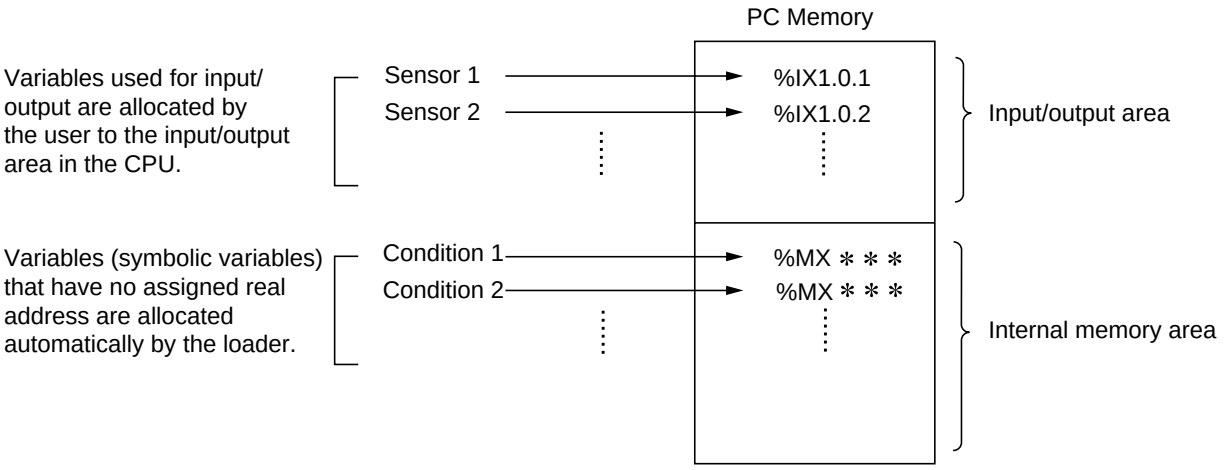
Declare every variable within one line.

1-4-2 Memory assignment

Variables are assigned to the memory in the CPU when they are declared on a variable worksheet and compiled. You can assign variables to memory specifying specific addresses such as those in the input/output area (AT specification) or without specifying an address. Variables without an AT specification are automatically allocated by the loader to real addresses in the CPU's internal memory at the time of compilation.

```
VAR
    sensor1 AT %IX1.0.1 :   BOOL; (*I/O area*)
    sensor2 AT %IX1.0.2 :   BOOL; (*I/O area*)
    condition1  :   BOOL;
    condition2  :   BOOL;
    condition3  :   BOOL;
END_VAR
```

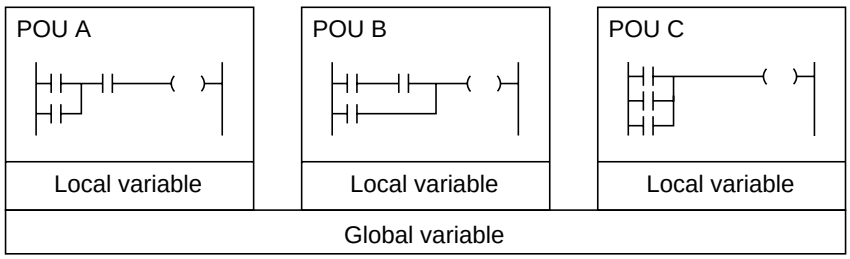
Example of AT specifications
In this example, sensors 1 and 2 are assigned to input/output memory locations. Normally, the user does not need to use the AT specification when allocating memory to a variable that is to be used as auxiliary memory for a program.



To the variables with no address assigned (no AT specification), the loader assigns automatically the CPU memory addresses. This eliminates the need for managing tedious memory assignment.

1-4-3 Local variables and global variables

The MICREX-SX series CPU modules allow the user to split an application program into functional system blocks. For modularization purposes, a program (POU) requires variables that are to be used only within that specific program. To maintain the relationship between programs (POUs), variables that are available globally to those POUs are also required.
In the MICREX-SX series, variables that are available to a POU are called local variables and those which are available to all POUs are called global variables.



1-4-4 Variable declaration

To declare a variable, code it on a variable worksheet in the predefined format using a variable declaration statement.

(1) Types of variable declaration statements

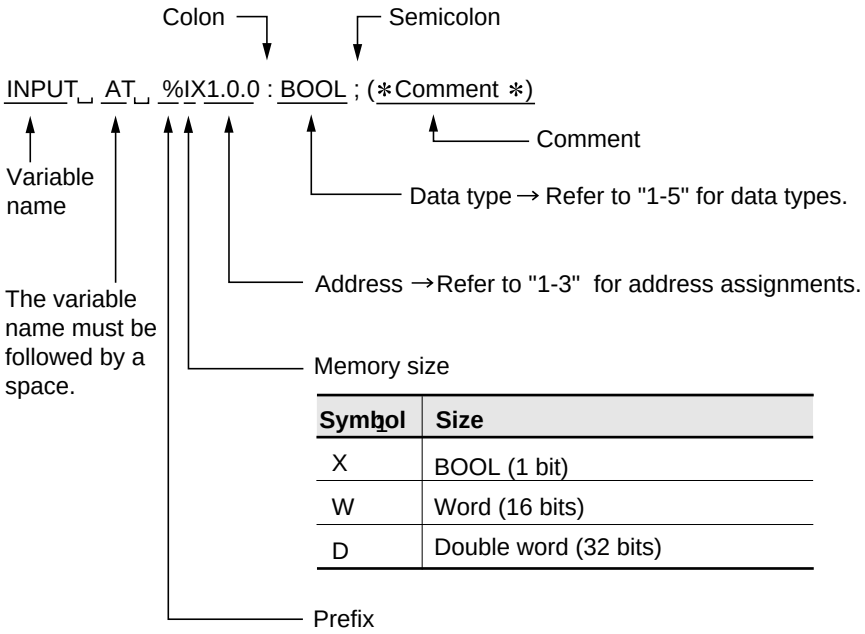
Statement	Description
VAR	Identifies the beginning of the declaration of a local variable.
VAR_INPUT	Identifies the beginning of the declaration of an input variable for a function/function block.
VAR_OUTPUT	Identifies the beginning of the declaration of an output variable for a function block. Used only to declare a symbolic variable.
VAR_IN_OUT	Identifies the beginning of the declaration of an input/output variable for a function block.
VAR_EXTERNAL	Identifies the beginning of the declaration of a global variable to be used in a POU. (Note)
VAR_GLOBAL	Identifies the beginning of the declaration of a variable that is to be available to all programs in a project (global variable).
END_VAR	Identifies the end of a variable block.
RETAIN	Added to the VAR or VAR_GLOBAL statement to declare a retain variable.
AT	Used to assign an immediate address to a variable.

Note: When using a global variable in POUs, it is necessary to declare it in a VAR_EXTERNAL declaration statement on each variable worksheet for the POUs.

(2) AT specification variables (position variables)

An AT specification variable is one which is used by the user to allocate a CPU's memory location to a variable.

The following naming conventions are used to specify an AT specification:



Symbol	Size
X	BOOL (1 bit)
W	Word (16 bits)
D	Double word (32 bits)

Symbol	Meaning
%I	Identifies an external input area.
%Q	Identifies an external output area.
%M	Identifies an internal memory area.

<Example of an AT specification variable declaration>

```
VAR
    INPUT_01    AT %IX1.0.10 : BOOL; (*Bit 10 of input 1*)
    LAMP_01     AT %QX5.0.0  : BOOL; (*Bit 0 of output 5*)
    RUN_F_01    AT %MX10.0.0 : BOOL; (*CPU is running*)
    DATA_I_01   AT %IW2.0    : INT; (*Input two data*)
END_VAR
```

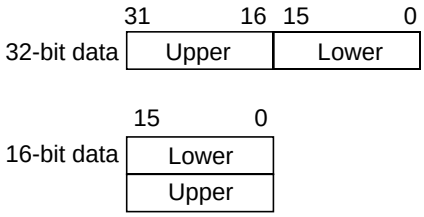
Key-point:

Key point

- To assign 32-bit DINT or DWORD type variables, 32-bit array type variables, and structure type variables to real addresses (AT specification), assign them to even addresses.
  (Example)

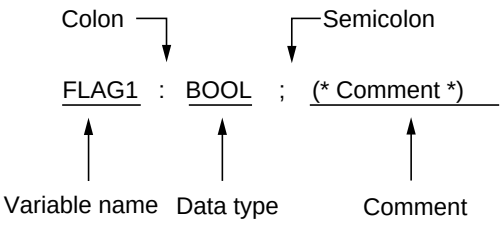
```
VAR
    DATA1   AT    %MD1.0   :   DINT ;
    DATA2   AT    %QD3.0   :   DWORD ;
END_VAR
```

- To divide a 32-bit data type into 16-bit data type for processing, the upper and lower words are assigned as shown below.



(3) Symbolic variables

Symbol variables have no CPU memory addresses assigned by the user. The D300win will automatically allocate memory to symbolic variables. A symbolic variable is declared as follows:



<Example of a symbolic variable declaration>

```
VAR
    DATA_01 :   INT;  (*Comparison data*)
    DATA_02 :   INT;  (*Standard value*)
    FLAG_01 :   BOOL; (*Comparison result flag*)
END_VAR
```

(4) Retain variables

Data declared as a retain variable is reset at cold start (at initial start from D300win) and retained at warm start (at start from D300win or power-on). To declare a retain variable, append a RETAIN statement to the VAR or VAR_GLOBAL statement declaring the variable. Variables that are declared as retain variables are automatically allocated to the retain area (retain memory) in the PC.

<Example of a retain variable declaration>

```
VAR
    DATA_01 :   INT;  (*Comparison data*)
    FLAG_01 :   BOOL; (*Comparison result flag*)
END_VAR

VAR RETAIN
    DATA_02 :   INT;  (*Standard value*)
END_VAR
```

(5) Initialization variable

A variable that is assigned an initial value when the PC application program is started (during the first scan) is called an initialization variable. A variable is declared as an initialization variable and allocated to the PC memory by assigning an initialization value using the ":=" operator. No initialization variable can be specified in the VAR_EXTERNAL declaration statement (for global variables).

<Example of an initialization variable declaration>

```
VAR
    INI_DATA_01 :   INT := 100;
    INI_DATA_02 :   INT := 0;
    ADD_DATA_01 :   INT;
END_VAR
```

Key-point:

Number of initial values assigned to variables
- High-performance CPU NP1PS-32/NP1PS-74/NP1PS-117R: up to 3200 (fixed value)
- Standard CPU NP1PH-16: 512 (default value)
- Standard CPU NP1PH-08: 153 (default value)

For standard CPUs, the number of variables to which initial values may be assigned depends on the memory space for setting initial values in the user memory area. Refer to "1-2-7 Initialization memory area" for details.

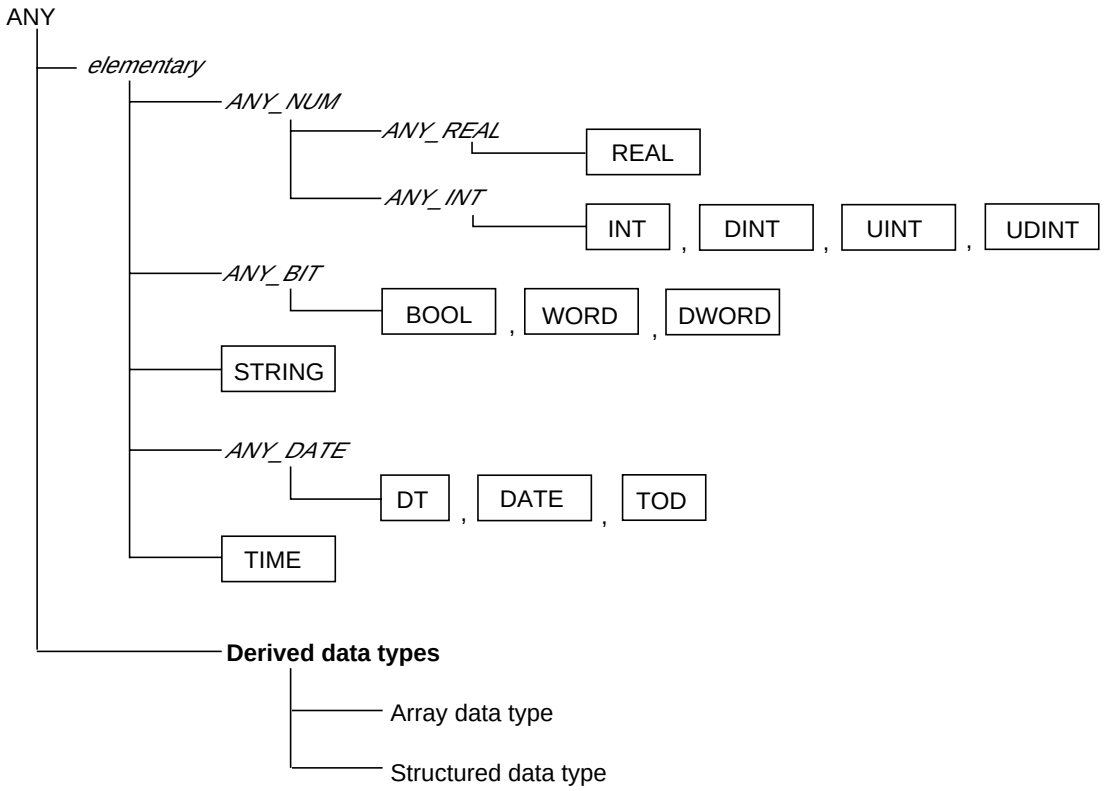
(6) Restrictions on variable names

- Only a symbol "_" (1-bit) may be used for a variable name. Note that no "_" can follow the variable name.
- No 1-bit digit can be used at the top of a variable name.
- Up to 30 1-bit characters (15 2-bit characters) can be used for a variable name.
- No 2-bit space can be used for a variable name.
- The words reserved for the system (data type names, instruction names, etc.) cannot be used. Refer to "Appendix 6. List of Reserved Words."

1-5-1 Organization of data types

The data types that are supported by the MICREX-SX series can be depicted in a tree form as shown in the figure below.

ANY is a data type that can hold a basic data type or any of derived data types.



Data type names in italics represent generic data types and boxed data type names represent basic data types.

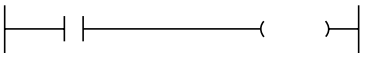
1-5-2 Basic data types

Basic data types refer to data whose value range and bit count are defined by IEC 61131-3. The MICREX-SX series CPU modules support the following basic data types:

No.	Keyword	Data Type	No. of bits	Value Range
1	BOOL	Boolean	1	0 or 1
2	INT	Integer	16	-32,768 to 2,767
3	DINT	Double integer	32	-2,147,483,648 to 2,147,483,647
4	UINT	Unsigned integer	16	0 to 65,535
5	UDINT	Unsigned double integer	32	0 to 4,294,967,295
6	REAL	Real	32	$-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ (single-precision floating point)
7	TIME	Duration	32	SPH: 0ms to 4,294,967,295ms (0ms to 49 days, 17:02:47s295ms) SPS: 0ms to 2,147,483,647ms
8	DATE	Date	32	January 1st, 1970 to February 7th, 2106 (Note 1)
9	TOD	Time of day	32	0:00:00 to 23:59:59 (Note 1)
10	DT	Date and time of day	32	January 1st 0:00:00, 1970 to February 7th 6:28:15, 2106 (Note 1)
11	STRING	Variable-length character string	-	-
12	WORD	Bit string of length 16	16	16#0000 to 16#FFFF (Note 2)
13	DWORD	Bit string of length 32	32	16#00000000 to 16#FFFFFFFF (Note 2)

Note: 1) Not supported by SPS.
2) 16# represents a hexadecimal number.

1) BOOL (Boolean)
The minimum unit of data that can be represented by a single memory bit (0 or 1).


← A contact or coil in LD language is an example of BOOL data.

Examples: BOOL#0, BOOL#1, TRUE, FALSE

2) INT (Integer)
An INT occupies 16 memory bits (1 word) and handles integer values from -32768 to 32767.
Examples: -1000, 0, 12345, INT#1234, INT#16#FF0F

3) DINT (double integer)
A DINT occupies 32 memory bits (1 double word) and handles integer values from -2147483648 to 2147483647.
Examples: DINT#12345678, DINT#16#F000F00F

4) UINT (unsigned integer)
A UINT occupies 16 memory bits (1 word) and handles integer values from 0 to 65535.
Examples: UINT#1000, UINT#16#FFFF

5) UDINT (unsigned double integer)
A UDINT occupies 32 memory bits (1 double word) and handles integer values from -0 to 4294967295.
Examples: UDINT#100000, UDINT#16#111FFFF

6) REAL (real <single-precision floating point>)
A REAL occupies 32 memory bits and handles real values of
$-2^{128} < N <= -2^{-126}$, 0, $-2^{-126} <= N < 2^{128}$
(-3.4028235E+38 to -1.1754945E-38, 0, 1.1754945E-38 to 3.4028235E+38).

Note 1: The value range of REAL type data with guaranteed accuracy is basically 6 significant digits. However, this may be lowered to 5 digits or less in some numeric FCTs. For details, refer to the individual instruction descriptions.

Examples:
-10.035, 0.0, 0.2345, REAL#10.0
-1.34E-12, 1.0E+6, 1.234E6
-1.34e-12, 1.0e+6, 1.234e6
} ← May be represented in exponential formats.

Note 2: Real values that are smaller than the minimum representable value ($\pm 2^{-126}$) are handled as zeros (0).

7) TIME (duration)
This deals with the set value or current value of timer. When SPH is used, time data in the range from 0 to 4,294,967,295ms can be expressed; when SPS is used, in the range from 0 to 2,147,483,647ms. Units are d (day), h (hours), m (minutes), s (seconds) and ms (milliseconds). You can combine them in various ways.

<Examples>

No.	Type		Example
1	Without underscores	Basic format	TIME#14ms time#14s
2		Abbreviated format	T#14ms T#14s T#14m T#14h t#25h15m t#5d14h12m18s3ms
3	With underscores	Basic format	TIME#25h_15m time#5d_14h_12m_18s_3ms
4		Abbreviated format	t#5d _14h_12m_18s_3ms t#25h _15m

8) DATE (date) Not supported by SPS.
The DATE data type handles date (year, month, day) data. A DATE occupies 32 memory bits and can represent date data from the calendar years 1970 to 2106.

9) TOD (time of day) Not supported by SPS.
The TOD data type handles time of day (hour, minute, second) data. A TOD occupies 32 memory bits and can represent time data from 0:00:00 to 23:59:59. The data type of TIME_OF_DAY is the same as that of DT.

10) DT (date and time of day) Not supported by SPS.
The DT data type handles date and time (year, month, day, hour, minute, second) data. A TOD occupies 32 memory bits and can represent date and time data from the calendar years 1970 to 2106. The data type of TIME_AND_DAY is the same as that of DT.

<Examples for DATE, TOD, and DATA:>

No.	Type		Example
1	DATE type	Basic format	DATE#1984-06-25 date#1984-06-25
		Abbreviated format	D#1984-06-25 d#1984-06-25
2	TOD type	Basic format	TIME_OF_DAY#15:36:55 time_of_day#15:36:55
		Abbreviated format	TOD#15:36:55 tod#15:36:55
3	DT type	Basic format	DATE_AND_TIME#1984-06-25-15:36:55 date_and_time#1984-06-25-15:36:55
		Abbreviated form	DT#1984-06-25-15:36:55 dt#1984-06-25-15:36:55

11) STRING (variable-length character string)

<In the case of SPH>

Character strings are represented in Fuji Electric's original code which is based on the Shift JIS coding system. Whereas Shift JIS strings are mixtures of 8- and 16-bit codes, this coding system extends 8-bit code into 16-bit code so that the length of a single character is fixed at 16 bits. Strings have a variable length and can be as long as 64 characters.
A NULL code (X'0000') is automatically appended to the end of each string. Consequently, when a character string is declared, a memory space equal to the length of the character string plus one character, 65 words, is reserved.

<Examples:>

No.	Example	Description
1	''	String of length 0 (null string)
2	'A'	A string that is made up of one character A.
3	'ABC'	A string that contains characters A, B, and C.
4	' '	A string that contains one blank character.

<In the case of SPS>

Shifted JIS code. Maximum number of characters is 80 when single-byte characters are used and 40 when double-byte characters are used.

12) WORD (bit string of length 16)

A WORD has a size equal to 16 BOOLs (16 bits), each of which represents the ON or OFF state or 1 or 0.

13) DWORD (bit string of length 32)

A WORD has a size equal to 32 BOOLs (32 bits), each of which represents the ON or OFF state or 1 or 0.

<Examples of WORD and WORD:>

No.	Type	Example
1	Binary representation	WORD#2#1010111110101111 DWORD#2#11110000111100001010111110001111
2	Hexadecimal representation	WORD#16#0F0E DWORD#16#FFFF000F

<About BCD data>

BCD is not defined as a data type in IEC. BCDs are represented as an encoding option of WORD or DWORD. The MICREX-SX series CPU modules handle BCD as unsigned data.

Data Length	Value Range
16 bits (WORD)	0 to 9999
32 bits (DWORD)	0 to 99999999

1-5-3 Derived data types

Derived data types are defined by the user or PC manufacturers. They are defined using “Data Types” in the project tree.

• Array data types
  1-dimensional array
  2-dimensional array (array of array)

The MICREX-SX series CPU modules support the following derived data types:

• Structured data types
  Structure
  Array of structures
  Structure of arrays, structure of structures

(1) Array data types

An array is made up of two or more elements of the same data type. Arrays may be either 1- or 2-dimensional arrays.

<Example of using a 1-dimensional array data type>

<Sample data type definition>

Data type “array1” is made up of 10 integers that are indexed from 1 to 10.

```
TYPE
    array1  :   ARRAY[1..10] OF INT;
END_TYPE
```

Note: Signed integers are legitimate values. Consequently, negative integers such as “-10” are allowed.

<Example of a variable declaration>

The data type of variable “file1” is array1.

```
VAR
    file1   AT %MW3.0   :   ARRAY1;
END_VAR
```

Note: To specify a 32-bit array of data type such as DINT or DWORD in the AT statement, assign even addresses.

1	INT type data
2	INT type data
3	INT type data
4	INT type data
5	INT type data
6	INT type data
7	INT type data
8	INT type data
9	INT type data
10	INT type data

To access this data item, code file 1[15].

Note: Any attempt to access an item beyond the valid index range (e.g., file 1[11]) will simply be ignored with no error being flagged.

<Example of using a 2-dimensional array data type>

<Sample data type definition>

Data type “x_data” is made up of 10 integers that are indexed from 1 to 10 and data type “y_data” is made up of 3 instances of “x_data,” indexed from 1 to 3.

```
TYPE
    x_data  :   ARRAY[1..10] OF INT;
    y_data  :   ARRAY[1..3] OF x_data;
END_TYPE
```

<Example of a variable declaration>

The data type of variable “file2” is y_data.

```
VAR
    file2   :   y_data;
END_VAR
```

	1	2	3
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

To access this data item, specify file2 [3] [5].

(2) Structured data types

A structured data type is made up of two or more similar or different data types. Structured data type is used in situations where data items of different data types are necessary for a single operation. Structured data type may be contained in an array (array of structures) or contain arrays (structures of arrays).

<Examples of using structured data types>

<Sample data type definition>

The structured data type “machine” consists of four data types, namely, x_pos (REAL), y_pos (REAL), depth_pos (INT), and rpm (INT).

```
TYPE
    machine:
    STRUCT
        x_pos   :   REAL;
        y_pos   :   REAL;
        depth   :   INT;
        rpm     :   INT;
    END_STRUCT;
END_TYPE
```

<Sample variable declaration>

The data type of variable "hole_pro" is machine.

```
VAR
    hole_pro  :  machine;
END_VAR
```

<Access method>

Variables of a structured data type are accessed in the form of <variable name>.<member name>. Examples of <member name> in the above example are x_pos, y_pos, depth, and rpm. To access the rotational speed in hole_pro, you specify hole_pro. rpm.

<Array of structures>
Structures may be contained in an array.

<Sample data type definition>

The example shown below defines an array of 10 machines, indexed from 1 to 10, which are defined in the above example.

```
TYPE
    machine:
    STRUCT
        x_pos    :   REAL;
        y_pos    :   REAL;
        depth    :   INT;
        rpm      :   INT;
    END_STRUCT;
    array3  :   ARRAY[1..10] OF machine;
END_TYPE
```

<Sample variable declaration>

The data type of variable “num_hole_pro” is array 3.

```
VAR
    hole_pro     :  machine;
    rhole_pro : array3;
END_VAR
```

<Access method>

Structures in an array are accessed in the form of <variable name>[<array index>].<member name>. Examples of <member name> in the above example are x_pos, y_pos, depth, and rpm. To access the rotational speed of the 5th machine in num_hole_pro, you specify num_hole_pro [5].rpm.

<Structure of arrays>
Array is defined as an element of a structure.

<Sample data type definition>

The example shown below defines a structure “machine2” of which the “depth” member is an array of depths. of depths.

```
TYPE
    pattern  :  ARRAY[1..10] OF INT;
    machine2 :
  STRUCT
        x_pos    :   REAL;
        y_pos    :   REAL;
        depth    :   pattern;
        rpm      :   INT;
  END_STRUCT;
END_TYPE
```

<Sample variable declaration>

The data type of variable “hole_pro_02” is machine2.

```
VAR
    hole_pro     :  machine;
    num_hole_pro :  array3;
    hole_pro_02  :  machine2;
END_VAR
```

<Access method>

An array in a structure is accessed in the form of <variable name>.<member name>[<array index>]. Examples of <member name> in the above example are x_pos, y_pos, depth, and rpm. To access the 5th depth in the member “depth” of “hole_pro_02,” you specify hole_pro_02.depth[5].

<Structure initialization>

No variable of a structured data type can be initialized in a variable declaration. Structured data type variables must be explicitly initialized in the program (code body).

<Sample variable declaration>

```
VAR
    hole_pro     :  machine;
    hole_def     :  BOOL := TRUE;
END_VAR
```

<Sample initialization program in ST language>

```
IF  hole_def  THEN
    hole_pro.depth := 15;
    hole_pro.rpm   := 3000;

    hole_def    := FALSE;
END_IF
```

(3) Restrictions on derived data types

- Derived data types, excluding arrays, can be assigned no initial values in their declarations.
- Initialize array/structure data type variables by an application program.
- No derived data type variable can be overwritten.
- Neither array data type variables nor structure data type variables can be monitored on the program. Therefore, when debugging a program that references array data type or structure data type variables, it is necessary to register them with the watch list and debug the program using the watch list.
- No derived data type can contain a STRING.
- If the value for a variable is out of the range of array elements specified in the data-type definition when array numbers are specified using variables, data at the highest or lowest array element number are accessed.

1-6 Tasks

Tasks determine the sequence (time schedule) of program execution. The MICREX-SX series CPU modules use three types of tasks: default task for cyclic processing, periodic tasks, and event tasks.
POUs that will always be executed need be assigned to tasks so that their execution sequence can be determined.

1-6-1 Task specifications

Item	Specification
Task type	Default task (cyclic processing) Periodic task Event task
Number of tasks	1 (default) + 4 (periodic and event tasks)
Task priority	SPH: 0> 1> 2> 3> default, SPS: 0> 1> 2> 3> ; default (Note)

Note: When SPS is used, the task of level 3 is given the same priority as default task.

1-6-2 Types and operations of tasks

1) Default task

- Always repeat execution in synchronization with Takt. Assign POUs requiring no responsibility and periodicity in arithmetical operations.
- Two or more POUs may be assigned to the default task.

Note: A user WDT is the timer which monitors the execution time of the default task. It checks the time when execution has be done. When no default task is used, the CPU processes the tasks equivalent to the default ones to execute internal processes such as the user WDT check.

2) Periodic task

- A periodic interrupt task is executed once at a predetermined interval (0.5ms, 1ms to 32s). It is assigned to POUs and filters that require high responsiveness to adjust to the speed of the control target and POUs such as integral instructions which need be executed at predetermined intervals.
- A period interrupt task is given a priority of 0 to 3 (0 has the highest priority).
- Multiple POUs can be assigned to one task.
- Two or more POUs can be assigned to a periodic interrupt task (only for SPH).
  When SPS is used, fixed cycle task asynchronously interrupts the currently executed task.

Note: A Takt period is an SX bus communication period. For the Tact period, 0.5ms, 1ms, ..., and 10ms may be set. Note that when a standard CPU is used, 0.5ms cannot be specified. The Takt period depends on the scale of system configuration (the numbers of I/O points, remote I/O master modules, communication modules, and CPU modules) and the number of application program executable steps. A 0.5ms Takt period may be executed under the condition of a powerful single CPU, 256 or less I/O modules connected to the SX bus and no remote I/O and communication modules. In the standard CPU system, setting a Takt period to 0.5ms causes “system operation definition error” to occur, resulting in CPU shutdown with a fatal fault.(Refer to the appropriate appendix for calculating a Takt period.)

3) Event task

- An event task is executed once each time a specified BOOL variable turns to “1” It is assigned to a POU that handles interrupts from a communications module or high-speed counter module.
- Two or more POUs can be assigned to an event task.

1-6-3 Example of periodic task operation

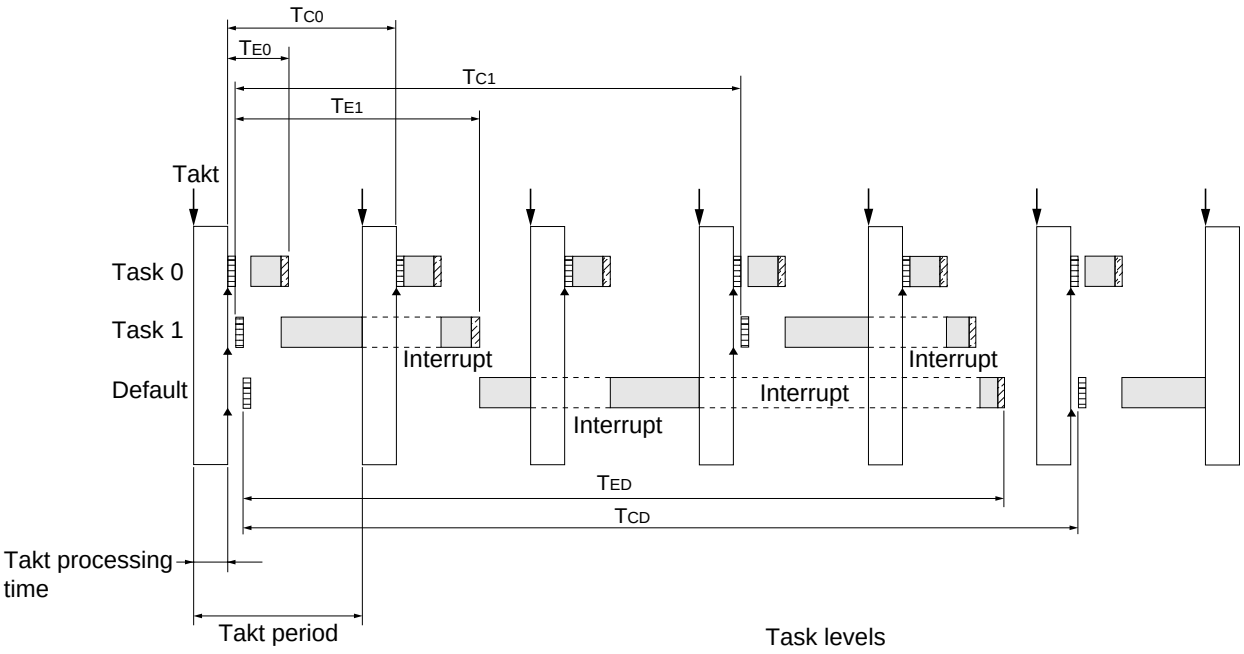
Example of the operation of periodic task when SPH is used is explained under the following operating conditions.

<Operating conditions>

Task priority: Task 0 > Task 1 > Default task (cyclic)
Takt period: 1 ms
Task type: Task 0: Periodic task (1 Takt period)
Task 1: Periodic task (3 Takt periods)
Default task: Asynchronous with the Takt period

Note: The task period of the periodic tasks must be set to an integral multiple of the SX bus Takt period.

<Task operation>



Task levels

Task 0 > Task 1 > Default

TE0: Task 0 execution time

TC0: Task 0 execution period

TE1: Task 1 execution time

TC1: Task 1 execution period

TED: Default task execution time

TCD: Default task execution period

▲ : Task activation request

▥ : Data input processing

▨ : Data output processing

The default tasks run while no periodic task or event task is operating. (They start in synchronization with a Takt period.) Be sure to adjust the run times to the start periods of upper tasks so that the run times may be reserved for default tasks. (Otherwise, a user WDTUP or upper task may be delayed.)

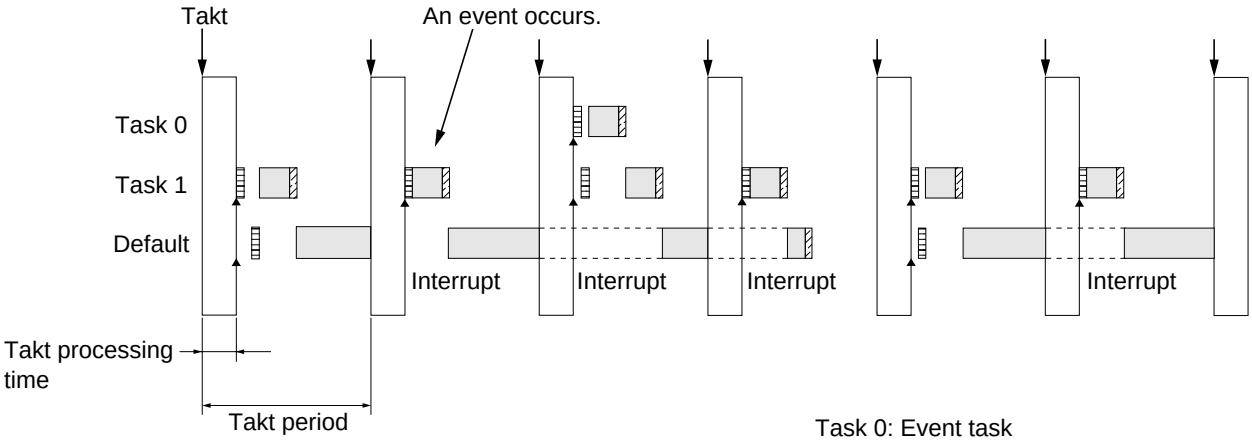
1-6-4 Example of event task operation

An example of operation of an event task is explained below.

<Operating conditions>

Task priority: Task 0 > Task 1 > Default task (cyclic)
Takt period: 1 ms
Task type: Task 0: Event task
Task 1: Periodic task (1 Takt period)
Default task: Asynchronous with the Takt period

<Task operation>



Task 0: Event task
Task 1: Periodic task

Task levels
Task 0 > Task 1 > Default

▲ : Task activation request

: Data input processing

: Data output processing

An event task is not started immediately when an event occurs but at the beginning of the next Takt period after the event is recognized.

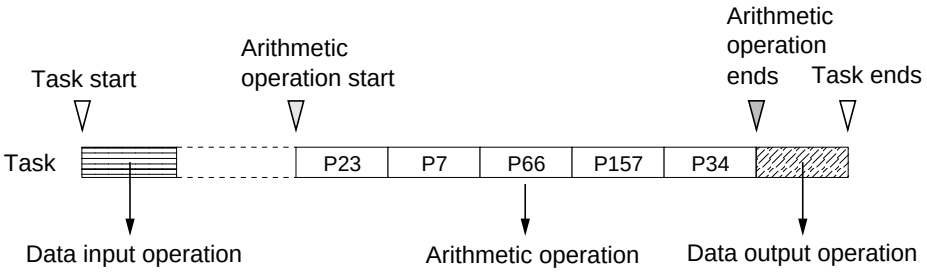
Monitoring the run times and run periods of tasks

The run time and run period of a task may be monitored on the resource information screen displayed from the "D300win Resource Control" dialog box.
Task run time: The time after input to the task starts until output from the task has been finished.
Task run period: The time after input to the task starts until input to the next task starts.

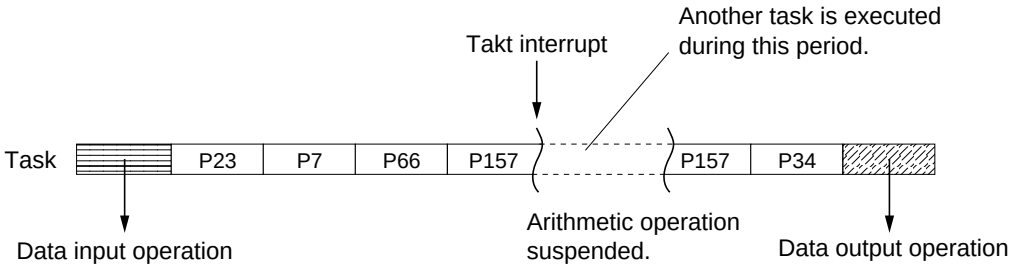
The task run period appears on the open resource information screen after the task is executed two times. When the CPU stops/starts or the CPU is switched between the operating and waiting sides in the duplex mode while the resource information screen is open, measurement stops temporarily and then restarts.
Before the run period can appear on the screen, the task must have been executed two times after the CPU stopped/started or was switched between the operating and waiting sides.

1-6-5 Task interrupt processing

The processing of a task is divided into data input processing, arithmetic operation, and data output processing as illustrated below. A task operation sequence is considered to terminate when all of these operations have completed.
Takt interrupts can be generated via the SX bus during the arithmetic and data output operations (no Takt interrupt can occur during the data input operation).



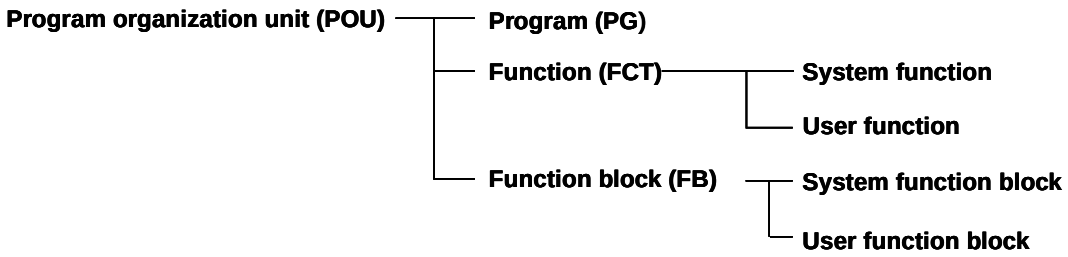
When a Takt interrupt occurs during the arithmetic operation, the operation is interrupted as illustrated in the figure below.
When a Takt interrupt occurs, the system checks the startup conditions for a new task. If a task startup request is present, the system performs the data input operation for that task and starts the task having the highest priority. Consequently, another task is likely to be executed while the current task is suspended as shown in the figure below.
When the arithmetic operation ends, the system checks the time until the next Takt interrupt occurs and, if it is longer than the time required to perform the data output operation, performs the data output processing and terminates the task. If the time to the next Takt interrupt is shorter than the time required to perform the data output operation, the system keeps the task suspended and executes no output operation for the task. The data output operation is carried out only after the system executes the next Takt processing. Since the time up to the next Takt interrupt is computed at the end of the arithmetic operation, no Takt interrupt can occur during the execution of the data output operation.



1-7 Program Organization Units (POUs)

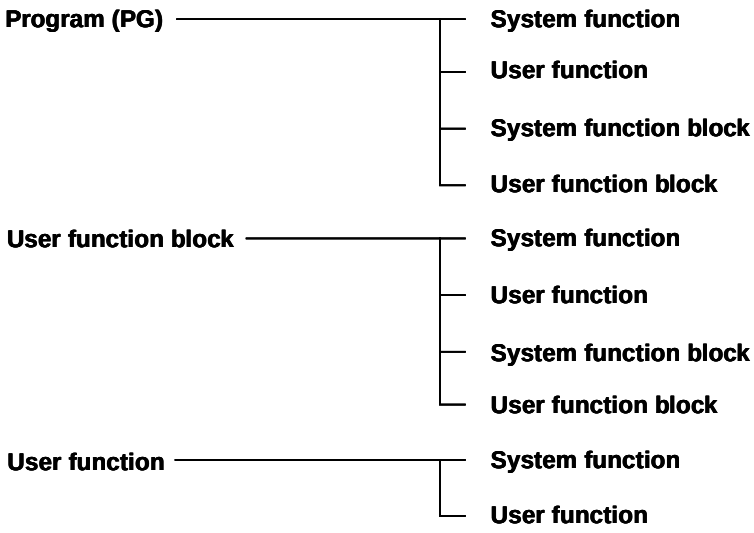
The program organization units include functions (FCTs), function blocks (FBs), and programs (PGs). Programs are programmed by the user. Functions and function blocks may be supplied by the manufacturer or made by the user.

<Types of program organization units>



No program organization unit can be defined recursively. No function block can be called from a function. However, functions can be called from a function block.

<Types of callable POUs>



<Number of variables that can be used in a POU>

High-performance CPU

	Program	User Function	User Function Block
VAR	8000 (Note 1)	4,096 words	256 words in total
VAR_INPUT	-	2,048 words	
VAR_OUTPUT	-	-	
(Note 2) VAR_IN_OUT	-	-	
VAR_EXTERNAL	8000 (Note 1)	-	8000 (Note 1)

Standard CPU

	Program	User Function	User Function Block
VAR	8000 (Note 1)	4,096 words	Total: Number of words occupying the user FB instance area. The default values are shown below. NP1PH-16: 4098 words NP1PH-08: 2048 words
VAR_INPUT	-	1,024 words	
VAR_OUTPUT	-	-	
VAR_IN_OUT	-	-	
VAR_EXTERNAL	8000 (Note 1)	-	8000 (Note 1)

Note: 1) The total of the POUs that are assigned to one resource. This is 8000 (15000 for V2 or later versions of D300win) irrespective of the data type of variables.
2) One VAR_IN_OUT declaration uses two words.

1-7 Program Organization Units (POUs)

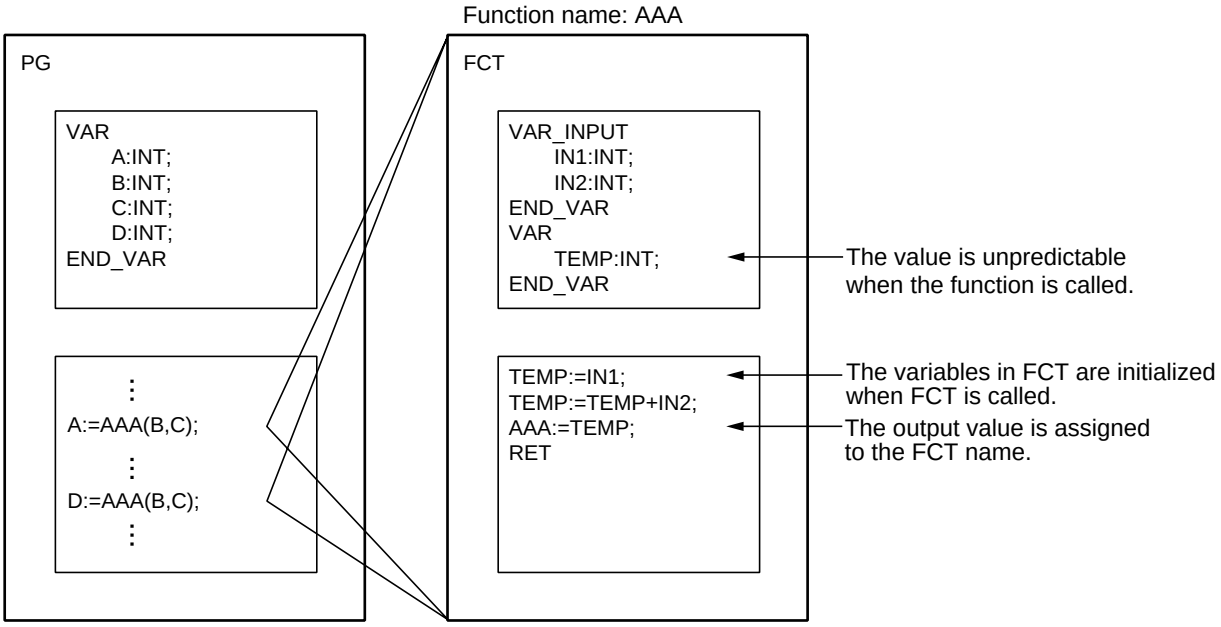
1) Function

A function is a program organization unit that, when executed, generates a single data item (when SPH (Note), including multiple elements such as an array or structure).

A function has no internal state, that is, executions of a function with the same input parameters always generate the same output. Predefined functions can be used in other programs, functions, and function blocks.

The value of variables that are used only within a function are unpredictable when the function is called.

Note: When SPS is used, neither STRING type data nor structure/array type data can be used for input parameters or the variables in functions.



2) Function block

A function block is a program organization unit that, when executed, generates one or more data items. A function block can have two or more copies of data which are called instances. Each instance is given an identifier called the instance name. An instance has output, internal, and input variables.

Some of the output and internal variables must retain their contents until the function block is called next time. Consequently, a function block does not always generate the same output even when it is called multiple times with the same input parameters.

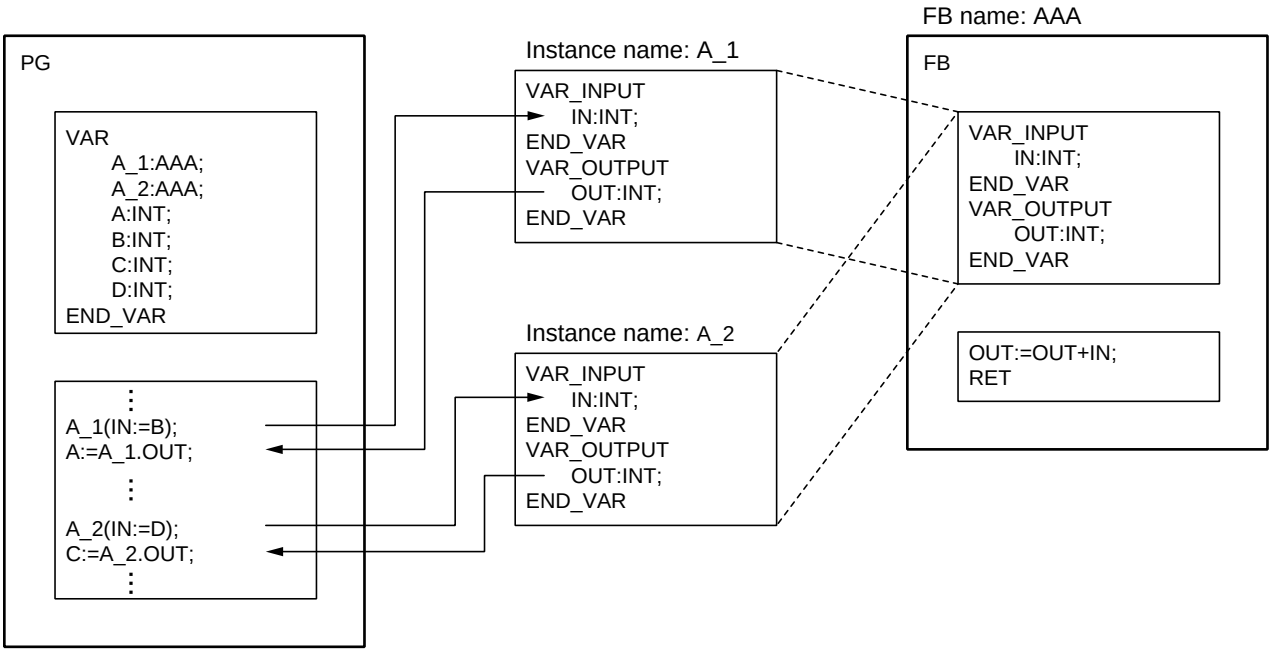
The program that calls a function block can access only the input and output variables of the function block; it can access none of the internal variables of the function block.

Predefined function blocks can be used within a program or function block.

The instances of a function block can be referenced only within the program organization unit in which they are declared unless the instances are declared globally.

<Uses of input/output variables of a function block>

Use	Within a Function Block	Outside a Function Block
Read an input variable	○	x
Write an input variable	○	○
Read an output variable	○	○
Write an output variable	○	x



Note: 1) OUT retains the old value when it is called the next time.
2) Instances of the same FB do not affect each other if they are of different types.

3) Program

A program is the basic program organization unit for a user application. Only programs can be assigned to tasks (neither functions nor function blocks can be assigned to a task). A program has neither input variables nor output variables.

1-8 Calendar Function

(These functions are not supported by SPS.)

The MICREX-SX series CPU modules incorporate a clock that provides calendar functions. The values of the calendar can be monitored and set from the D300win. They can also be monitored and set from an application program.

1-8-1 Calendar's value range

The calendar can measure calendar values from January 1st, 00:00:00, 1970 through December 31st, 23:59:59, 2069.

Note: One second after December 31, 2069 23:59:59, the date and time will turn back to January 1, 1970.

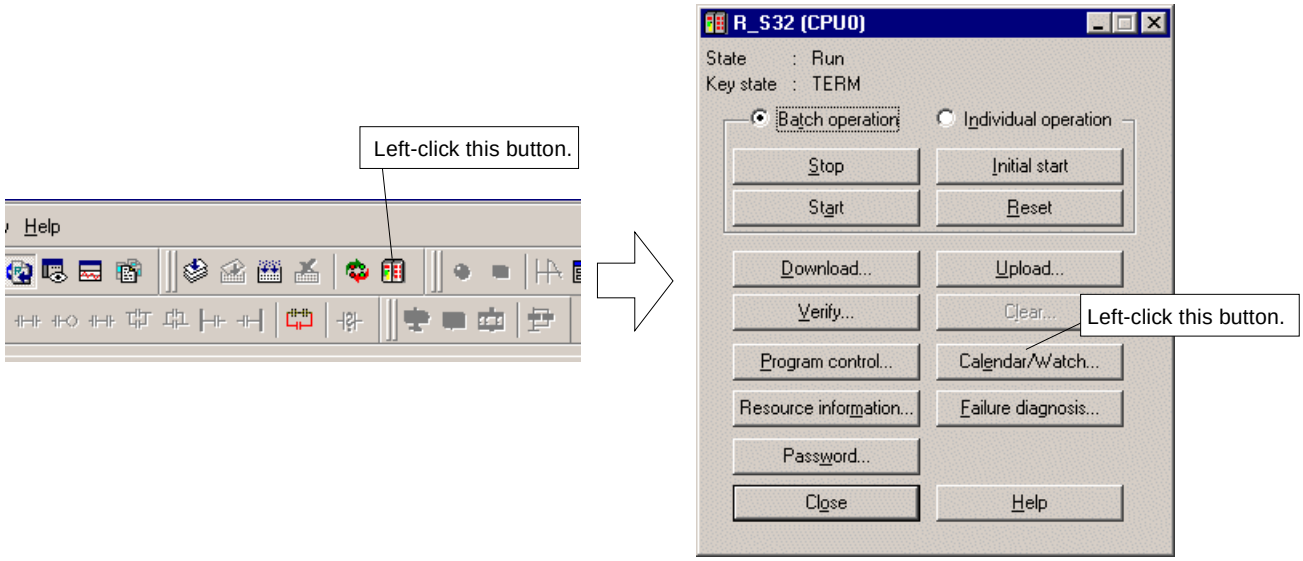
1-8-2 Calendar accuracy

The accuracy of the calendar (clock) in the CPU is 27 seconds per month (at an ambient temperature of 25°C).

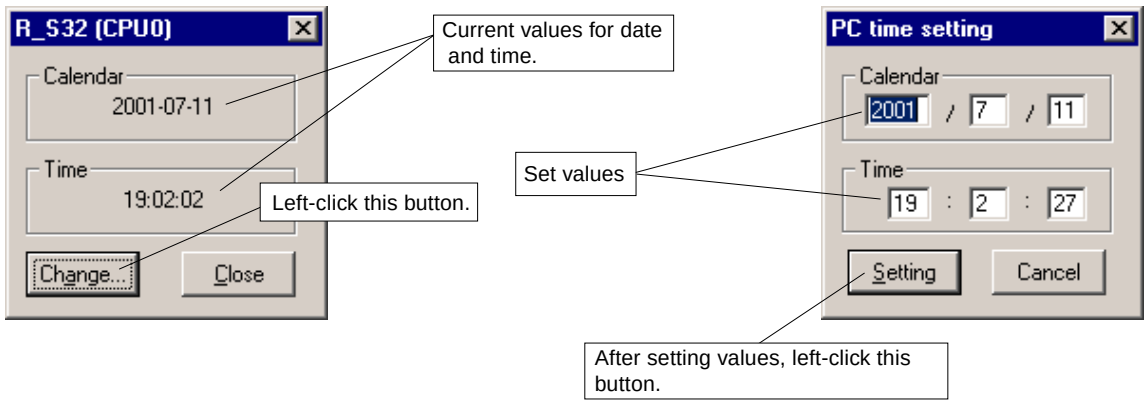
Note: The accuracy of the calendar clock varies depending on environmental conditions such as ambient temperature. When the CPU module is to be used in a system where a high calendar accuracy is required, measure the actual calendar accuracy and review the inspection (clock calibration) period.

1-8-3 Monitoring and setting up the calendar clock from the D300win

1) Left-click the [Control] Button in the menu bar.
The "CONTROL" dialog box will open.



2) Left-click the [Calendar/Watch...] Button, and the current value will be displayed.



3) Left-click the [Change ...] button, and the "PC time setting" dialog box opened. On this dialog box, you can set any desired value.

1-8-4 Monitoring and setting up the calendar clock from an application program

1) HW_RTC (Hardware RTC) - Original FB

(These functions are not supported by SPS.)

Use the hardware RTC function block (HW_RTC) to monitor and set up the calendar clock from an application program. For detailed instructions, refer to the manual for HW_RTC.

Note) DT type (date and time type) data should be specified for HW_RTC. The range of DT data is January 1, 1970 00:00:00 to February 7, 2160 6:18:15. The data allowed for HW_RTC ranges from January 1, 1970 00:00:00 to December 31 23:59:59. (If any date and time is specified, an expected value will be set.)

When setting up the calendar clock using data supplied from an external device, it is necessary to convert the input data to the DT type. The DT type data is equivalent to a 32-bit unsigned integer in seconds that starts at January 1st, 00:00:00, 1970.

(Examples:)

(1) January 1st, 12:34:54, 1970 to DINT#45296 WORD#16#0000B0F0
(2) January 1st, 00:00:00, 1998 to DINT#883612800 WORD#16#34AADC80

2) RTC (real-time clock) - IEC standard FB

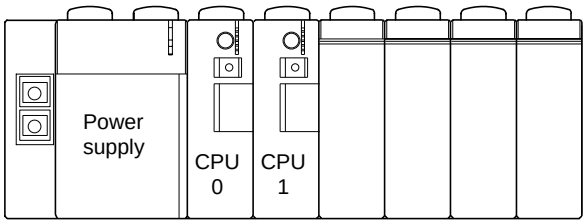
(These functions are not supported by SPS.)

The IEC standard function block RTC cannot be used to set up the calendar clock. If RTC is used to set up the calendar clock, a relative value is stored in an instance area on which the calendar clock will run.

1-8-5 Time adjustment function

(These functions are not supported by SPS.)

In a multi-CPU system the MICREX-SX series CPU modules provide a function that automatically adjusts the time of their internal clock (real-time clock).



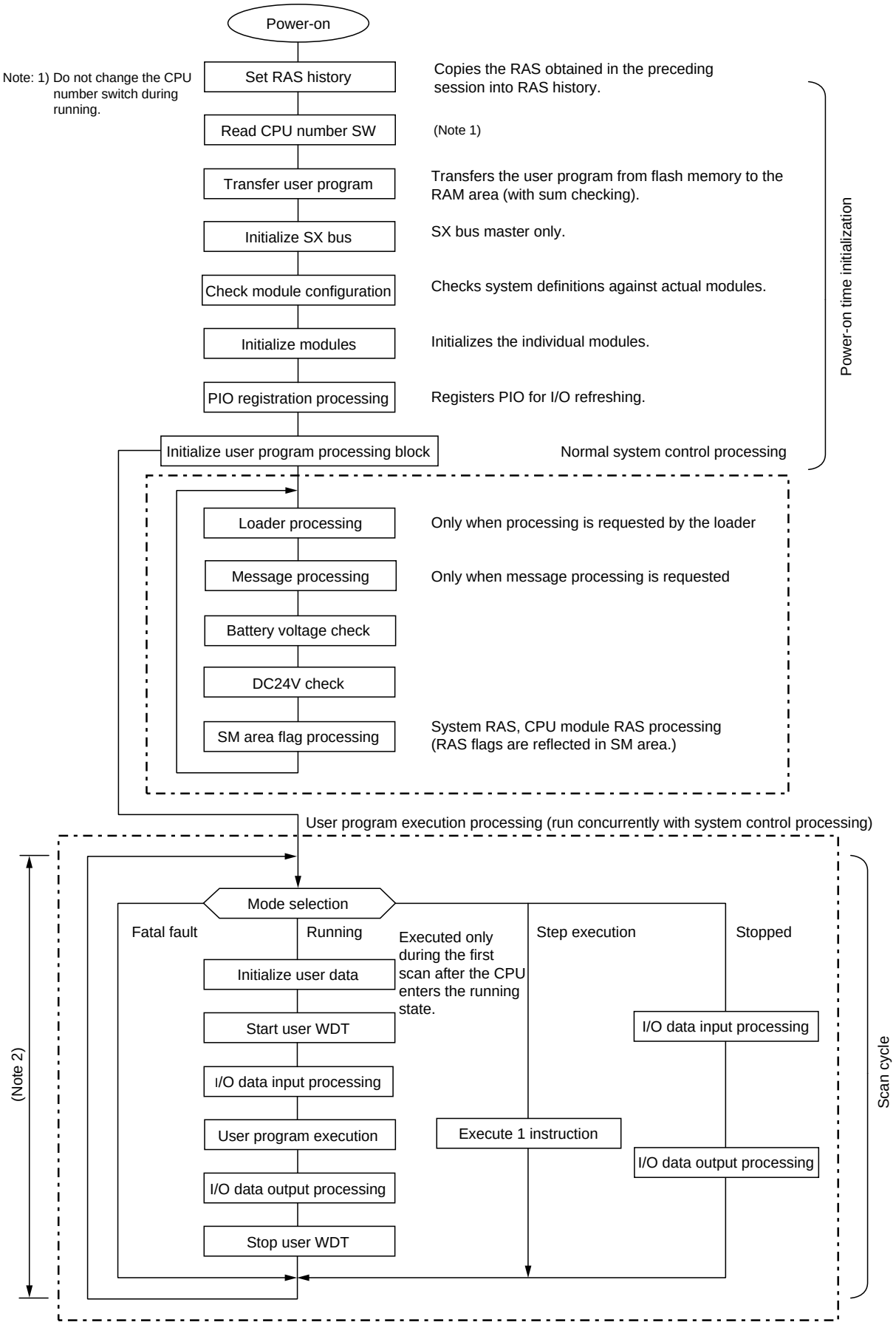
1) Time adjustment management

The number “0”CPU module adjusts the real-time clock of the other CPU modules. If the CPU0 goes down, another CPU module is assigned to adjust the real-time clock of the other CPU modules.

2) Timing of time adjustment

- When the system is powered on. Subsequently, it is adjusted at predetermined intervals (every minute).
- When the real-time clock is updated from the D300win or an application program.

The operating flowchart given below shows the power-on sequence of the MICREX-SX series SPH system and the subsequent operation sequence.



Note: 2) The execution time of the default task specified by the programming loader refers to the time in this section.
3) Sometimes it takes time to finish message related instruction when mode transition has occured.

Section 2 Programming Languages

Page

Section 2 Programming Languages

2-1 Types of Programming Languages

The MICREX-SX series CPU modules support five programs languages that are defined in IEC 61131-3.

These programming languages are classified under two text languages and three graphics languages as shown below.

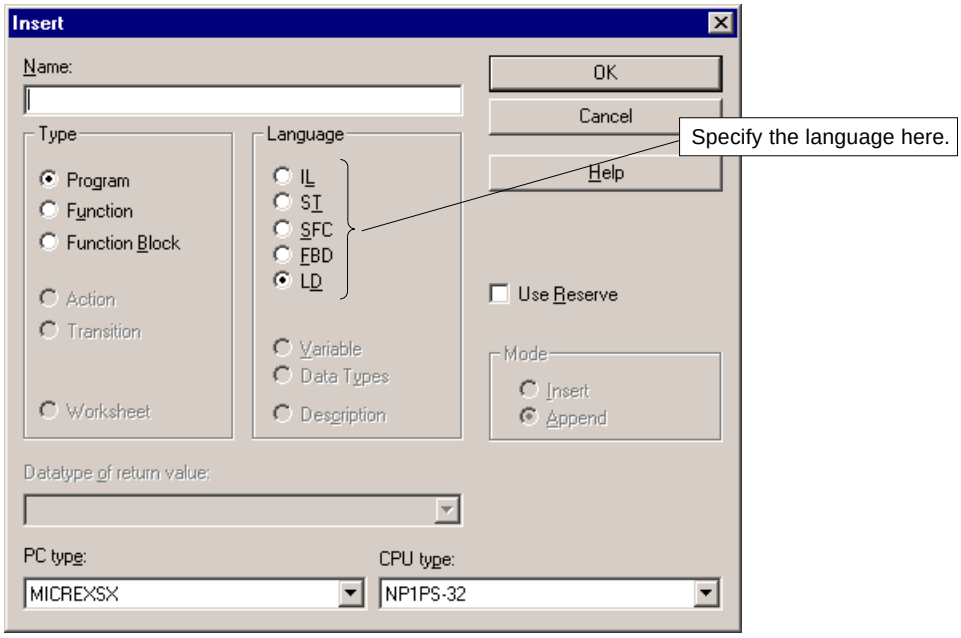
- **Text languages**
  - **IL language (Instruction List)**
  - **ST language (Structured Text)**
- **Graphic languages**
  - **LD language (Ladder Diagram)**
  - **FBD language (Function Block Diagram)**
  - **SFC elements (Sequential Function Chart)**

Graphic languages can be used for programming in combination with any of the above languages on one worksheet. The available combinations of languages are listed in the table below.

Language Specified on Sheet (Note 1)	**Available Combination**
LD language	LD language, FBD language
FBD language	LD language, FBD language
SFC elements	LD language, FBD language, SFC elements (Note 2)

Note: 1) The Programming languages are specified by the D300win loader in the “Insert” dialog box of the POU.

<Insert dialog box>



Note: 2) In actions and transitions of SFC elements, text languages (IL and ST) are available.

3) When the transfer error occurs in each instruction operation, CPU performs as follows (only for SPH):

1) When a transfer error occurs while reading data from the global memory via the processor bus, ENO is set to 0 (*1) and the operation results are indefinite (*2).

*1: ENO: Is an execution flag in the CPU that is set to “0” when an error occurs.

*2: Indefinite data refers to a value that is defined by the individual CPU hardware (example: “0” or the operation results executed at that time).

2) When a transfer error occurs while writing the data into the global memory via the processor bus (*3), ENO is set to 0 and the data storing may be indefinite.

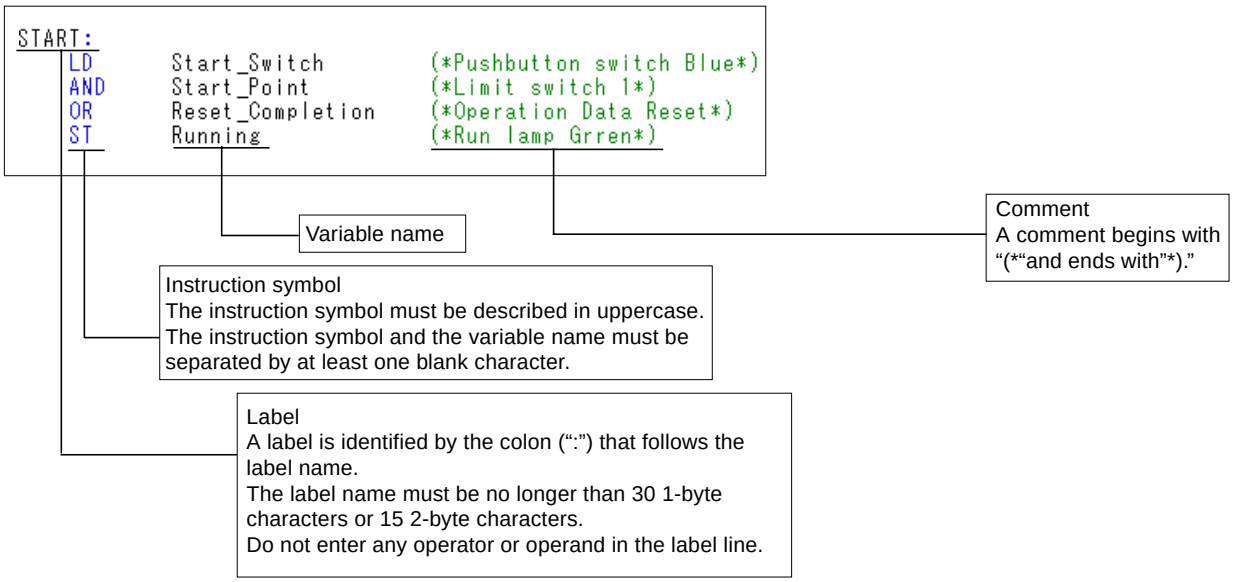
*3: The value in the destination area remains the same unless the write is carried out.

Note: 4) The number of steps in the instruction summary table indicates the minimum number of steps in the intermediate language. One step of the intermediate language may be from 1 to 4 words in size depending on the type of data or operands.

5) EN (enable input) and ENO (enable output) described in the detailed explanations of instructions are the functions for controlling execution. Note that these functions are not supported in versions earlier than D300winV2.0 and software PLC (SPS).

An instruction in the IL language consists of an instruction symbol, a variable name, and a comment specified on a single line. As shown the first line in the figure below, the instruction symbol of an instruction may be preceded by a label. The label serves as the destination of jump instructions.

<Examples of IL language instructions>



2-2-1 IL instruction summary

The IL instructions can be used only in the IL language.

Instruction	Name	Data Type	Description	No. of steps	Page
LD	Load	ANY	Loads the operand value and sets it to the operation result.	1	P2-6
LDN	Load not	ANY_BIT	Loads the inverted value of the operand and sets it to the operation result.	1	P2-6
ST	Store	ANY	Stores the current operation result in the operand.	1	P2-7
STN	Store not	ANY_BIT	Stores the inverted value of the current operation result in the operand.	1	P2-7
S	Set	BOOL	Sets the operand to 1 if the current operation result is 1.	1	P2-8
R	Reset	BOOL	Resets the operand to 0 if the current operation result is 1.	1	P2-8
AND	Logical product	ANY_BIT	Operates the logical product of the current operation result and the operand value, and sets the result to the operation result.	1	P2-9
AND(			Operates the logical product of the current operation result and the operation result specified in parentheses, and sets the result to the operation result.	1	P2-9
ANDN	Logical inverted product	ANY_BIT	Operates the logical product of the current operation result and the inverted value of the operand value, and sets the result to the operation result.	1	P2-10
ANDN(			Operates the logical product of the current operation result and the inverted value of the operation value specified in parentheses, and sets the result to the operation result.	1	P2-10
OR	Logical add	ANY_BIT	Operates the logical add of the current operation result and the operand value, and sets the result to the operation result.	1	P2-11
OR(			Operates the logical add of the current operation result and the operation result specified in parentheses, and sets the result to the operation result.	1	P2-11

Instruction	Name	Data Type	Description	No. of steps	Page
ORN	Logical inverted add	ANY_BIT	Operates the logical add of the current operation result and the inverted value of the operand value, and sets the result to the operation result.	1	P2-12
ORN(			Operates the logical add of the current operation result and the inverted value of the operation result specified in parentheses, and sets the result to the operation result.	1	P2-12
XOR	Exclusive OR	ANY_BIT	Operates the exclusive OR of the current operation result and the operand value, and sets the result to the operation result.	1	P2-13
XOR(			Operates the exclusive OR of the current operation result and the operation result specified in parentheses, and sets the result to the operation result.	1	P2-13
XORN	Exclusive NOR	ANY_BIT	Operates the exclusive NOR of the current operation result and the inverted value of operand value, and sets the result to the operation result.	1	P2-14
XORN(			Operates the exclusive NOR of the current operation result and the inverted value of the operation result specified in parentheses, and sets the result to the operation result.	1	P2-14
ADD	Addition	ANY_NUM	Operates the addition of the current operation result and the operand value, and sets the result to the operation result.	1	P2-15
ADD(			Operates the addition of the current operation result and the operation result specified in parentheses, and sets the result to the operation result.	1	P2-15
SUB	Subtraction	ANY_NUM	Operates the subtraction of the operand value from the current operation result, and sets the result to the operation result.	1	P2-16
SUB(			Operates the subtraction of the operation result specified in parentheses from the current operation result, and sets the result to the operation result.	1	P2-16
MUL	Multiplication	ANY_NUM	Operates the multiplication of the current operation result and the operand value, and sets the result to the operation result.	1	P2-17
MUL(			Operates the multiplication of the current operation result and the operation result specified in parentheses, and sets the result to the operation result.	1	P2-17
DIV	Division	ANY_NUM	Operates the division of the current operation result by the operand value, and sets the result to the operation result.	1	P2-18
DIV(			Operates the division of the current operation result by the operation result specified in parentheses, and sets the result to the operation result.	1	P2-18
GT	Comparison (>)	elementary (excluding STRING) (Note)	Operates the comparison of the current operation result as the left-hand side of the ">" operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-19
GT(			Operates the comparison of the current operation result as the left-hand side of the ">" operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-19
GE	Comparison (≥)	elementary (excluding STRING) (Note)	Operates the comparison of the current operation result as the left-hand side of the "≥ " operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-20
GE(			Operates the comparison of the current operation result as the left-hand side of the "≥ "operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-20
EQ	Comparison (=)	elementary (excluding STRING) (Note)	Operates the comparison of the current operation result as the left-hand side of the "=" operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-21
EQ(			Operates the comparison of the current operation result as the left-hand side of the "=" operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-21
NE	Comparison (≠)	elementary (excluding STRING) (Note)	Operates the comparison of the current operation result as the left-hand side of the "≠ " operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-22
NE(			Operates the comparison of the current operation result as the left-hand side of the "≠ " operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-22

Note: When SPS is used, neither STRING type nor ANY_DATE type can be used. BOOL type cannot be used for GT, GT(, GE or GE(, either.

Instruction	Name	Data Type	Description	No. of steps	Page
LE	Comparison (≤)	elementary (excluding STRING) (Note 1)	Operates the comparison of the current operation result as the left-hand side of the "≤ " operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-23
LE(			Operates the comparison of the current operation result as the left-hand side of the "≤ " operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-23
LT	Comparison (<)	elementary (excluding STRING) (Note 1)	Operates the comparison of the current operation result as the left-hand side of the "<" operator with the operand value as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-24
LT(			Operates the comparison of the current operation result as the left-hand side of the "<" operator with the operation result specified in parentheses as the right-hand side, and sets the result (BOOL value) to the operation result.	1	P2-24
JMP	Unconditional jump	-	Jumps to the step specified by the label of the operand.	1	P2-25
JMPC	TRUE conditional jump	-	Jumps to the step specified by the label of the operand when the current operation result is TRUE (nonzero).	1	P2-25
JMPCN	FALSE conditional jump	-	Jumps to the step specified by the label of the operand when the current operation result is FALSE (zero).	1	P2-26
Label_name	Label of the destination address	-	Indicates the destination address. (Note 2)	0	P2-3
CAL	Unconditional call	-	Calls the function block specified in the operand.	1	P2-26
CALC	TRUE conditional call	-	Calls the function block specified in the operand if the current operation result is TRUE (nonzero).	1	P2-27
CALCN	FALSE conditional call	-	Calls the function block specified in the operand if the current operation result is FALSE (zero).	1	P2-27
RET	Unconditional return	-	Returns to the calling program.	1	P2-28
RETC	TRUE conditional return	-	Returns to the calling program if the current operation result is TRUE (nonzero).	1	P2-28
RETCN	FALSE conditional return	-	Returns to the calling program if the current operation result is FALSE (zero).	1	P2-29
)	Closing Parenthesis	-	Performs the deferred operation.	1	-

Note: 1) When SPS is used, none of BOOL type, STRING type and ANY_DATE type can be used.
2) When SPS is used, only LD, CAL, JMP or RET instruction can be used just after a label.

Key-point:

The number of steps depends on the operand used. The number of steps for a variable that holds an array may be large.

2-2-2 IL language instructions

(1) Load LD

Name, Symbol, Function		Example
Name	Load	<Programming example> Loads the value specified in “INPUT1” and sets it to the operation result (PC internal register). LD INPUT1 ⋮ <Operation> • When the data type of INPUT1 is BOOL type INPUT1 1 ⇩ Operation result 1 • When the data type of INPUT1 is INT type INPUT1 1 2 3 4 ⇩ Operation result 1 2 3 4 • When the data type of INPUT1 is WORD type INPUT1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 ⇩ Operation result 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0
Symbol	LD	
Function	(1) Starts the calculations in a series of processes. Loads the value specified in the operand and sets it to the operation result. (2) Available data types are all data types (ANY type) of MICREX-SX series. (3) When SPS is used, ANY_DATE type (DATE type, DT type, TOD type) cannot be used.	

(2) Load not LDN

Name, Symbol, Function		Example
Name	Load not	<Programming example> • Loads the inverted value of “INPUT1” and sets it to the operation result. LDN INPUT1 ⋮ <Operation> • When the data type of INPUT1 is WORD type INPUT1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 1 ⇩ Operation result 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0
Symbol	LDN	
Function	(1) Starts the calculations in a series of processes. LDN loads the inverted value of the operand and sets it to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

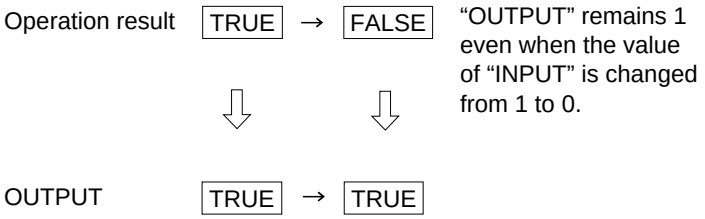
(3) Store ST

Name, Symbol, Function		Example
Name	Store	<Programming example> Stores the current operation result (before the ST instruction) in “OUTPUT.” ⋮ ST OUTPUT <Operation> • When the data type of the operation result is BOOL type Operation result 1 ⇩ OUTPUT 1 • When the data type of the operation result is INT type Operation result 1 2 3 4 ⇩ OUTPUT 1 2 3 4 • When the data type of the operation result is WORD type Operation result 1001 1011 0111 0011 ⇩ OUTPUT 1001 1011 0111 0011
Symbol	ST	
Function	(1) Stores the current operation result into the address specified in the operand. (2) Available data types are all data types (ANY type) of MICREX-SX series. (3) The data type of the operand must be the same as that of the operation result. (4) When SPS is used, ANY_DATE type (DATE type, DT type, TOD type) cannot be used.	

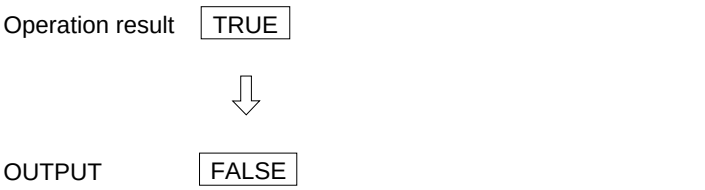
(4) Store not STN

Name, Symbol, Function		Example
Name	Store not	<Programming example> Stores the inverted boolean value of the current operation result (before the STN instruction) in “OUTPUT.” ⋮ STN OUTPUT <Operation> • When the data type of the operation result is WORD type Operation result 0111 0101 0101 0111 ⇩ INPUT2 1000 1010 1010 1000
Symbol	STN	
Function	(1) Stores the inverted value of the current operation result in the address specified in the operand. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type). (3) The data type of the operand must be the same as that of the operation result.	

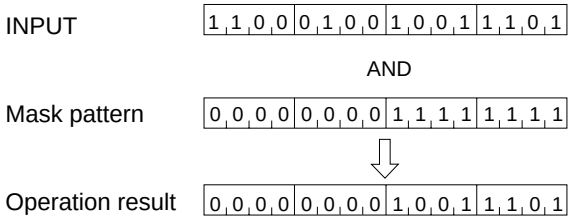
(5) Set S

Name, Symbol, Function		Example
Name	Set	<Programming example> Sets the value of “OUTPUT” to “TRUE” if the value of “INPUT” is “TRUE.” LD INPUT S OUTPUT <Operation> 
Symbol	S	
Function	(1) Sets the value of the address specified in the operand to “TRUE” if the current operation result is “TRUE.” The value in the address specified in the operand remains unchanged if the current operation result is “FALSE.” (2) Available data type is BOOL type.	

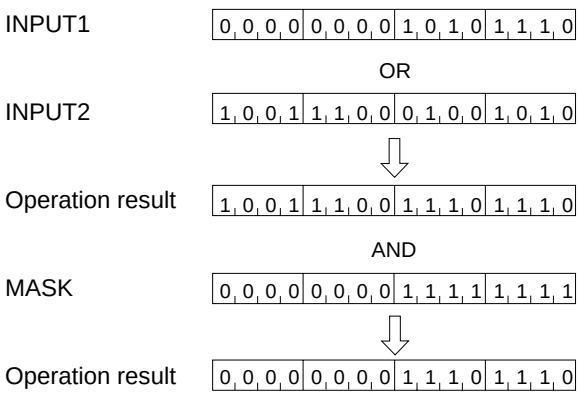
(6) Reset R

Name, Symbol, Function		Example
Name	Reset	<Programming example> Resets the value of “OUTPUT” which is set by “INPUT1” if the value of “INPUT2” is 1. LD INPUT1 S OUTPUT ⋮ LD INPUT2 R OUTPUT <Operation> 
Symbol	R	
Function	(1) Resets the value of the address specified in the operand to “FALSE” if the current operation result is “TRUE.” The value in the address specified in the operand remains unchanged if the current operation result is “FALSE.” (2) Available data type is BOOL type.	

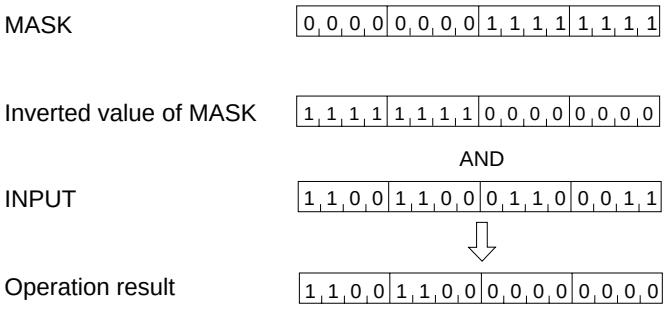
(7) Logical product AND

Name, Symbol, Function		Example
Name	Logical product	<Programming example> Masks the highest-order 2 digits of “INPUT” and sets the result to the operation result. LD INPUT AND WORD#16#00FF ⋮ <Operation> 
Symbol	AND	
Function	(1) Operates the logical product of the current operation result and the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

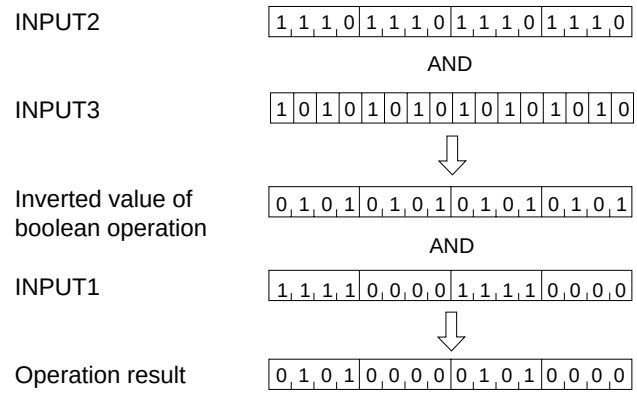
(8) Logical product AND(

Name, Symbol, Function		Example
Name	Logical product	<Programming example> Operates the logical product of the operation result, obtained by the logical add of “INPUT1” and “INPUT2,” and the “MASK” data, and sets the result to the operation result. LD MASK AND(INPUT1 OR INPUT2) ⋮ <Operation> 
Symbol	AND(	
Function	(1) Operates the logical product of the current operation result and the operation result specified in parentheses, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

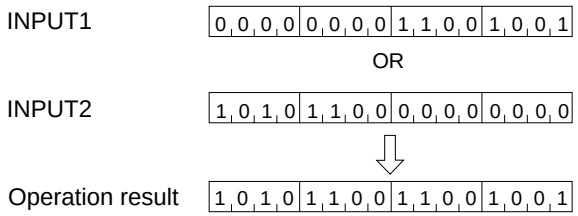
(9) Logical inverted product ANDN

Name, Symbol, Function		Example
Name	Logical inverted product	<Programming example> Operates the logical product of the "INPUT" data and the inverted value of the "MASK," and sets the result to the operation result. LD INPUT ANDN MASK ⋮
Symbol	ANDN	
Function	(1) Operates the logical product of the current operation result and the inverted value of the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	<Operation> 

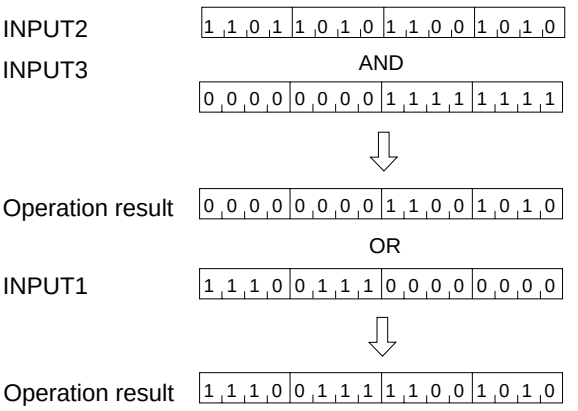
(10) Logical inverted product ANDN(

Name, Symbol, Function		Example
Name	Logical inverted product	<Programming example> Operates the logical product of the "INPUT1" and the inverted value of the operation value obtained by the logical product of "INPUT2" and "INPUT3," and sets the result to the operation result. LD INPUT1 ANDN(INPUT2 AND INPUT3) ⋮
Symbol	ANDN(	
Function	(1) Operates the logical product of the current operation result and the inverted value of the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	<Operation> 

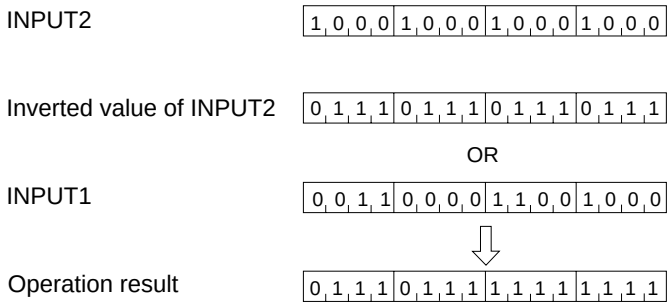
(11) Logical add OR

Name, Symbol, Function		Example
Name	Logical add	<Programming example> Operates the logical add of the “INPUT1” data and the “INPUT2” data, and sets the result to the operation result. LD INPUT1 OR INPUT2 ⋮
Symbol	OR	
Function	(1) Operates the logical add of the current operation result and the value specified in the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	<Operation> 

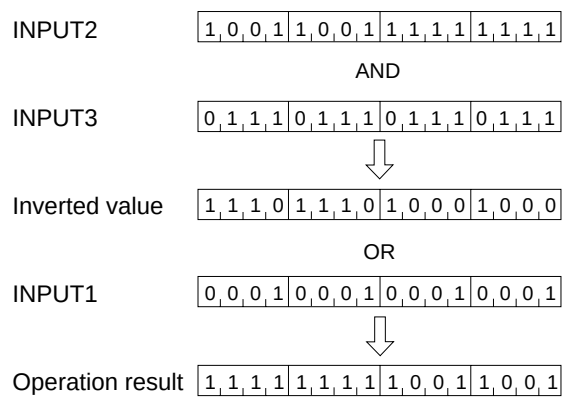
(12) Logical add OR(

Name, Symbol, Function		Example
Name	Logical add	<Programming example> Operates the logical add of the “INPUT1” data and the operation result obtained by the logical product of the “INPUT2” data and the “INPUT3” data, and sets the result to the operation result. LD INPUT1 OR(INPUT2 AND INPUT3) ⋮
Symbol	OR(	
Function	(1) Operates the logical add of the current operation result and the operation result specified in parentheses, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	<Operation> 

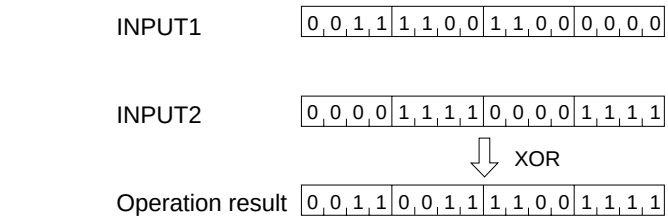
(13) Logical inverted add ORN

Name, Symbol, Function		Example
Name	Logical inverted add	<Programming example> Operates the logical add of the “INPUT1” data and the inverted value of the “INPUT2” data, and sets the result to the operation result. LD INPUT1 ORN INPUT2 ⋮ <Operation> 
Symbol	ORN	
Function	(1) Operates the logical add of the current operation result and the inverted value of the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

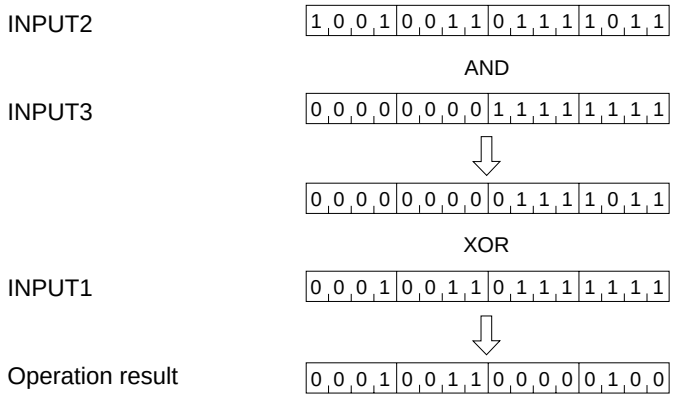
(14) Logical inverted add ORN(

Name, Symbol, Function		Example
Name	Logical inverted add	<Programming example> Operates the logical add of the “INPUT1” data and the inverted value of the operation result obtained by the logical product of the “INPUT2” data and the “INPUT3” data, and sets the result to the operation result. LD INPUT1 ORN(INPUT2 AND INPUT3) ⋮ <Operation> 
Symbol	ORN(	
Function	(1) Operates the logical add of the current operation result and the inverted value of the operation result specified in parentheses, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

(15) Exclusive OR XOR

Name, Symbol, Function		Example
Name	Exclusive OR	<Programming example> Operates the exclusive OR of the “INPUT1” data and the “INPUT2” data, and sets the result to the operation result. LD INPUT1 XOR INPUT2 ⋮ <Operation> 
Symbol	XOR	
Function	(1) Operates the exclusive OR of the current operation result and the operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

(16) Exclusive OR XOR(

Name, Symbol, Function		Example
Name	Exclusive OR	<Programming example> Operates the exclusive OR of the “INPUT1” data and the operation result obtained by the logical product of the “INPUT2” data and the “INPUT3” data, and sets the result to the operation result. LD INPUT1 XOR(INPUT2 AND INPUT3) ⋮ <Operation> 
Symbol	XOR(	
Function	(1) Operates the exclusive OR of the current operation result and the operation result specified in parentheses, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

(17) Exclusive NOR XORN

Name, Symbol, Function		Example
Name	Exclusive NOR	<Programming example> Operates the exclusive NOR of the “INPUT1” data and the inverted value of the “INPUT2” data, and sets the result to the operation result. LD INPUT1 XORN INPUT2 ⋮ <Operation> INPUT2 0 0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 Inverted value of INPUT2 1 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 XOR INPUT1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 ⇩ Operation result 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1
Symbol	XORN	
Function	(1) Operates the exclusive NOR of the current operation result and the inverted value of operand value, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

(18) Exclusive NOR XORN(

Name, Symbol, Function		Example
Name	Exclusive NOR	<Programming example> Operates the exclusive NOR of the “INPUT1” data and the inverted value of the operation result obtained by the logical product of the “INPUT2” data and the “INPUT3” data, and sets the result to the operation result. LD INPUT1 XORN(INPUT2 AND INPUT3) ⋮ <Operation> INPUT2 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 AND INPUT3 1 0 1 0 1 0 1 0 0 0 1 1 0 0 1 1 ⇩ Inverted value 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 XOR INPUT1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 ⇩ Operation result 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 0
Symbol	XORN(	
Function	(1) Operates the exclusive NOR of the current operation result and the inverted value of the operation result specified in parentheses, and sets the result to the operation result. (2) Available data type is ANY_BIT type (BOOL type, WORD type, or DWORD type).	

(19) Addition ADD

Name, Symbol, Function		Example
Name	Addition	<Programming example> Operates the addition of the current operation result and the "ADATA" data, and sets the result to the operation result. ⋮ ADD ADATA ⋮
Symbol	ADD	
Function	(1) Operates the addition of the current operation result and the operand value, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (4) ENO is set to 0 if the result of operation exceeds the valid value range of the operands (only for SPH). Note: 1) The CPU performs no boundary processing when the operation result of operand types other than REAL exceeds the valid value range of the operands. Make sure that the operation result does not exceeds the valid value range of the operands. The CPU performs boundary processing for REAL operands.	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result + ADATA → Operation → Operation result 1234 + 5678 → 6912 EN= 1 ENO= 1 • When the result of operation exceeds the valid value range 32767 + 32767 → Operation → -2 EN= 1 ENO= 0 -32768 + -32768 → Operation → 0 EN= 1 ENO= 0 Note: 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of all the data types used.

(20) Addition ADD(

Name, Symbol, Function		Example
Name	Addition	<Programming example> Operates the addition of the current operation result and the operation result obtained by the addition of "INPUT1" and "INPUT2", and sets the result to the operation result. ⋮ ADD(INPUT1 ADD INPUT2) ⋮
Symbol	ADD(	
Function	(1) Operates the addition of the current operation result and the operation result specified in parentheses, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (4) ENO is set to 0 if the result of operation exceeds the valid value range of the operands (only for SPH). Note: 1) The CPU performs no boundary processing when the operation result of operand types other than REAL exceeds the valid value range of the operands. Make sure that the operation result does not exceeds the valid value range of the operands. The CPU performs boundary processing for REAL operands.	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result + Result of operation specified in parentheses → Operation → Operation result 1234 + 5678 → 6912 EN=1 ENO=1 • When the result of execution exceeds the valid value range 32767 + 32767 → Operation → -2 EN=1 ENO=0 -32768 + -32768 → Operation → 0 EN=1 ENO=0 Note: 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of all the data types used.

(21) Subtraction SUB

Name, Symbol, Function		Example
Name	Subtraction	<Programming example> Operates the subtraction of the "SDATA" value from the current operation result, and sets the result to the operation result. ⋮ SUB SDATA ⋮
Symbol	SUB	
Function	(1) Operates the subtraction of the operand value from the current operation result, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (4) ENO is set to 0 if the result of operation exceeds the valid value range of the operands (only for SPH). (5) For REAL type data, when the operation results approaches 0 (zero) and cannot be expressed by REAL type, output value and EN0 become "0" and "1," respectively (only for SPH).	<Operation> Example of INT type data • When the result of execution falls within the valid value range Current operation result: 5678 − SDATA: 1234 → Operation → Operation result: 4444 EN=1, ENO=1 • When the result of execution exceeds the valid value range -32768 − 50 → Operation → 32718 EN=1, ENO=0 Note: 1) The CPU performs no boundary processing when the operation result of operand types other than REAL exceeds the valid value range of the operands. Make sure that the operation result does not exceeds the valid value range of the operands. The CPU performs boundary processing for REAL operands. 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of all the data types used.

(22) Subtraction SUB(

Name, Symbol, Function		Example
Name	Subtraction	<Programming example> Operates the subtraction of the sum of "INPUT1" and "INPUT2" from the current operation result, and sets the result to the operation result. ⋮ SUB(INPUT1 ADD INPUT2) ⋮
Symbol	SUB(	
Function	(1) Operates the subtraction of the operation result specified in parentheses from the current operation result, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (4) ENO is set to 0 if the result of operation exceeds the valid value range of the operands (only for SPH). (5) For REAL type data, when the operation results approaches 0 (zero) and cannot be expressed by REAL type, output value and EN0 become "0" and "1," respectively (only for SPH).	<Operation> Example of INT type data • When the result of execution falls within the valid value range Current operation result: 5678 − Result of operation specified in parentheses: 1234 → Operation → Operation result: 4444 EN=1, ENO=1 • When the result of execution exceeds the valid value range -32768 − 50 → Operation → 32718 EN=1, ENO=0 Note: 1) The CPU performs no boundary processing when the operation result of operand types other than REAL exceeds the valid value range of the operands. Make sure that the operation result does not exceeds the valid value range of the operands. The CPU performs boundary processing for REAL operands. 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of all the data types used.

(23) Multiplication MUL

Name, Symbol, Function		Example
Name	Multiplication	<Programming example> Operates the multiplication of the current operation result and the "MDATA" data, and sets the result to the operation result. ⋮ MUL MDATA ⋮
Symbol	MUL	
Function	(1) Operates the multiplication of the current operation result and the operand value, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) When the operation result exceeds the boundary value of the data type, it is treated as follows: SPH: The boundary value of the data type is regarded as the operation result, and EN0 is set to "0." SPS: No boundary processing is performed. (4) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (5) For REAL type data, when the operation results approaches 0 (zero) and cannot be expressed by REAL type, output value and EN0 become "0" and "1," respectively (only for SPH).	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result: 222 x MDATA: 10 → Operation result: 2220 (Operation) EN=1 ENO=1 • When the result of operation exceeds the valid value range 32767 x 32767 → 32767 (Operation) EN=1 ENO=0 -32768 x 32767 → -32768 (Operation) EN=1 ENO=0 Note: When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of the data type.

(24) Multiplication MUL(

Name, Symbol, Function		Example
Name	Multiplication	<Programming example> Operates the multiplication of the current operation result and the sum of the "INPUT1" and "INPUT2," and sets the result to the operation result. ⋮ MUL(INPUT1 ADD INPUT2) ⋮
Symbol	MUL(	
Function	(1) Operates the multiplication of the current operation result and the operation result specified in parentheses, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) When the operation result exceeds the boundary value of the data type, it is treated as follows: SPH: The boundary value of the data type is regarded as the operation result, and EN0 is set to "0." SPS: No boundary processing is performed. (4) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (5) For REAL type data, when the operation results approaches 0 (zero) and cannot be expressed by REAL type, output value and EN0 become "0" and "1," respectively (only for SPH).	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result: 222 x Result of operation specified in parentheses: 10 → Operation result: 2220 (Operation) EN=1 ENO=1 • When the result of operation exceeds the valid value range 32767 x 32767 → 32767 (Operation) EN=1 ENO=0 -32768 x 32767 → -32768 (Operation) EN=1 ENO=0 Note: When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of the data type.

(25) Division DIV

Name, Symbol, Function		Example
Name	Division	<Programming example> Operates the division of the current operation result by the "DDATA" data, and sets the result to the operation result. ⋮ DIV DDATA ⋮
Symbol	DIV	
Function	(1) Operates the division of the current operation result by the operand value, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) When the operation result exceeds the boundary value of the data type, it is treated as follows: SPH: The boundary value of the data type is regarded as the operation result, and EN0 is set to "0." SPS: No boundary processing is performed. (4) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (5) When SPH is used, if the divisor is 0, the maximum absolute value with the sign of the dividend is generated and ENO is set to 0.	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result: 3940 ÷ DDATA: 5 ⇨ Operation result: 788 (Operation) EN= 1 ENO= 1 • When the divisor is 0 3940 ÷ 0 ⇨ 32767 (Operation) EN= 1 ENO= 0
	(6) When SPS is used, SPS stops if divisor is 0 (zero) and data type is other than REAL. If data type is REAL, "-NAN" or "INF" is output.	

(26) Division DIV(

Name, Symbol, Function		Example
Name	Division	<Programming example> Operates the division of the current operation result by the sum of the "INPUT1" value and the "INPUT2" value, and sets the result to the operation result. ⋮ DIV(INPUT1 ADD INPUT2) ⋮
Symbol	DIV(	
Function	(1) Operates the division of the current operation result by the operation result specified in parentheses, and sets the result to the operation result. (2) The data type of the operands must be the same. (3) When the operation result exceeds the boundary value of the data type, it is treated as follows: SPH: The boundary value of the data type is regarded as the operation result, and EN0 is set to "0." SPS: No boundary processing is performed. (4) Available data type is ANY_NUM type (REAL type, INT type, DINT type, UINT type or UDINT type). (5) If the divisor is 0, the maximum absolute value with the sign of the dividend is generated and ENO is set to 0. (6) When SPS is used, SPS stops if divisor is 0 (zero) and data type is other than REAL. If data type is REAL, "-NAN" or "INF" is output.	<Operation> Example of INT type data • When the result of operation falls within the valid value range Current operation result: 3940 ÷ Result of operation specified in parentheses: 5 ⇨ Operation result: 788 (Operation) EN= 1 ENO= 1 • When the divisor is 0 3940 ÷ 0 ⇨ 32767 (Operation) EN= 1 ENO= 0

(27) Comparison GT

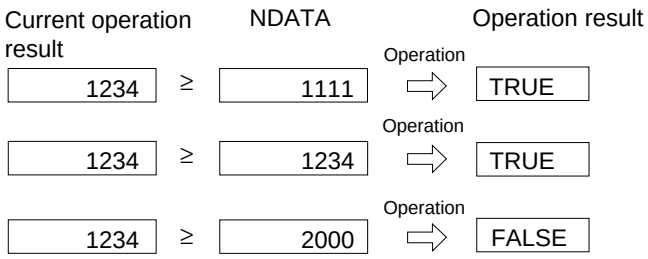
Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (>) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ GT NDATA ⋮ <Operation> Example of INT type data Current operation result / NDATA / Operation result 1234 > 1111 Operation ⇨ TRUE 1234 > 1234 Operation ⇨ FALSE 1234 > 2000 Operation ⇨ FALSE
Symbol	GT	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the ">" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither BOOL type, ANY_DATE type nor STRING type can be used.	

(28) Comparison GT(

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (>) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ GT(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data Current operation result / Result of operation specified in parentheses / Operation result 1234 > 1111 Operation ⇨ TRUE 1234 > 1234 Operation ⇨ FALSE 1234 > 2000 Operation ⇨ FALSE
Symbol	GT(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the ">" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

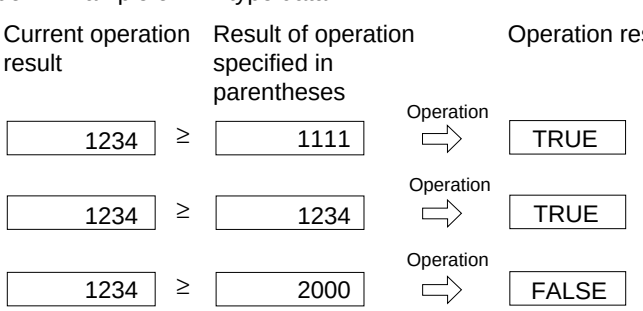
(29) Comparison GE

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≥) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ GE NDATA ⋮ <Operation> Example of INT type data
Symbol	GE	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≥" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither BOOL type, ANY_DATE type nor STRING type can be used.	

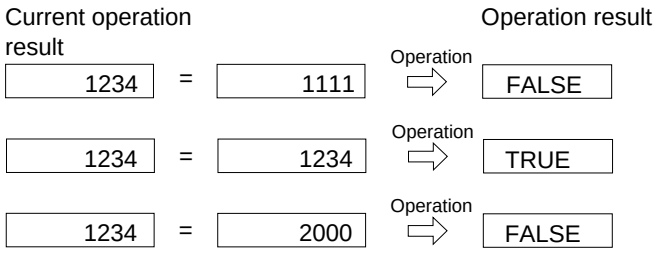


(30) Comparison GE(

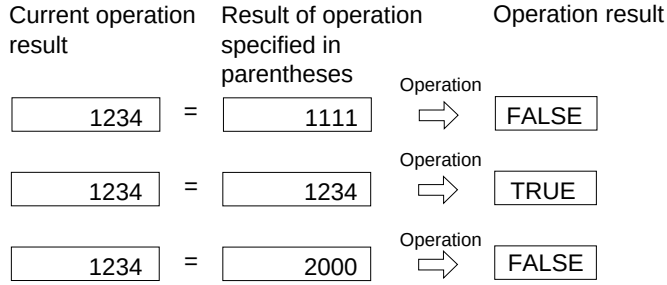
Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≥) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ GE(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data
Symbol	GE(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≥" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	



(31) Comparison EQ

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (=) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ EQ NDATA ⋮ <Operation> Example of INT type data 
Symbol	EQ	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "=" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

(32) Comparison EQ(

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (=) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ EQ(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data 
Symbol	EQ(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "=" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

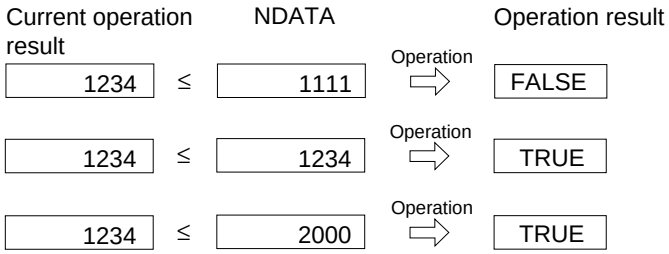
(33) Comparison NE

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≠) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ NE NDATA ⋮ <Operation> Example of INT type data Current operation result ≠ NDATA → Operation result 1234 ≠ 1111 → Operation → TRUE 1234 ≠ 1234 → Operation → FALSE 1234 ≠ 2000 → Operation → TRUE
Symbol	NE	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≠" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

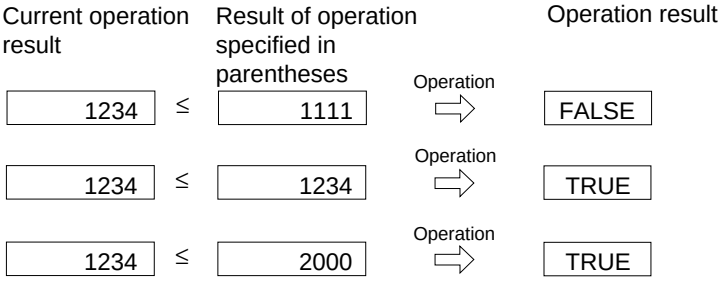
(34) Comparison NE(

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≠) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ NE(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data Current operation result ≠ Result of operation specified in parentheses → Operation result 1234 ≠ 1111 → Operation → TRUE 1234 ≠ 1234 → Operation → FALSE 1234 ≠ 2000 → Operation → TRUE
Symbol	NE(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≠" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

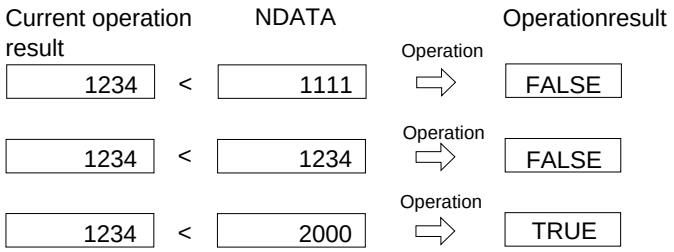
(35) Comparison LE

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≤) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ LE NDATA ⋮ <Operation> Example of INT type data 
Symbol	LE	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≤" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither BOOL type, ANY_DATE type nor STRING type can be used.	

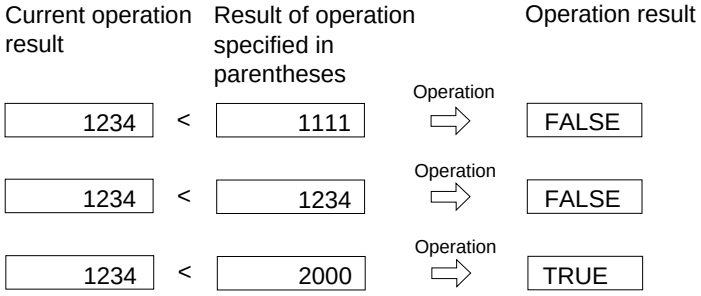
(36) Comparison LE(

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (≤) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ LE(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data 
Symbol	LE(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "≤" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

(37) Comparison LT

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (<) of the current operation result with the "NDATA" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ LT NDATA ⋮ <Operation> Example of INT type data 
Symbol	LT	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "<" operator with the operand value as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither BOOL type, ANY_DATE type nor STRING type can be used.	

(38) Comparison LT(

Name, Symbol, Function		Example
Name	Comparison	<Programming example> Operates the comparison (<) of the current operation result with the sum of the "INPUT1" value and the "INPUT2" value, and sets the result (TRUE or FALSE) to the operation result. ⋮ LT(INPUT1 ADD INPUT2) ⋮ <Operation> Example of INT type data 
Symbol	LT(	
Function	(1) Operates the comparison of the current operation result as the left-hand side of the "<" operator with the operation result specified in parentheses as the right-hand side, and sets the result to the operation result. (2) The type of the operation result is BOOL (TRUE or FALSE). (3) The data type of the operands must be the same. (4) Available data types are ANY_NUM type (REAL type, INT type, DINT type, UINT type, or UDINT type), ANY_BIT type (BOOL type, WORD type, or DWORD type), ANY_DATE type (DT type, DATE type, or TOD type), and TIME type (STRING type cannot be used). Note: When SPS is used, neither ANY_DATE type nor STRING type can be used.	

(39) Unconditional jump JMP

Name, Symbol, Function		Example
Name	Unconditional jump	<Programming example> Jumps unconditionally from “JMP END” to the instruction that is labeled “END.”
Symbol	JMP	
Function	(1) Jumps to the step specified by the label of the operand. The instructions following the JMP instruction are not executed.	

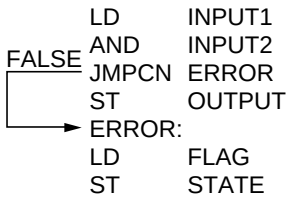
```
        Common
        processing
        block
        JMPC  ERROR_1

        Processing 1

        JMP  END

ERROR_1:  Processing 2

END:
```

(40) TRUE conditional jump JMPC

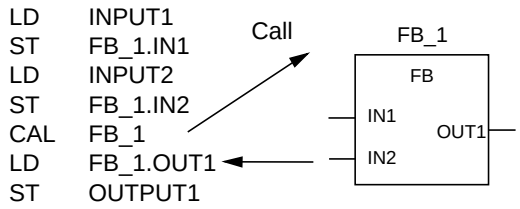
Name, Symbol, Function		Example
Name	TRUE conditional jump	<Programming example> Jumps from “JMPC ERROR” to the instruction that is labeled “ERROR” if either “INPUT1” or “INPUT2” is TRUE.
Symbol	JMPC	Note: “INPUT1” and “INPUT2” are assumed to be BOOL type.
Function	(1) Jumps to the step specified by the label of the operand when the current operation result is TRUE. The instruction following this instruction is executed if the current operation result is FALSE.	

```
        LD      INPUT1
        OR      INPUT2
TRUE    JMPC    ERROR
          ⋮

        ST      OUTPUT
        AND     XP4

        ERROR:
        LD      FLAG
        ST      STATE
```

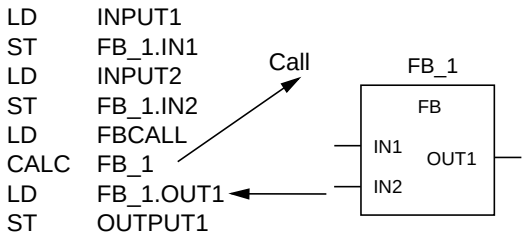
(41) FALSE conditional jump JMPCN

Name, Symbol, Function		Example
Name	FALSE conditional jump	<Programming example> Jumps from “JMPCN ERROR” to the instruction that is labeled “ERROR” if either “INPUT1” or “INPUT2” is FALSE.  Note: When SPS is used, only LD, CAL, JMP or RET instruction can be used just after a label.
Symbol	JMPCN	
Function	(1) Jumps to the step specified by the label of the operand when the current operation result is FALSE. The instruction following this instruction is executed if the current operation result is TRUE.	

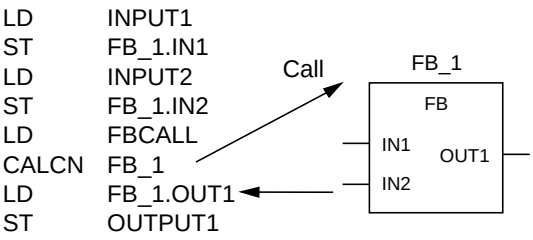
(42) Unconditional call CAL

Name, Symbol, Functioninsert figure		Example
Name	Unconditional call	<Programming example> Executing “CAL FB_1” unconditionally calls the function block “FB_1.” An instruction that passes a parameter to the input terminal is required before the CAL instruction, and an instruction that receives a parameter from the output terminal is required after the CAL instruction. 
Symbol	CAL	
Function	(1) Calls unconditionally the function block specified in the operand.	

(43) TRUE conditional call CALC

Name, Symbol, Function		Example
Name	TRUE conditional call	<Programming example> Calls the function block “FB_1” if “FBCALL” is TRUE. An instruction that passes a parameter to the input terminal is required before the CAL instruction, and an instruction that receives a parameter from the output terminal is required after the CAL instruction. 
Symbol	CALC	
Function	(1) Calls the function block specified in the operand if the current operation result is TRUE. If the current operation result is FALSE, the function block is not called and the next sequential instruction is executed.	

(44) FALSE conditional call CALCN

Name, Symbol, Function		Example
Name	FALSE conditional call	<Programming example> Calls the function block “FB_1” if “FBCALL” is FALSE. An instruction that passes a parameter to the input terminal is required before the CAL instruction, and an instruction that receives a parameter from the output terminal is required after the CAL instruction. 
Symbol	CALCN	
Function	(1) Calls the function block specified in the operand if the current operation result is FALSE. If the current operation result is TRUE, the function block is not called and the next sequential instruction is executed.	

(45) Unconditional return RET

Name, Symbol, Function		Example
Name	Unconditional return	<Programming example>
Symbol	RET	
Function	(1) Returns unconditionally to the instruction immediately following the instruction that called the current program (e.g., CAL).	

```
LD      INPUT1
AND     PSWICH
ST      IN2
  ⋮
LD      OUT1
ST      OUTPUT1
RET
```

User-supplied function block

(46) TRUE conditional return RETC

Name, Symbol, Function		Example
Name	TRUE conditional return	<Programming example>
Symbol	RETC	
Function	(1) Returns unconditionally to the instruction immediately following the instruction that called the current program (e.g., CAL) if the current operation result is TRUE. The next sequential instruction is executed if the current operation result is FALSE.	

```
LD      INPUT1
AND     PSWICH
ST      IN2
  ⋮
LD      OUT1
ST      OUTPUT1
AND     FLAG7
RETC
LD      INPUT2
  ⋮
RET
```

User-supplied function block

(47) FALSE conditional return RETCN

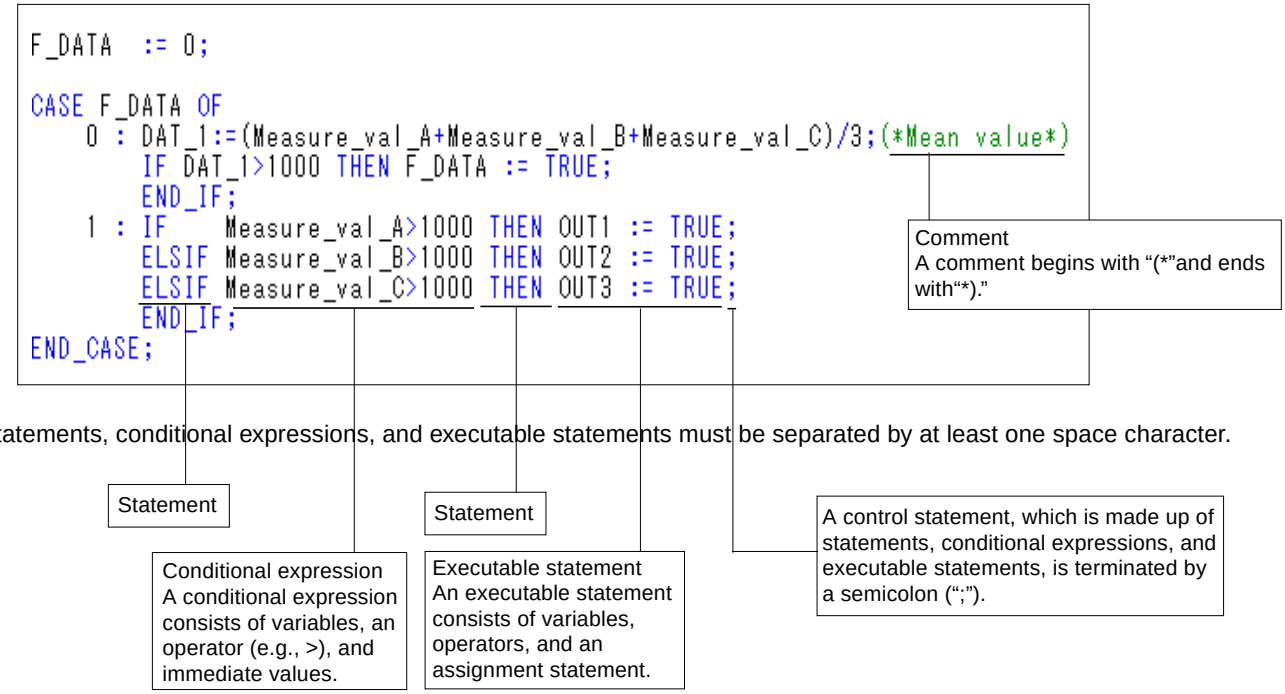
Name, Symbol, Function		Example
Name	FALSE conditional return	<Programming example>
Symbol	RETCN	
Function	(1) Returns unconditionally to the instruction immediately following the instruction that called the current program (e.g., CAL) if the current operation result is FALSE. The next sequential instruction is executed if the current operation result is TRUE.	

```
LD      INPUT1
AND     PSWICH
ST      IN2
        ⋮
LD      OUT1
ST      OUTPUT1
AND     FLAG7
RETCN
LD      INPUT2
        ⋮
RET
```

User-supplied function block

The ST language consists of operators and statements. Operators perform basic operations such as arithmetic or comparison operations. Statements specify the sequence of program execution and the flow of program control. ST language is a structured text language that is composed of a combination of these operators and statements.

<Examples of ST language>



2-3-1 ST operators

No.	Operation	Operator	Data Type	Description	Sample Value	Precedence
1	Parentheses	(expression)		(2+3)*(4+5)	45	Highest
2	Function	Function name (parameter)		LN(A) MAX(X, Y)		
3	Exponentiation	**	REAL (base, exponent)	3.0**4.0	8.1E+1	
4	Sign inversion	-	INT, DINT, REAL	-Voo1 (Voo1=10)	-10	
5	Logical not	NOT	BOOL, WORD, DWORD	Logical NOT of each list	FALSE	
6	Multiplication	*	INT, DINT, UINT, UDINT, REAL	10*3	30	
7	Division	/	INT, DINT, UINT, UDINT, REAL	6/2	3	
8	Division remainder	MOD	INT, DINT, UINT, UDINT	17 MOD 10	7	
9	Addition	+	INT, DINT, UINT, UDINT, REAL	2+3	5	
10	Subtraction	-	INT, DINT, UINT, UDINT, REAL	4-2	2	
11	Comparison	<, >, <=, >=	Elementary (excluding STRING type)	4>12	FALSE	
12	Equality	=	Elementary (excluding STRING type)	T#26h =T#1d2h	TRUE	
13	Inequality	< >	Elementary (excluding STRING type)	8<>16	TRUE	
14	Logical product	&, AND	BOOL, WORD, DWORD	TRUE & FALSE	FALSE	
15	Exclusive or	XOR	BOOL, WORD, DWORD	TRUE XOR FALSE	TRUE	
16	Logical add	OR	BOOL, WORD, DWORD	TRUE OR FALSE	TRUE	Lowest

2-3-2 ST statements

Format	Description	Page
: =	Assignment statement. Assigns the value of the expression, variable, or numeric value on the right-hand side to the variable on the left-hand side.	P2-31
IF	Condition. Executes the executable statement if the conditional expression is true.	P2-32
CASE	Condition. Selects the executable statement to be executed according to the value of the conditional expression.	P2-33
FOR	Iteration statement. Repeatedly executes the executable statement according to the initial value, final value, and incremental or decremental value.	P2-34
WHILE	Iteration statement. Repeatedly executes the executable statement while the loop condition is true.	P2-35
REPEAT	Iteration statement. Repeatedly executes the executable statement until the loop condition is true.	P2-35
RETURN	Return statement. Returns control from the called function or function block to the calling POU.	P2-36
EXIT	Exit statement. Used to exit an iteration loop.	P2-36

2-3-3 ST language statements

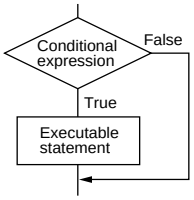
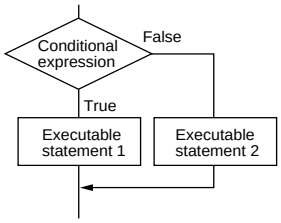
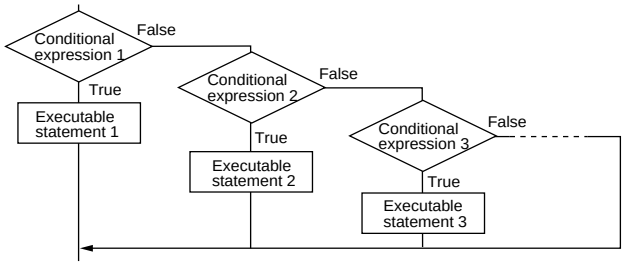
(1) Assignment statement (:=)

Name, Symbol, Function		Example

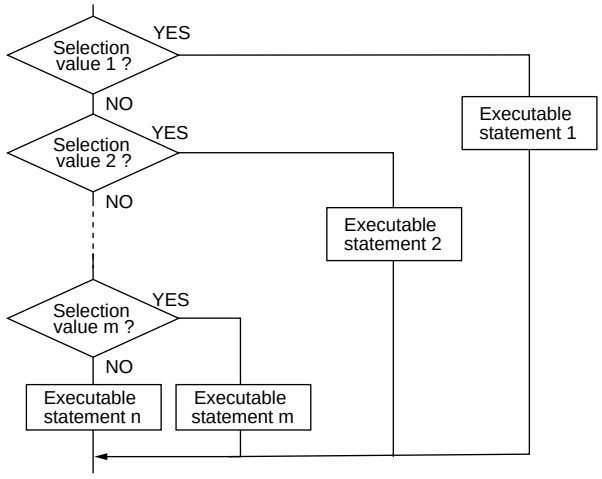
Name	Assignment statement	<Statement construction>
		variable := expression, variable, or numeric value;
		
		<Programming examples>
		• Assigning the average of a, b, and c to v
		v := (a + b + c)/3;
		• Assigning numeric value 52 to v
		v := 52;
		• Assigning the value of m to v.
		v := m;

Symbol	:=	
Function	(1) The expression on the right-hand side consists of operands, and instructions. The operands may be characters, variables, function calls, or other expressions. (2) The precedence of expression evaluation is determined by the operators involved; the operator with the highest precedence is applied to its operand first. This evaluation process is repeated until the evaluation ends. (See the operator summary charts on the previous pages for the precedence of the operators.) (3) The variable on the left-hand side and the expression on the right-hand side of an assignment statement must be of the same data type. When assigning data of a data type different from that of the target variable, it is necessary to convert the data type using a type conversion function.	

(2) IF statement

Name, Symbol, Function		Example
Name	IF statement	<Statement constructions> (1) IF, THEN construction
Symbol	IF	IF <conditional expression> THEN <executable statement>; END_IF;  The executable statement is executed when the conditional expression evaluates to be true. Nothing is executed if the conditional expression evaluates to be false.
Function	(1) The executable statement is executed if the conditional expression evaluates to be true. (2) The conditional expression can contain only the data that evaluates to BOOL type data. (3) An IF statement must always be paired with an END_IF statement. (4) A single conditional expression may have two or more executable statements. (5) The conditional expression may be a variable of type BOOL. (6) Up to eight levels of nesting are allowed. Even the FOR and CASE statements in the IF statement are counted in the nesting depth.	(2) IF, ELSE construction IF <conditional expression> THEN <executable statement 1>; ELSE <executable statement 2>; END_IF;  The executable statement 1 is executed when the conditional expression evaluates to be true. The executable statement 2 is executed when the conditional expression evaluates to be false. (3) IF, ELSIF construction IF <conditional expression1> THEN <executable statement 1>; ELSIF <conditional expression2> THEN <executable statement 2>; ELSIF <conditional expression3> THEN <executable statement 3>; ⋮ END_IF;  The executable statement 1 is executed when the conditional expression 1 evaluates to be true. The executable statement 2 is executed when the conditional expression 2 evaluates to be true. The executable statement 3 is executed when the conditional expression 3 evaluates to be true. Nothing is executed if none of the conditional expressions evaluate to be true. <Programming examples> The sample program shown below assigns 10 to M if N = 0, 20 to M if N = 1, 30 to M and 40 to L if N = 2. * It assigns 123 to M if N has a value other than 0, 1, and 2. IF N=0 THEN M:= 10; ELSIF N= 1 THEN M:= 20; ELSIF N= 2 THEN M:= 30; L:= 40; ELSE M:= 123; END_IF; *Example of two or more executable statements for a single conditional expression:

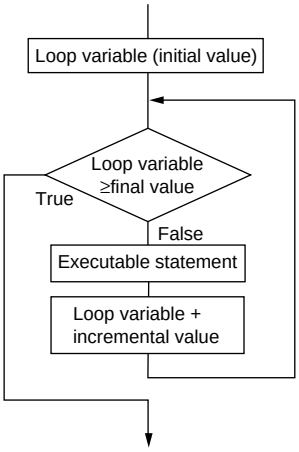
(3) CASE statement

Name, Symbol, Function		Example
Name	CASE statement	<Statement construction>
Symbol	CASE	CASE <conditional expression> OF <selection value 1>: <executable statement 1>; <selection value 2>: <executable statement 2>; ⋮ <selection value m>: <executable statement m> ELSE <executable statement n>; END_CASE;
Function	(1) The executable statement to be executed is selected according to the value is executed if the conditional expression evaluates to be true. (2) Only INT type data is allowed for the conditional expression and selection values. (3) A CASE statement must always be paired with OF and END_CASE statements. (4) A single conditional expression may have two or more executable statements. (5) Up to eight levels of nesting are allowed. Even the IF and FOR statements in the CASE statement are counted in the nesting depth. (6) An enum type (example, 1,2,2,...) or range type (example, 1..10) may be used for the selection values.	 <Programming example> This program assigns 1 to signal when v (an average of a, b, and c) = 10, 2 when V = 20, 3 when V = any of a range of 21 to 31, and 0 in other cases. v:= (a + b + c)/ 3 ; CASE v OF 10: signal :=1; 20: signal :=2; 21..30: signal :=3; ELSE signal :=0; END_CASE;

(4) FOR statement

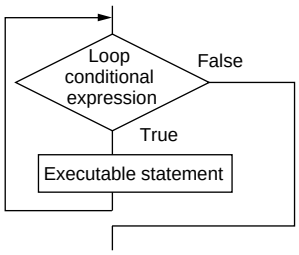
Name, Symbol, Function		Example
Name	FOR statement	<Statement construction>

Symbol	FOR	FOR <a:=initial value> TO <final value> BY <incremental value> DO
		<executable statement>
		END FOR;

Function	(1) The FOR statement performs loop processing. The loop conditions are given by the initial value, final value, and incremental value.
	(2) The type of data that can be used as the initial value, final value, and incremental value is ANY_INT type (INT, DINT, UNIT, or UDINT).
	(3) A FOR statement must always be paired with TO, BY, DO, and END_FOR statements.
	(4) Up to eight levels of nesting are allowed. Even the IF and CASE statements in the FOR statement are counted in the nesting depth.
		
		<Programming example>
		The sample program shown below assigns the value of b to array f at every other index positions starting at a = 1.
		(a=1, a=3, a=5, a=7, a=9)
		
		FOR a:=1 TO 10 BY 2 DO
		f [a]:=b;
		END FOR;

(5) WHILE statement

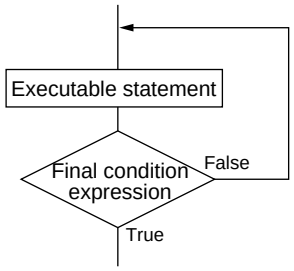
Name, Symbol, Function		Example

Name	WHILE statement	<Statement construction>
		WHILE <loop condition> DO
		<executable statement>;
		END_WHILE;
		 
		
		<Programming example>
		The sample program shown below divides the value of b by 2 while b > 1 and assigns the result (quotient) to b. The fractional part of the quotient is truncated b.
		
		a:=1;
		WHILE b>1 DO
		b:=b/2;
		f[a]:=b;
		a:=a+1;
		END_WHILE;

Symbol	WHILE	
Function	(1) The WHILE statement performs loop processing. It executes the executable statement while its loop condition evaluates to be true. (2) A WHILE statement must always be paired with DO and END_WHILE statements. Note) If a repeat condition has been false before the loop processing is executed, no Execution statement is processed.	

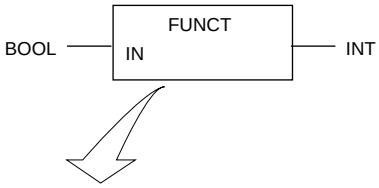
(6) REPEAT statement

Name, Symbol, Function		Example

Name	REPEAT statement	<Statement construction>
		REPEAT
		<executable statement>;
		UNTIL <final condition>
		END_REPEAT;
		 
		
		<Programming example>
		The sample program shown below computes a sum of numbers from 1 to 10.
		
		a:=1;
		total:=0;
		REPEAT
		total:=total+a;
		a:=a+1;
		UNTIL a >10
		END_REPEAT;

Symbol	REPEAT	
Function	(1) The REPEAT statement performs loop processing. It executes the executable statement until its loop condition evaluates to be true. (2) A REPEAT statement must always be paired with UNTIL and END_REPEAT statements. Note) Even if the value for a final condition has been true before the loop processing is executed, the execution statement is always processed more than one time.	

(7) RETURN statement

Name, Symbol, Function		Example
Name	RETURN statement	<Statement construction> RETURN;
Symbol	RETURN	<Programming example> When returning control to the user function “FUNCT”
Function	(1) The RETURN statement returns control from a user function or usr function block to the calling POU.	 Function of <FUNCT> FUNCT returns 500 if IN is true and 100 if IN is false. IF IN=TRUE THEN FUNCT:=500; (* Sets the return value to 500 *) RETURN; (* Returns without executing the *) END_IF; (* following statements *) FUNCT:=100; RETURN;

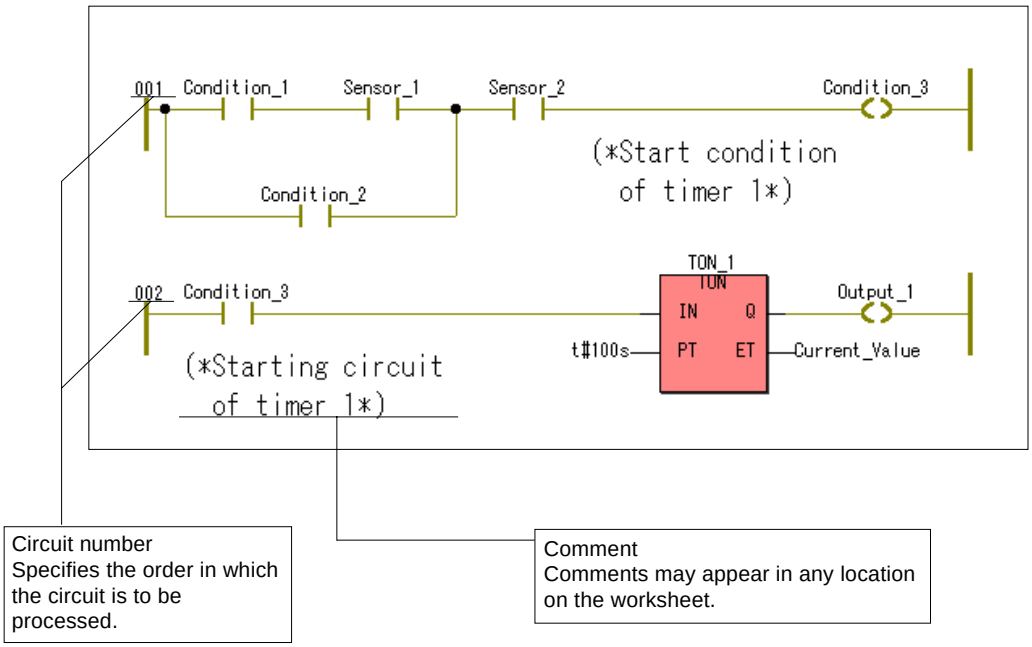
(8) EXIT statement

Name, Symbol, Function		Example
Name	EXIT statement	<Statement construction> EXIT;
Symbol	EXIT	<Programming examples> The sample program shown below exits the loop when M + N becomes greater than 20 where M is the initial value.

Function	(1) The EXIT statement terminates processing and returns control to the statement immediately following the loop construction in which the EXIT statement is used.	(* Exiting from a FOR loop *)
		FOR N:=1 TO 10 BY 1 DO
		M:=M+N ;
		IF M > 20 THEN EXIT ;
		END_IF ;
		END_FOR ;
		
		(* Exiting from a WHILE loop *)
		N:=1 ;
		WHILE N <= 10 DO ;
		M:=M+N ;
		IF M > 20 THEN EXIT ;
		END_IF ;
		N:=N+1;
		END_WHILE ;
		
		(* Exiting from a REPEAT loop *)
		N:=1 ;
		REPEAT
		M:=M+N ;
		IF M > 20 THEN EXIT ;
		END_IF ;
		N:=N+1 ;
		UNTIL N > 10
		END_REPEAT ;

LD language is a graphical language that consists of contacts and coils that are connected to two vertical bus lines. LD language are used with FBD language in applications where timers, counters, and applications instructions are to be used.

<Examples of LD language>



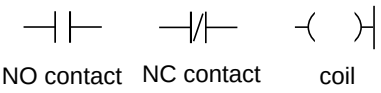
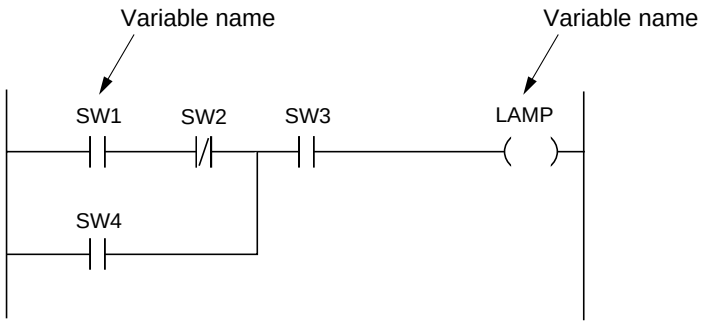
Note: The graphical languages supported by the MICREX-SX series are of the free-layout format. Multiple circuits may be laid out in any location on a work sheet.

2-4-1 LD language

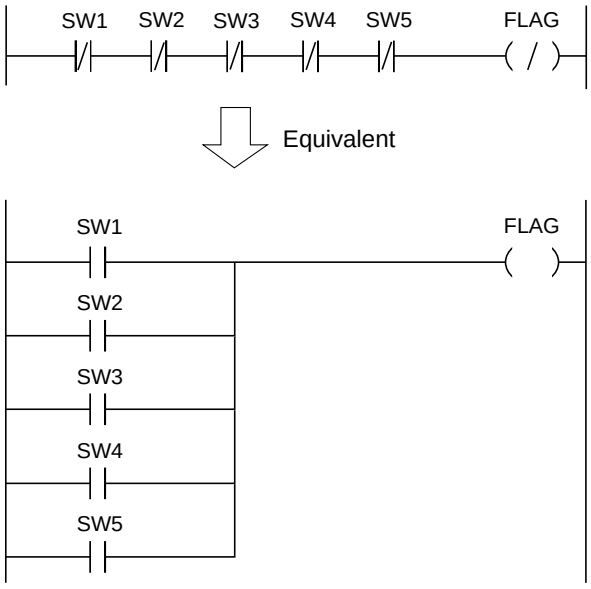
Instruction	Name	No. of steps	Page
⊣ ⊢	Normal open contact (NO contact)	1	P2-38
⊣/⊢	Normal close contact (NC contact)	1	P2-38
-( )-	Coil	1	P2-38
-(/)-	Inverted coil	1	P2-38
-(S)-	Set	1	P2-39
-(R)-	Reset	1	P2-39
→"connector name"	Connect to "connector name"	1	P2-39
"connector name"→	Connect from "connector name"	1	P2-39
→→"label name"	Source of jump	1	P2-40
"label name" :	Label of jump destination	0	P2-40
←RETURN→	Return	1	P2-40

2-4-2 LD language instructions

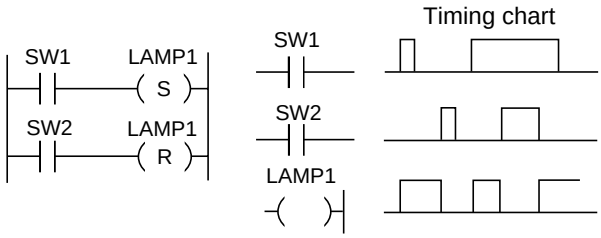
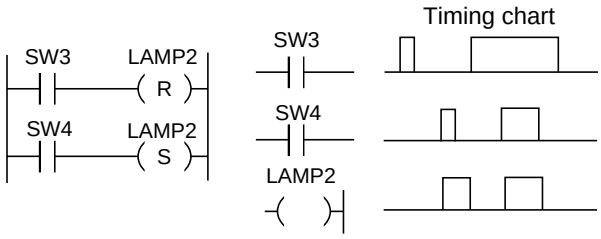
(1) Normal open contact (NO contact), normal close contact (NC contact), and coil

Name, Symbol, Function		Example
Name	NO contact, NC contact, coil	<Programming example> Variable representation example
Symbol		
Function	(1) These symbols denote NO contacts, NC contacts, and coils, respectively. Their address may be designated by variables or direct addressing representations. (2) There is no restriction on the number of contacts that can be assigned to the same address. (3) Available data type is BOOL type.	*: There is no restriction on the number of contacts in both row and column directions.

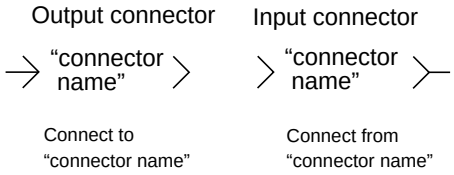
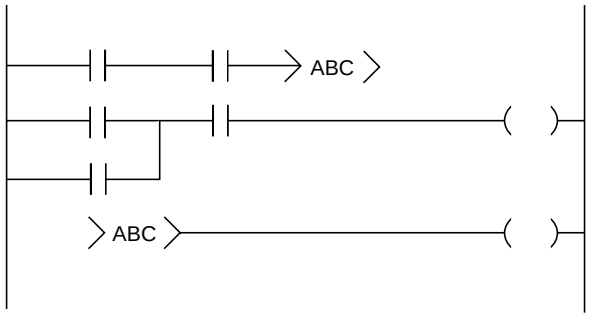
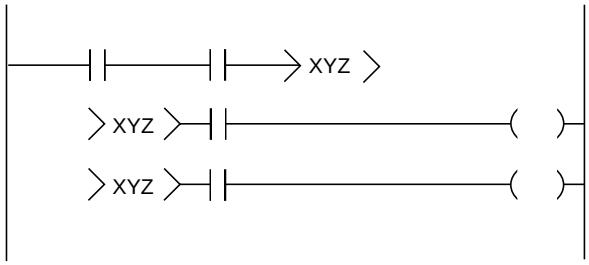
(2) Inverted coil

Name, Symbol, Function		Example
Name	Inverted coil	<Programming example>
Symbol		
Function	(1) The inverted coil generates the inverted value of a boolean object.	The upper and lower circuits are equivalent.

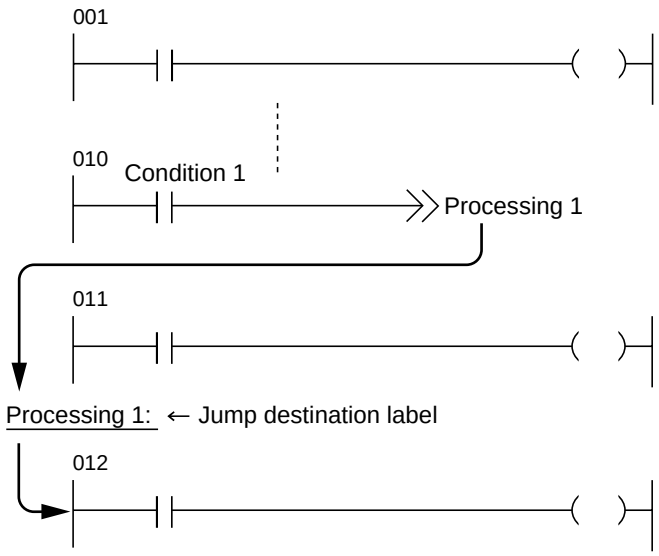
(3) Set coil, reset coil

Name, Symbol, Function		Example
Name	Set coil, reset coil	<Programming examples> • Reset-prioritized circuit
Symbol		
Function	(1) The set coil turns on when its set input goes on and remains on when the set input goes off. The set coil turns off when its reset input goes on. (2) A pair of set and reset coils are referenced by the same variable.	• Set-prioritized circuit 

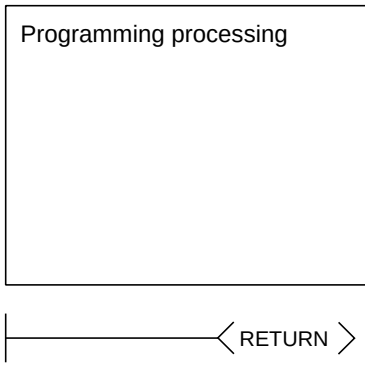
(4) Connector

Name, Symbol, Function		Example
Name	Connector	<Programming examples> • The sample circuit shown below assumes that the source is connected to the destination.
Symbol		
Function	(1) Connectors substitute lines. The output and input connectors must be given the same name. (2) There can be one output connector and two or more input connectors with the same name. (3) A connector name must be not longer than 30 1-byte characters or 15 2-byte characters. Note: Connectors must be described so that they are completed on a single work sheet.	• Example of using one output connector and two input connectors having the same name. 

(5) Jump

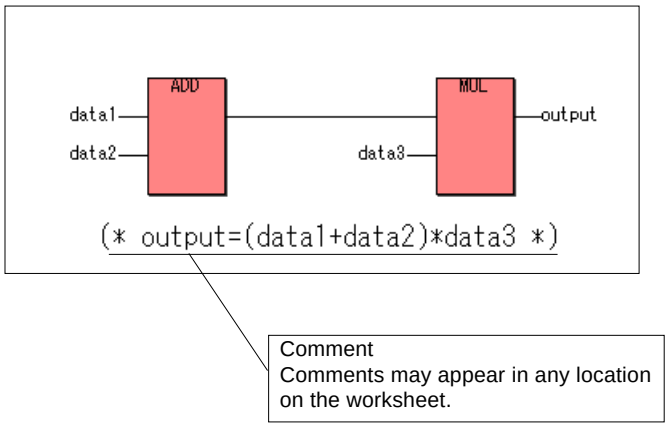
Name, Symbol, Function		Example
Name	Jump	<Programming example>  Note: The source and destination circuits must be described separately. <Operation> Program execution causes a jump to Processing 1 and circuit 012 is processed when condition 1 turns on. Circuit 011 is skipped.
Symbol	Jump destination label (jump source) Jump destination label : (jump destination)	
Function	(1) The jump instruction transfers control from the jump source to the jump destination. (2) The label name of the jump destination can be up to 30 1-byte characters (or 15 2-byte characters).	

(6) Return

Name, Symbol, Function		Example
Name	Return	
Symbol	RETURN	
Function	(1) The return instruction returns control from a user function or user function block to the calling POU.	

The FBD language allows block diagrams to be used to represent programs. FBD language clearly show the relationship between arithmetic instructions and input/output and visually show the flow of operation. For example, the operation “output = (data1 + data2) x data3” can be represented as shown below in FBD language.

<Sample FBD language representation>



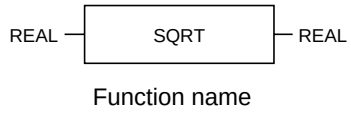
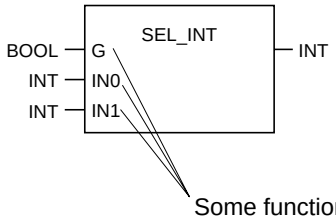
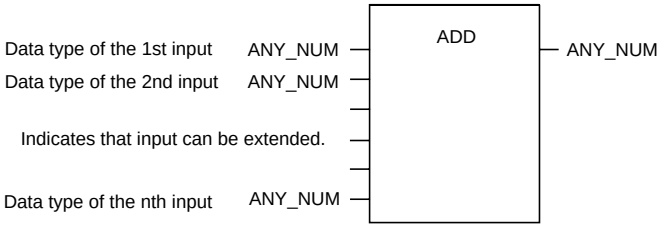
Note: The graphical languages supported by the MICREX-SX series are of the free-layout format. Multiple circuits may be laid out in any location on a work sheet.

2-5 FBD Language

2-5-1 Function summary

Functions are supported by the IL, ST, LD, and FBD languages. The operation of a function remains the same for these languages.

(1) Symbols used in the function summary

Instruction Symbol	Name	No. of steps	Page
Input data type Output data type  Function name	Square root SQRT	3	2-○
	Get left sub-string LEFT	10	2-○
 Some functions use terminal names.	Select SEL_INT	8	2-○
	Arithmetic addition ADD	2 x number of input operands	2-○

Note: 1) The number of steps depends on the operands used. Consequently, the number of steps of array variables will be greater than that of scalar variables.
2) The number of extended inputs is a maximum of 16 (only for SPH).
3) D300win do not check the data type of the terminals associated with functions ANY and ANY_OF_WORD. Be sure to use the data types appropriate for individual functions, following their detailed descriptions.

(2) Describing a function in the IL language

To describe a function call in the IL language, specify, as the first input, the current operation result and, as the operator, the function call instruction (function name) that specifies the second and subsequent inputs as operands.

1) 1-input function

```
LD    “Input”
SQRT
ST    “Area for storing operation result"
```

2) 2-input function

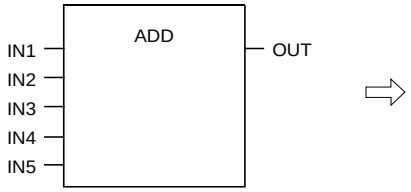
```
LD    “First input”
LEFT  “Second input”
ST    “Area for storing operation result”
```

3) 3-input function

```
LD    “First input”
SEL   “Second input,” “Third input”
ST    “Area for storing operation result”
```

Note: You cannot specify, in the IL language, any function that has the same mnemonic as an IL instruction (e.g. ADD). Instructions such as "ADD" must always have two fixed inputs when they are to be used in the IL language.

<Representation in FBD or LD language>



<Representation in IL language>

```
LD    IN1
ADD   IN2
ADD   IN3
ADD   IN4
ADD   IN5
ST    OUT
```

(3) Describing a function in the ST language

1) 1-input function

```
“Area for storing operation result” := SQRT (“Input”);
```

2) 2-input function

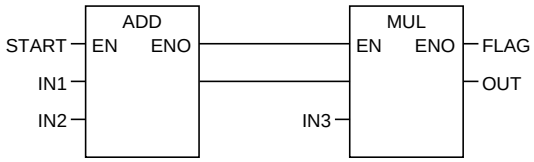
```
“Area for storing operation result” := LEFT
(“First input,” “Second input”);
```

3) 3-input function

```
“Area for storing operation result” := SEL
(“First input,” “Second input,” “Third input”);
```

(4) Specification of enable flags (EN/ENO) (only for SPH)
In D300win V2.0 or later, the start operation terminal EN and result terminal ENO can be used for the functions in the graphic language.

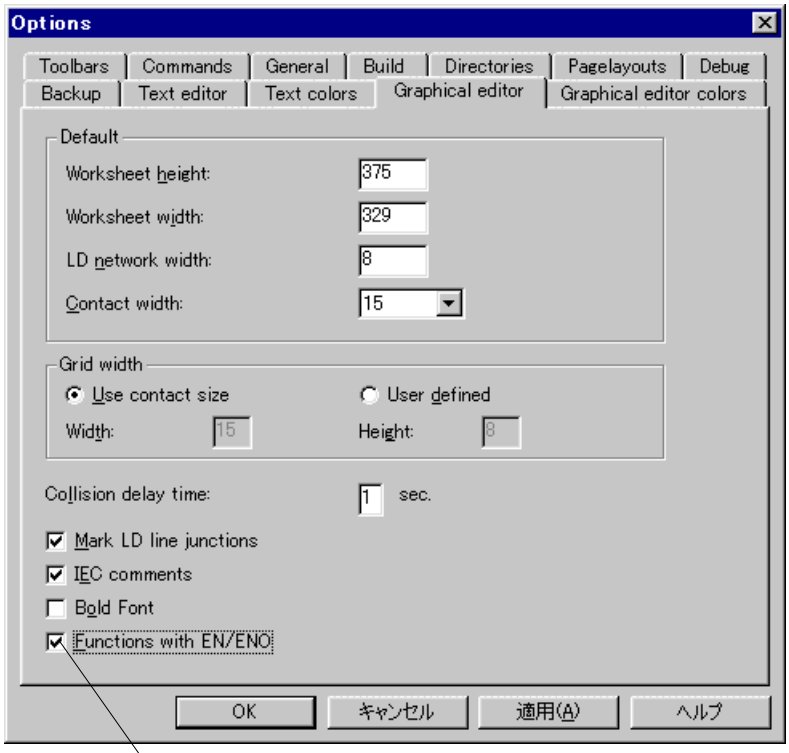
<Example of a function with EN/ENO>



Associate a BOOL type operation condition variable with the EN terminal. When the variable associated with the EN terminal is TRUE (“1”), arithmetical operations are executed, while when FALSE (“0”), they are not performed. (In the above example, an OUT value remains unchanged.) A TRUE or FALSE value for the operation is output to the ENO terminal. When ENO is TRUE (“1”), the operation has been normally done and when FALSE (“0”), an error has occurred in the operation.

Note: Refer to the descriptions of functions, for EN/ENO behavior.

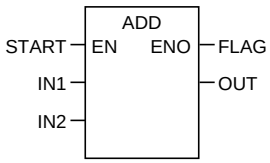
<Enabling/disabling EN/ENO>
Execute the “Option” command shown in the D300win extended menu. The “Options” dialog box appears. Click the “Graphic Editor” tab with the left-mouse button, check the EN/ENO function, and double-click the “OK” button with the left-mouse button, since any function added is shown with EN/ENO enabled on a worksheet. To disable EN/ENO, use the same procedure.



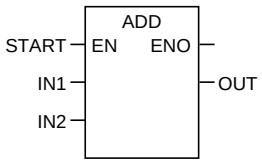
Checking here enables EN/ENO for the function newly added. Note that for the existing functions, EN/ENO remains disabled.

<Numbers of function steps with EN/ENO enabled>
The numbers of function steps with EN/ENO enabled are shown below.

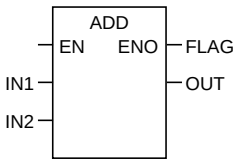
1) When variables are associated with both of EN/ ENO terminals, the number of steps increases by three.



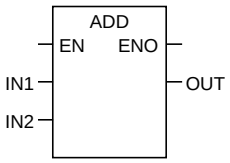
2) When a variable is associated only with the EN terminal, the number of steps increases by two.



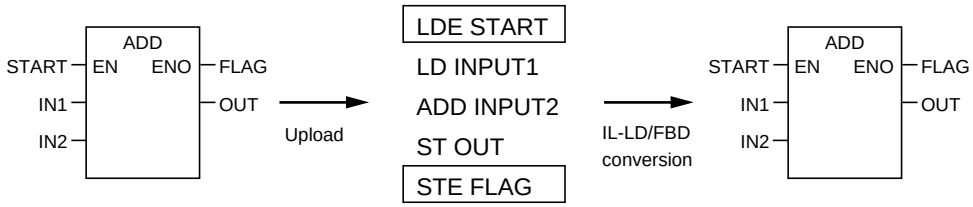
3) When a variable is associated only with the ENO terminal, the number of steps increases by two.



4) When a variable is associated with neither the EN nor ENO terminal, the number of steps is the same as with EN/ENO disabled.



<LI language when any function with EN/ENO enabled is uploaded>
When a function with EN/ENO enabled is uploaded from the CPU, LDE and STE IL operators are added. Conversion of IL to LD/FBD returns the function to its original format.

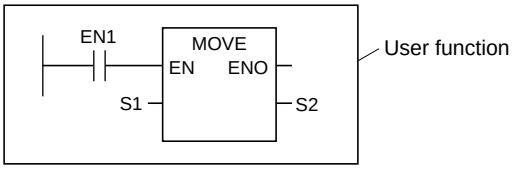


Note: To upload functions in D300win V2, connect both of the EN/ENO terminals.

<Considerations in using functions with EN/ENO enabled>

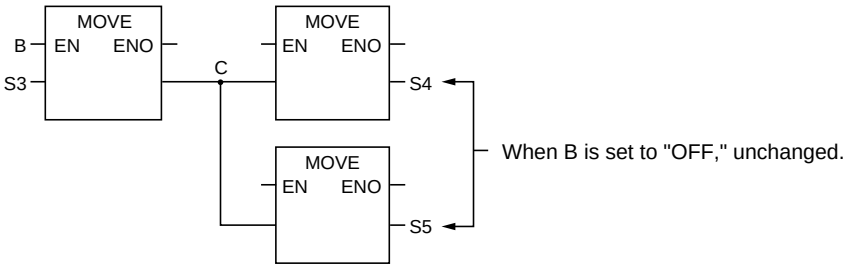
1) Using user functions

When the circuitry shown in the diagram below is created by using a user function, S2 is assigned to the temporary area. If a variable A associated with the EN terminal is set to “OFF,” an S2 value is undefined.

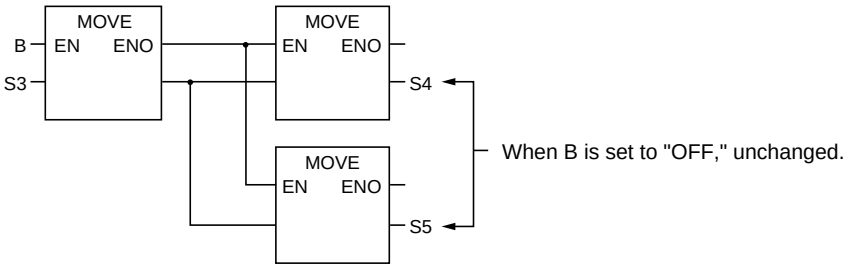


2) Using programs and user function blocks

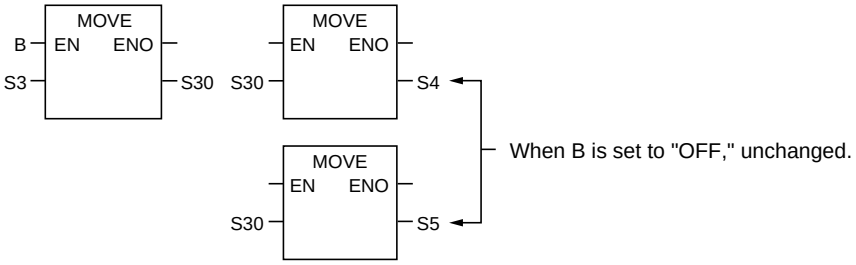
When the circuitries shown in the diagrams below are created by using a program or user function block, portion C is assigned to the temporary area. If a variable B associated with the EN terminal is set to “OFF,” C is undefined, resulting in S4 and S5 being undefined. In any such case, take either one of actions 1 and 2.



<Action 1> Connect EN/ENO.

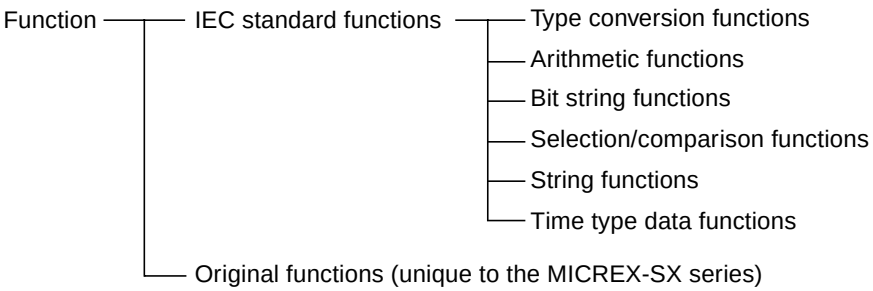


<Action 2> Use a variable explicitly for portion C.



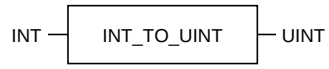
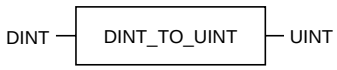
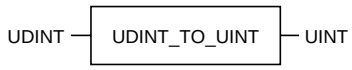
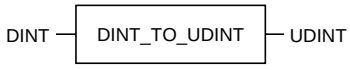
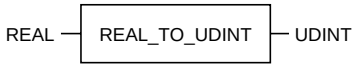
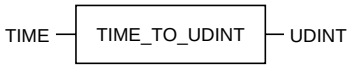
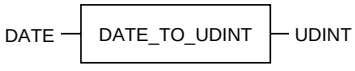
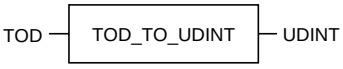
(5) Function summary

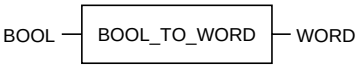
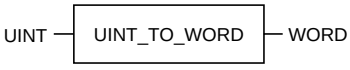
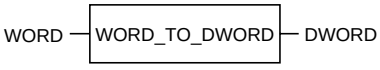
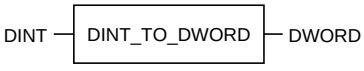
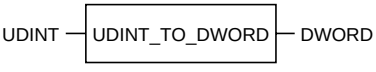
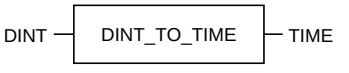
The functions that are supported by the MICREX-SX series are classified into the following categories:

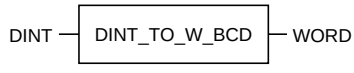


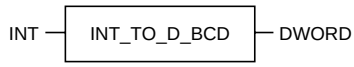
1) Type conversion functions

Instruction Symbol	Name	No. of steps	Page
DINT — DINT_TO_INT — INT	Type conversion DINT_TO _INT	3	P2-76
UINT — UINT_TO_INT — INT	Type conversion UINT_TO _INT	3	P2-76
UDINT — UDINT_TO_INT — INT	Type conversion UDINT_TO _INT	3	P2-77
REAL — REAL_TO_INT — INT	Type conversion REAL_TO _INT	3	P2-77
TIME — TIME_TO_INT — INT	Type conversion TIME_TO _INT	3	P2-78
WORD — WORD_TO_INT — INT	Type conversion WORD_TO _INT	3	P2-78
INT — INT_TO_DINT — DINT	Type conversion INT_TO _DINT	3	P2-79
UINT — UINT_TO_DINT — DINT	Type conversion UINT_TO _DINT	3	P2-79
UDINT — UDINT_TO_DINT — DINT	Type conversion UDINT_TO _DINT	3	P2-80
REAL — REAL_TO_DINT — DINT	Type conversion REAL_TO _DINT	3	P2-80
TIME — TIME_TO_DINT — DINT	Type conversion TIME_TO _DINT	3	P2-81
DWORD — DWORD_TO_DINT — DINT	Type conversion DWORD_TO _DINT	3	P2-81

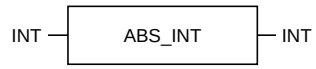
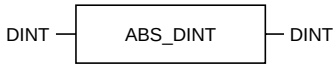
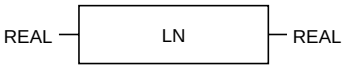
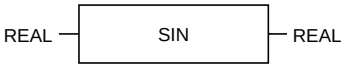
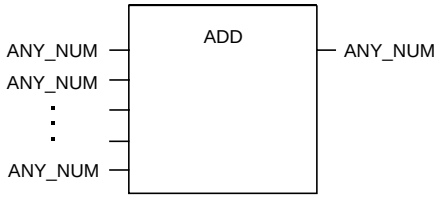
Instruction Symbol	Name	No. of steps	Page
	Type conversion INT_TO _UINT	3	P2-82
	Type conversion DINT_TO _UINT	3	P2-82
	Type conversion UDINT_TO _UINT	3	P2-83
	Type conversion REAL_TO _UINT	3	P2-83
	Type conversion TIME_TO _UINT	3	P2-84
	Type conversion WORD_TO _UINT	3	P2-84
	Type conversion INT_TO _UDINT	3	P2-85
	Type conversion DINT_TO _UDINT	3	P2-85
	Type conversion UINT_TO _UDINT	3	P2-86
	Type conversion REAL_TO _UDINT	3	P2-86
	Type conversion TIME_TO _UDINT	3	P2-87
	Type conversion DWORD_TO _UDINT	3	P2-87
	Type conversion DT_TO _UDINT Note: These functions are not supported in SPS.	3	P2-88
	Type conversion DATE_TO _UDINT Note: These functions are not supported in SPS.	3	P2-88
	Type conversion TOD_TO _UDINT Note: These functions are not supported in SPS.	3	P2-89
	Type conversion INT_TO _REAL	3	P2-89

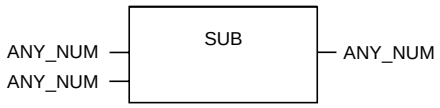
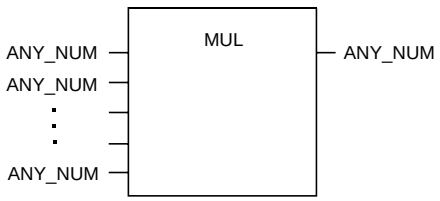
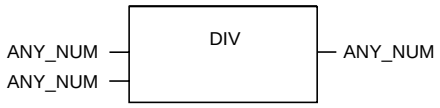
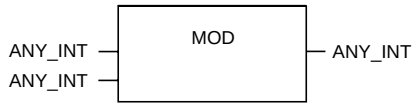
Instruction Symbol	Name	No. of steps	Page
	Type conversion DINT_TO _REAL	3	P2-90
	Type conversion UINT_TO _REAL	3	P2-90
	Type conversion UDINT_TO _REAL	3	P2-91
	Type conversion TIME_TO _REAL	3	P2-91
	Type conversion WORD_TO _BOOL	3	P2-92
	Type conversion DWORD_TO _BOOL	3	P2-92
	Type conversion BOOL_TO _WORD	3	P2-93
	Type conversion DWORD_TO _WORD	3	P2-93
	Type conversion INT_TO_WORD	3	P2-94
	Type conversion UINT_TO_WORD	3	P2-94
	Type conversion BOOL_TO _DWORD	3	P2-95
	Type conversion WORD_TO _DWORD	3	P2-95
	Type conversion DINT_TO _DWORD	3	P2-96
	Type conversion UDINT_TO _DWORD	3	P2-96
	Type conversion INT_TO _TIME	3	P2-97
	Type conversion DINT_TO _TIME	3	P2-97

Instruction Symbol	Name	No. of steps	Page
	Type conversion UINT_TO _TIME	3	P2-98
	Type conversion UDINT_TO _TIME	3	P2-98
	Type conversion REAL_TO _TIME	3	P2-99
	Type conversion UDINT_TO _DT Note: These functions are not supported in SPS.	3	P2-99
	Type conversion UDINT_TO _DATE Note: These functions are not supported in SPS.	3	P2-100
	Type conversion UDINT_TO _TOD Note: These functions are not supported in SPS.	3	P2-100
	Truncation TRUNC_INT	3	P2-101
	Truncation TRUNC_DINT	3	P2-101
	Truncation TRUNC_UINT	3	P2-102
	Truncation TRUNC_UDINT	3	P2-102
	BCD conversion W_BCD_TO _INT	3	P2-103
	BCD conversion D_BCD_TO _INT	3	P2-103
	BCD conversion W_BCD_TO _DINT	3	P2-104
	BCD conversion D_BCD_TO _DINT	3	P2-104
	BCD conversion INT_TO _W_BCD	3	P2-105
	BCD conversion DINT_TO _W_BCD	3	P2-105

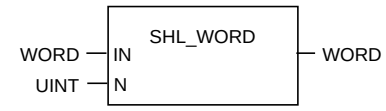
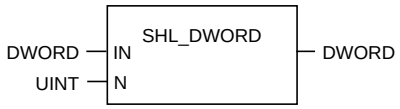
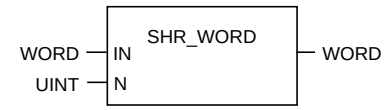
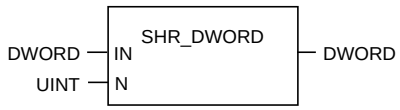
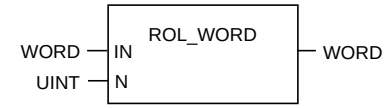
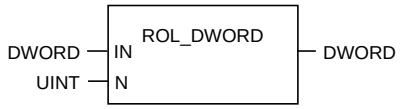
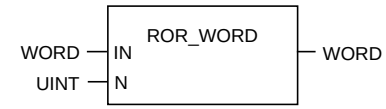
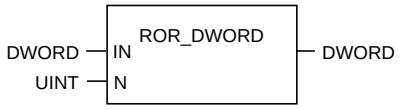
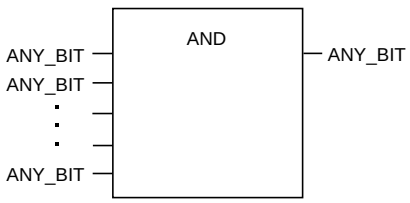
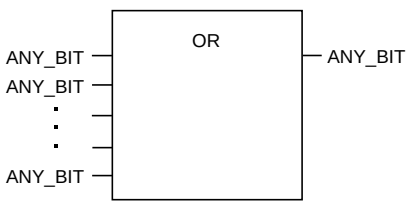
Instruction Symbol	Name	No. of steps	Page
	BCD conversion INT_TO _D_BCD	3	P2-106
	BCD conversion DINT_TO _D_BCD	3	P2-106

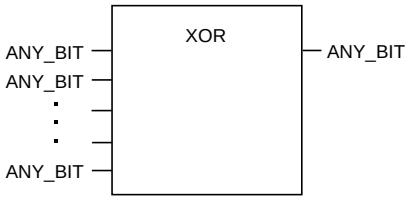
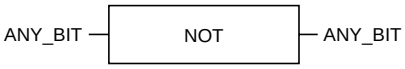
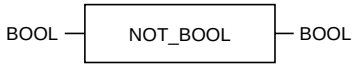
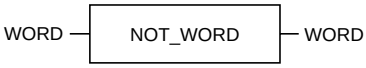
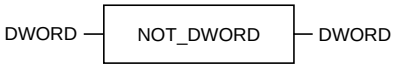
2) Arithmetic functions

Instruction Symbol	Name	No. of steps	Page
	Absolute value ABS _INT	3	P2-107
	Absolute value ABS _DINT	3	P2-107
	Absolute value ABS _REAL	3	P2-108
	Square root SQRT	3	P2-108
	Natural logarithm LN	3	P2-109
	Common logarithm LOG	3	P2-109
	Exponent EXP	3	P2-110
	Sine SIN	3	P2-110
	Cosine COS	3	P2-111
	Tangent TAN	3	P2-111
	Arc sine ASIN	3	P2-112
	Arc cosine ACOS	3	P2-112
	Arc tangent ATAN	3	P2-113
	Addition ADD	No. of input operands +1	P2-113

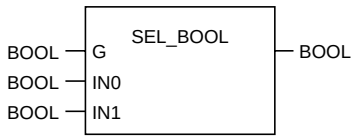
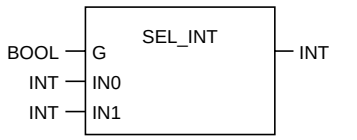
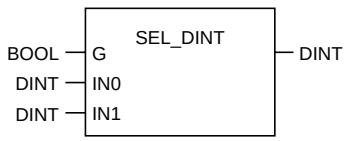
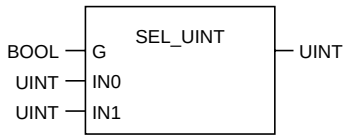
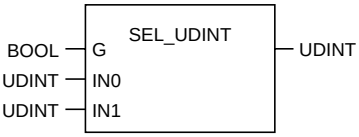
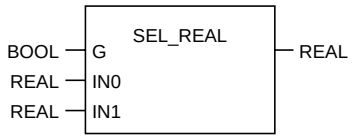
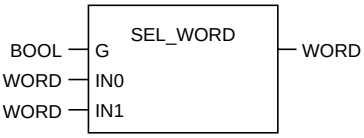
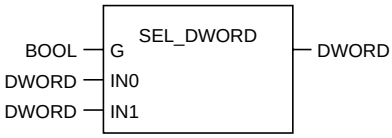
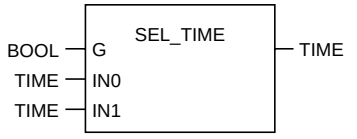
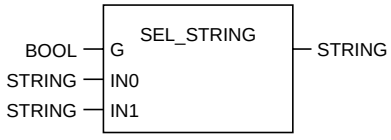
Instruction Symbol	Name	No. of steps	Page
	Subtraction SUB	3	P2-114
	Multiplication MUL	No. of input operands +1	P2-114
	Division DIV	3	P2-115
	Division remainder MOD	4	P2-115
	Exponent EXPT	3	P2-116
	Move MOVE	4	P2-116
	Negation NEG	3	P2-117

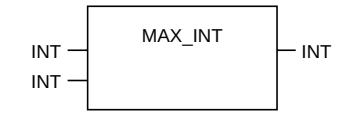
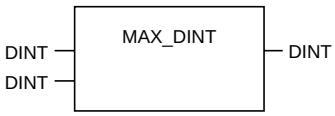
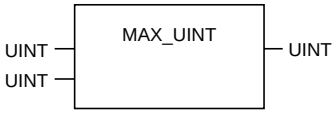
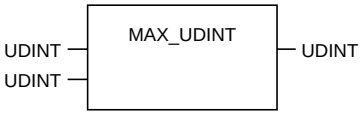
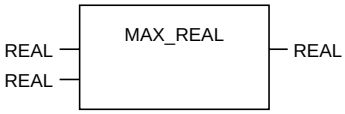
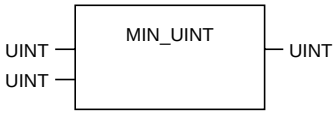
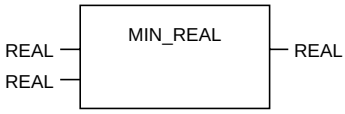
3) Bit string functions

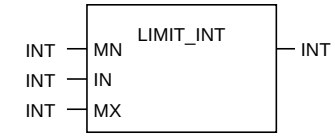
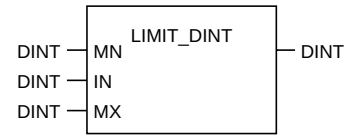
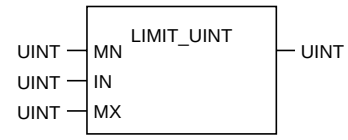
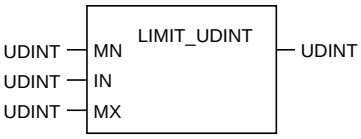
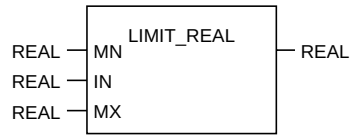
Instruction Symbol	Name	No. of steps	Page
	Shift left SHL _WORD	3	P2-118
	Shift left SHL _DWORD	3	P2-118
	Shift right SHR _WORD	3	P2-119
	Shift right SHR _DWORD	3	P2-119
	Rotate left ROL _WORD	3	P2-120
	Rotate left ROL _DWORD	3	P2-120
	Rotate right ROR _WORD	3	P2-121
	Rotate right ROR _DWORD	3	P2-121
	Logical product AND	No. of input operands +1	P2-122
	Logical add OR	No. of input operands +1	P2-122

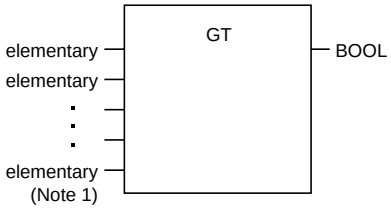
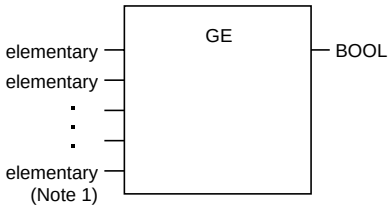
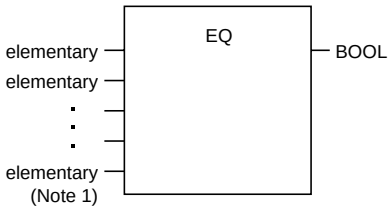
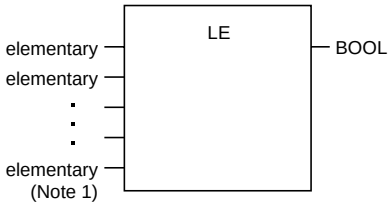
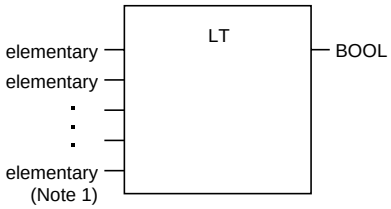
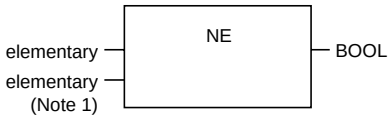
Instruction Symbol	Name	No. of steps	Page
	Exclusive XOR	No. of input operands +1	P2-123
	Logical negation NOT	3	P2-123
	Negation NOT_BOOL	3	P2-124
	Negation NOT_WORD	3	P2-124
	Negation NOT_DWORD	3	P2-125

4) Selection/comparison functions

Instruction Symbol	Name	No. of steps	Page
	Select SEL_BOOL	8	P2-126
	Select SEL_INT		
	Select SEL_DINT		
	Select SEL_UINT		
	Select SEL_UDINT		
	Select SEL_REAL		
	Select SEL_WORD		
	Select SEL_DWORD		
	Select SEL_TIME		
	Select SEL_STRING		

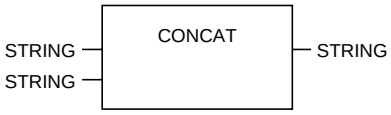
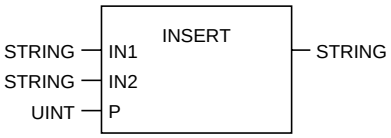
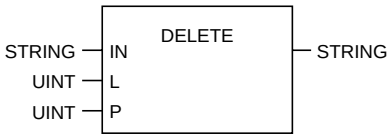
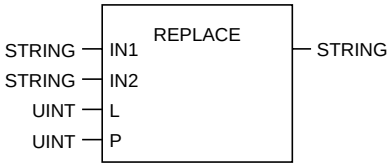
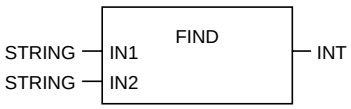
Instruction Symbol	Name	No. of steps	Page
	Maximum value MAX_INT	3	P2-127
	Maximum value MAX_DINT		
	Maximum value MAX_UINT		
	Maximum value MAX_UDINT		
	Maximum value MAX_REAL		
	Minimum value MIN_INT	3	P2-128
	Minimum value MIN_DINT		
	Minimum value MIN_UINT		
	Minimum value MIN_UDINT		
	Minimum value MIN_REAL		

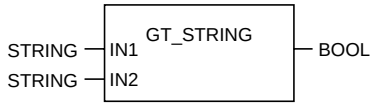
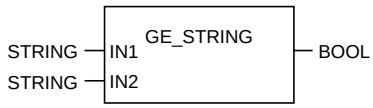
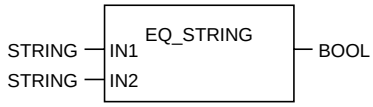
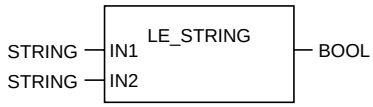
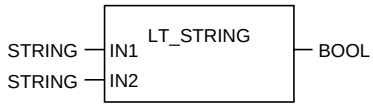
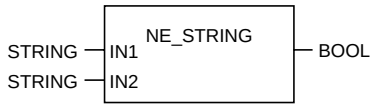
Instruction Symbol	Name	No. of steps	Page
	Limit LIMIT_INT	6	P2-128
	Limit LIMIT_DINT		
	Limit LIMIT_UINT		
	Limit LIMIT_UDINT		
	Limit LIMIT_REAL		

Instruction Symbol	Name	No. of steps	Page
	Comparison (>) GT	No. of input operands x 3-1	P2-129
	Comparison (≥) GE	No. of input operands x 3-1	P2-130
	Comparison (=) EQ	No. of input operands x 3-1	P2-131
	Comparison (≤) LE	No. of input operands x 3-1	P2-132
	Comparison (<) LT	No. of input operands x 3-1	P2-133
	Comparison (≠) NE	3	P2-134

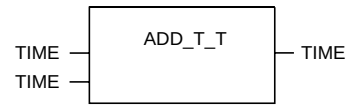
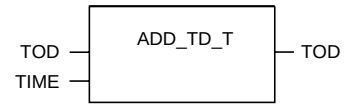
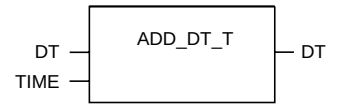
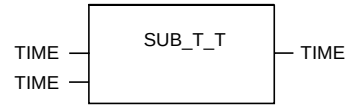
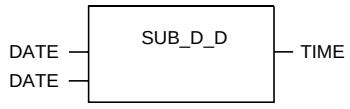
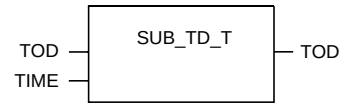
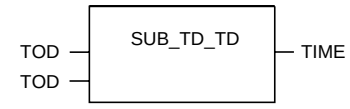
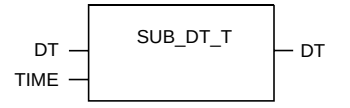
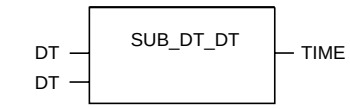
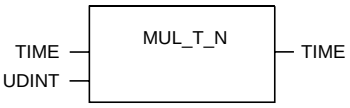
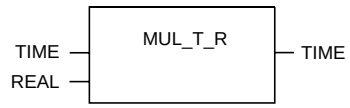
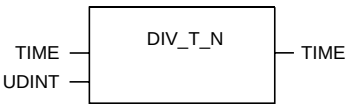
Note: 1) When SPS is used, the input cannot be extended.
2) STRING type data is not available to an “elementary” of GT, GE, EQ, LE, LT or NE.

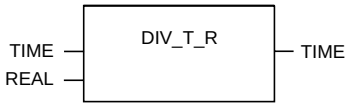
5) String functions

Instruction Symbol	Name	No. of steps	Page
	Get length LEN	3	P2-135
	Get left sub-string LEFT	10	P2-135
	Get right sub-string RIGHT	10	P2-136
	Get middle sub-string MID	11	P2-136
	Concatenate CONCAT	11	P2-137
	Insert string INSERT	11	P2-137
	Delete string DELETE	11	P2-138
	Replace string REPLACE	12	P2-138
	Find string FIND	4	P2-139

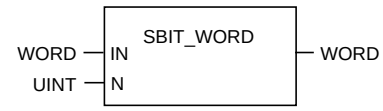
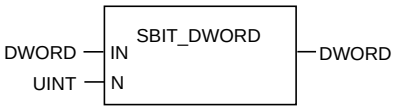
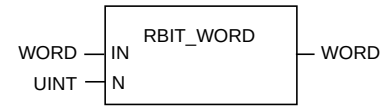
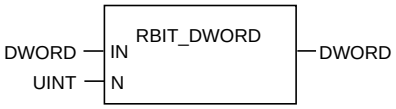
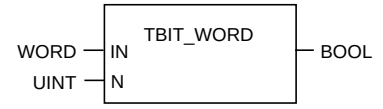
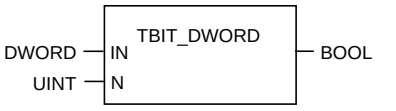
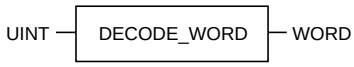
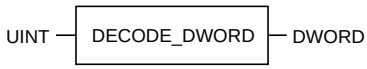
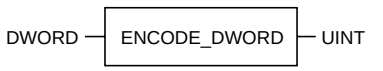
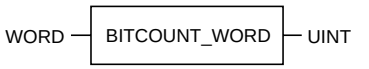
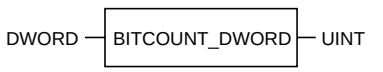
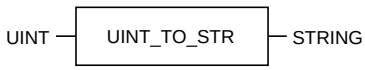
Instruction Symbol	Name	No. of steps	Page
	Compare string GT_STRING	4	P2-139
	Compare string GE_STRING	4	P2-140
	Compare string EQ_STRING	4	P2-140
	Compare string LE_STRING	4	P2-141
	Compare string LT_STRING	4	P2-141
	Compare string NE_STRING	4	P2-142

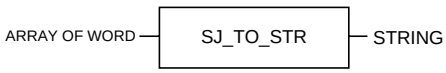
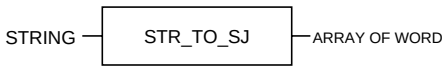
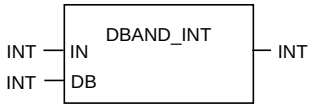
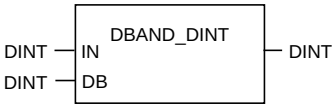
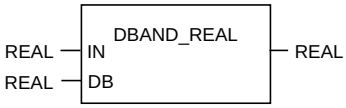
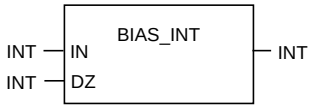
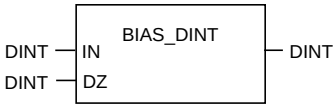
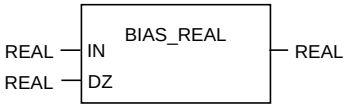
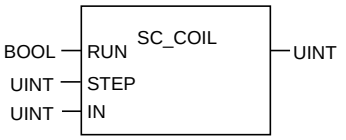
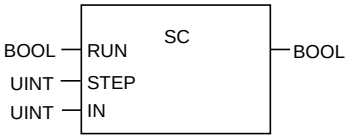
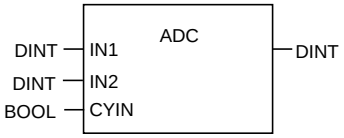
6) Time type data functions

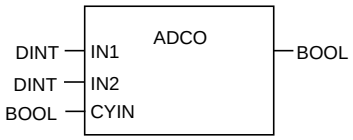
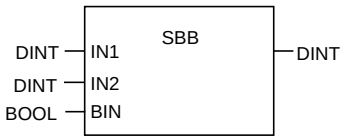
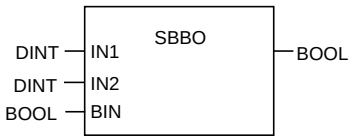
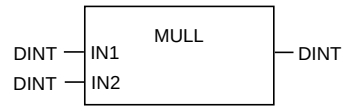
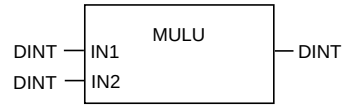
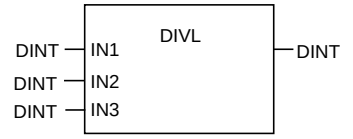
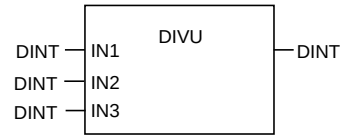
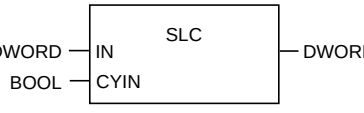
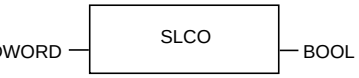
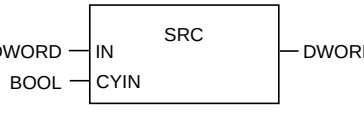
Instruction Symbol	Name	No. of steps	Page
	Add time ADD_T_T	5	P2-143
	Add time ADD_TD_T Note: These functions are not supported in SPS.	8	P2-143
	Add time ADD_DT_T Note: These functions are not supported in SPS.	8	P2-143
	Subtract time SUB_T_T	5	P2-144
	Subtract time SUB_D_D Note: These functions are not supported in SPS.	6	P2-144
	Subtract time SUB_TD_T Note: These functions are not supported in SPS.	8	P2-144
	Subtract time SUB_TD_TD Note: These functions are not supported in SPS.	6	P2-145
	Subtract time SUB_DT_T Note: These functions are not supported in SPS.	8	P2-145
	Subtract time SUB_DT_DT Note: These functions are not supported in SPS.	6	P2-145
	Multiply time MUL_T_N	7	P2-146
	Multiply time MUL_T_R	7	P2-146
	Divide time DIV_T_N	7	P2-147

Instruction Symbol	Name	No. of steps	Page
	Divide time DIV_T_R	7	P2-147
	Concatenate time CONCAT_D_D Note: These functions are not supported in SPS.	5	P2-148
	Convert DT to TOD DT_TO_TOD Note: These functions are not supported in SPS.	5	P2-148
	Convert DT to DATE DT_TO_DATE Note: These functions are not supported in SPS.	6	P2-148

7) Original FCTs (Functions)

Instruction Symbol	Name	No. of steps	Page
	Set bit SBIT_WORD	3	P2-149
	Set bit SBIT_DWORD	3	P2-149
	Reset bit RBIT_WORD	3	P2-150
	Reset bit RBIT_DWORD	3	P2-150
	Test bit TBIT_WORD	3	P2-151
	Test bit TBIT_DWORD	3	P2-151
	Decode DECODE_WORD	3	P2-152
	Decode DECODE_DWORD	4	P2-152
	Encode ENCODE_WORD	3	P2-152
	Encode ENCODE_DWORD	4	P2-153
	Bit count BITCOUNT_WORD	3	P2-153
	Bit count BITCOUNT_DWORD	3	P2-153
	Convert string to number STR_TO_UINT	3	P2-154
	Convert number to string UINT_TO_STR	3	P2-154

Instruction Symbol	Name	No. of steps	Page
	Convert shift-JIS to string SJ_TO_STR Note: These functions are not supported in SPS.	10	P2-155
	Convert string to shift-JIS STR_TO_SJ Note: These functions are not supported in SPS.	10	P2-155
	Byte length BYTE_LEN	3	P2-156
	Dead band DBAND_INT	5	P2-156
	Dead band DBAND_DINT	5	P2-157
	Dead band DBAND_REAL	5	P2-157
	Bias BIAS_INT	5	P2-158
	Bias BIAS_DINT	5	P2-158
	Bias BIAS_REAL	5	P2-159
	Step sequence coil SC_COIL	6	P2-160
	Step sequence bit SC	6	P2-160
	32-bit addition with carry ADC	6	P2-161

Instruction Symbol	Name	No. of steps	Page
	Carry after 32-bit addition ADCO	6	P2-161
	32-bit subtraction with borrow SBB	6	P2-162
	Borrow after 32-bit subtraction SBBO	6	P2-162
	Lower-order digit in 64-bit multiplication MULL	5	P2-163
	Upper-order digit in 64-bit multiplication MULU	5	P2-163
	Lower-order digit in 64-bit division DIVL	6	P2-164
	Upper-order digit in 64-bit division DIVU	6	P2-164
	Shift left 32 bits with carry SLC	5	P2-165
	Carry after 32 bits shift left SLCO	4	P2-165
	Shift right 32 bits with carry SRC	5	P2-166
	Carry after 32 bits shift right SRCO	4	P2-166

2-5-2 Function block summary

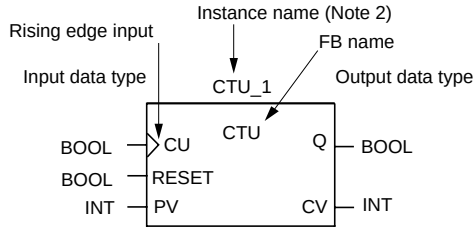
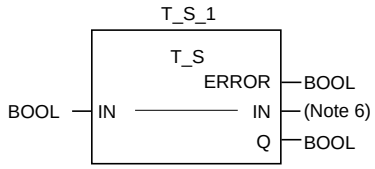
Function blocks are supported by the IL, ST, LD, and FBD languages. The operation of a function block remains the same for these languages.

The notational conventions used in the function block summary charts are almost the same as those which are used in the function summary charts except the following:

- There are instructions that have two or more output terminals.
- Edge inputs can be used.
- The number of input terminals is fixed (and cannot be extended).
- Both input and output terminals have a predefined name (parameter name).
- There are instructions that have the IN_OUT parameter.

(1) Symbols used in the function block summary

(Note 1)

Instruction Symbol	Name	No. of steps	Page
	Up counter CTU	12	P 2-○
	Test & set T_S	6	P 2-○

Note: 1) The number of steps depends on the operands used. Consequently, the number of steps of array variables will be greater than that of scalar variables.

2) Every FB uses its own memory area called an instance memory in order to operate. An FB number called an instance name is assigned to each FB. This means that the FBs with the same instance name assigned share the same instance area. Pay attention to any duplicate instance name when an FB is used.

3) The FBs provided in the MICREX-SX Series (IEC standard FBs and original FBs) are called “system Fbs,” and user-generated FBs and extended FBs supplied by Fuji Electric are “user FBs.” The system FBs use the instance memory area for user FBs and user FBs use that for user FBs.

4) Use a standard memory area not specified in the AT statement (2-word/FB instance).

5) D300win does not check the data type of the terminal associated with functions ANY or ARRAY_OF** or ARRAY_OF_*X. Use the data types appropriate for function blocks, following their detailed explanations.

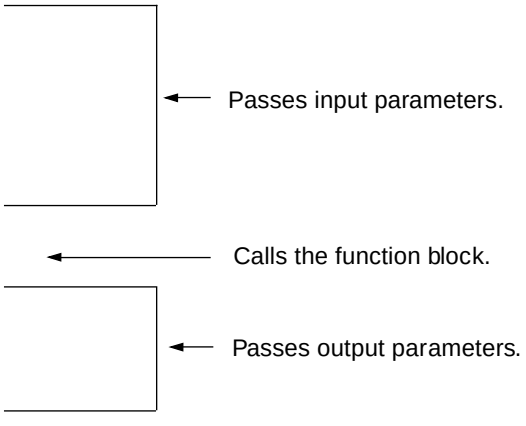
6) When FBs with IN and OUT terminals are used with SPS, it is necessary to connect same variable to both IN and OUT.

(2) Describing a function block in the IL language

When describing a function block call in the IL language, it is necessary to specify input/output parameters and a call instruction (“CAL,” “CALC,” or “CALCN”) as shown below.

<Sample program for calling an up counter FB(CTU_1)>

```
LD    INPUT1      (* Count input *)
ST    CTU_1.CU
LD    INPUT2      (* Reset input *)
ST    CTU_1.RESET
LD    SETDATA     (* Set value *)
ST    CTU_1.PV
CAL   CTU_1       (* Read CTU_1 unconditionally *)
LD    CTU_1.Q
ST    OUTFLAG     (* Up bit *)
LD    CTU_1.CV
ST    CURRENT     (* Current value *)
```



(3) Describing a function block in the ST language

Specify a function block call in the ST language as shown below.

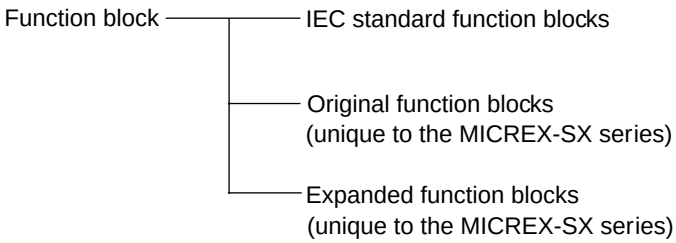
<Sample program for calling an up counter FB(CTU_1)>

```
CTU_1(CU: =INPUT1, RESET: =INPUT2, PT: =SETDATA);
OUTFLAG: =CTU_1.Q;
CURRENT: =CTU_1.CV
```

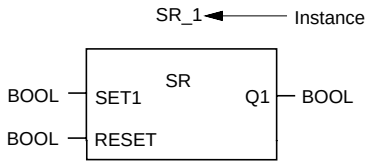
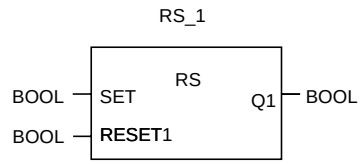
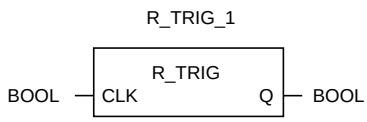
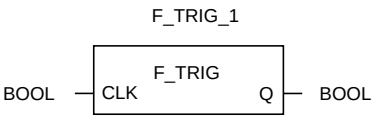
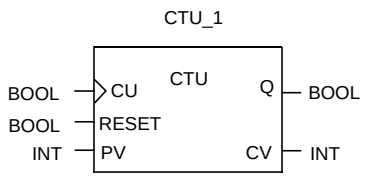
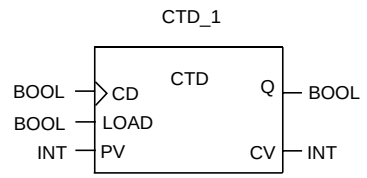
(4) Function block summary

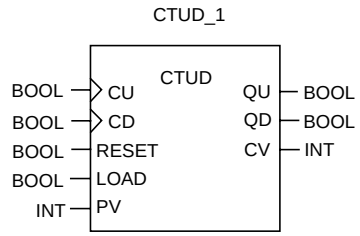
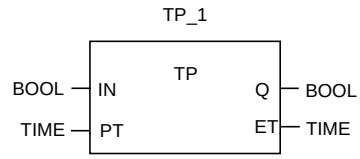
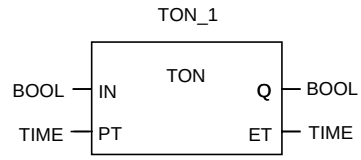
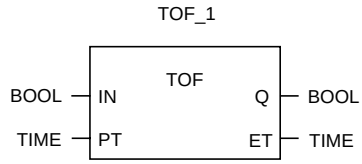
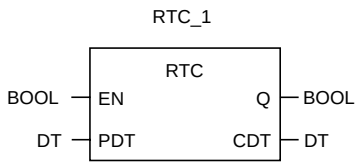
The function blocks that are supported by the MICREX-SX series are classified into the following categories:

FBs with a ◥ at an upper right corner in the name column are not supported by the standard CPUs.

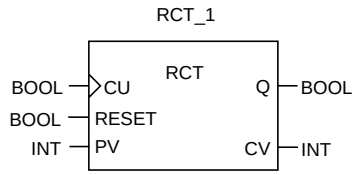
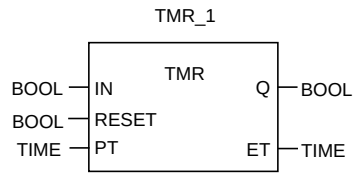
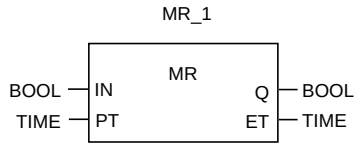
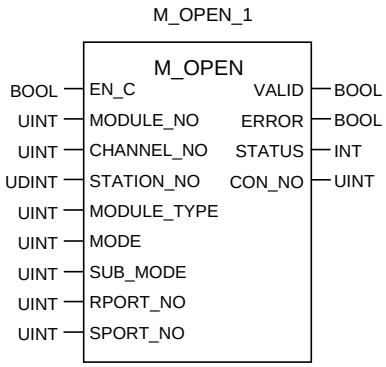
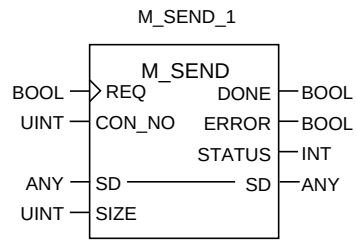
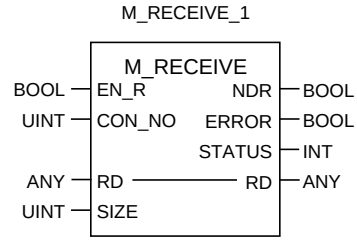


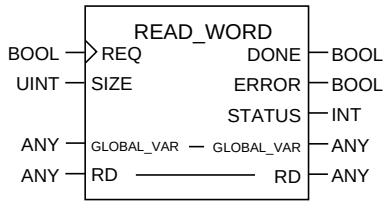
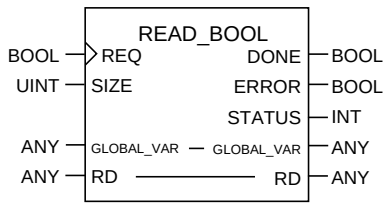
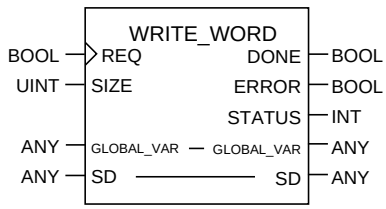
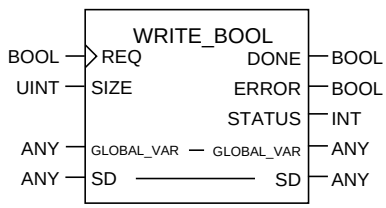
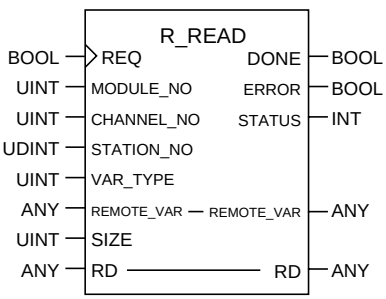
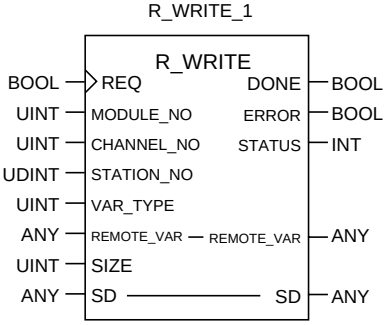
1) IEC standard function blocks

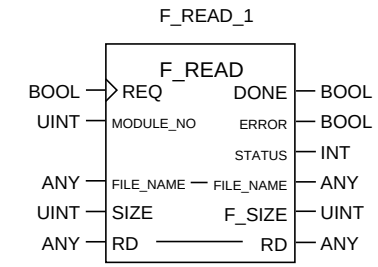
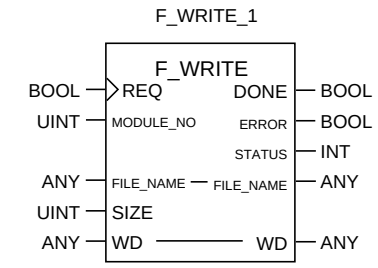
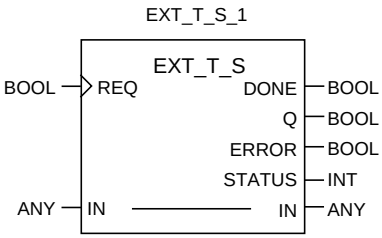
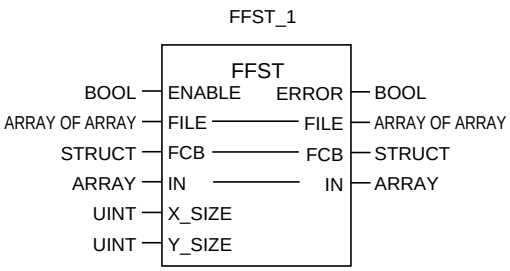
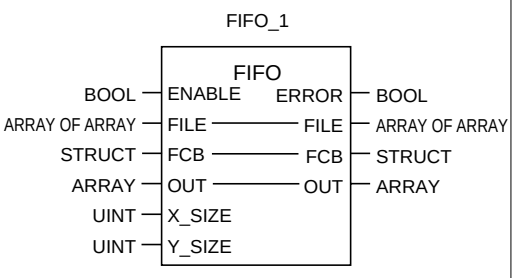
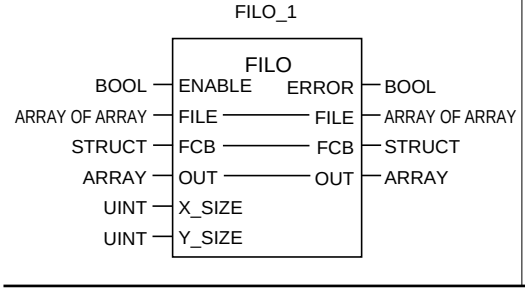
Instruction Symbol	Name	No. of steps	Page
	Set reset flip-flop SR	8	P2-167
	Reset set flip-flop RS	8	P2-167
	Rising edge detection R_TRIG	6	P2-168
	Falling edge detection F_TRIG	6	P2-168
	Up counter CTU	12	P2-169
	Down counter CTD	12	P2-169

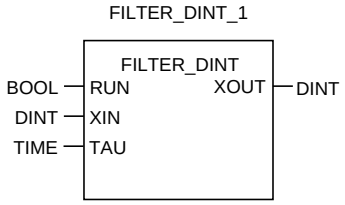
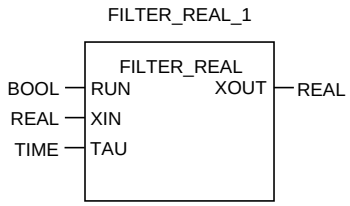
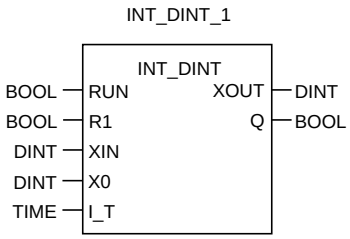
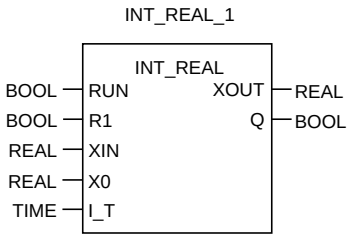
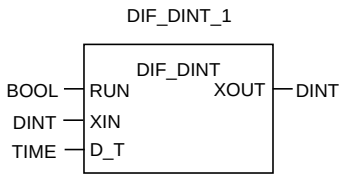
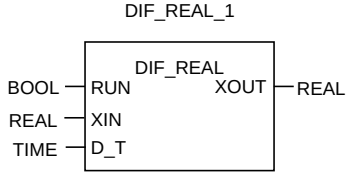
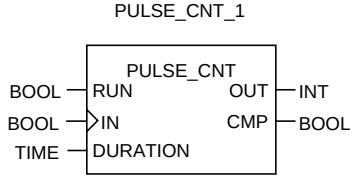
Instruction Symbol	Name	No. of steps	Page
	Up down counter CTUD	18	P2-170
	Pulse TP	10	P2-171
	On-delay timer TON	10	P2-171
	Off-delay timer TOF	10	P2-172
	Real-time clock RTC Note: These functions are not supported in SPS.	10	P2-172

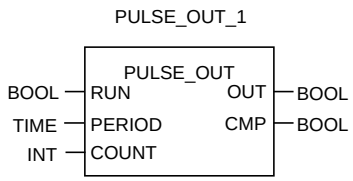
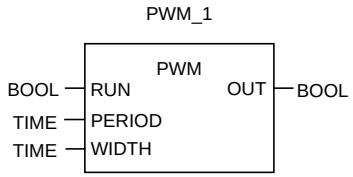
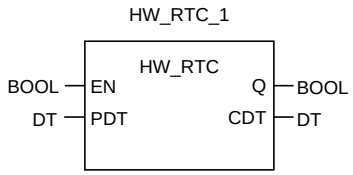
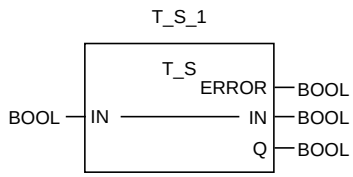
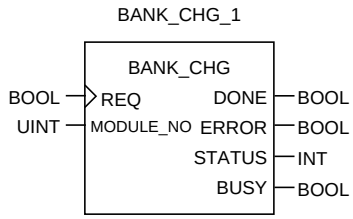
2) Original FBs (Function blocks)

Instruction Symbol	Name	No. of steps	Page
	Ring counter RCT	12	P2-173
	Integrating timer TMR	10	P2-174
	Retriggerable timer MR	10	P2-175
	Open channel M_OPEN	29	P2-176
	Send message M_SEND	16	P2-178
	Receive message M_RECEIVE	16	P2-179

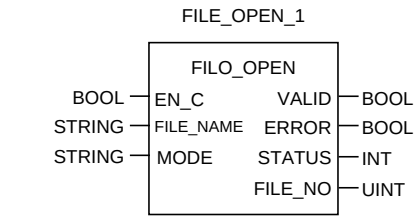
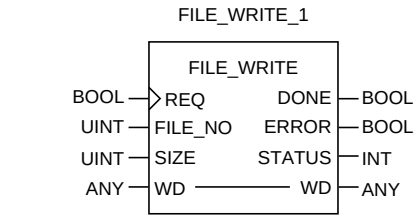
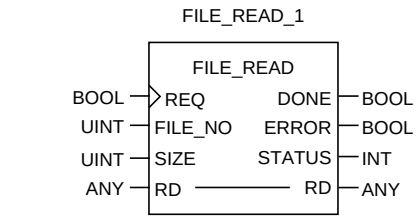
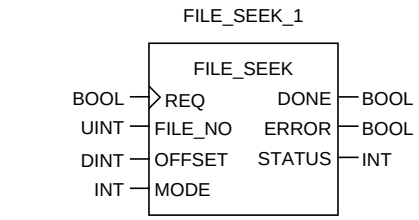
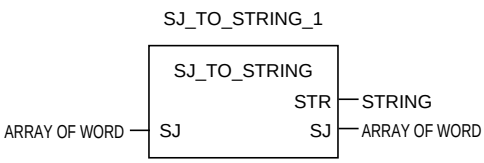
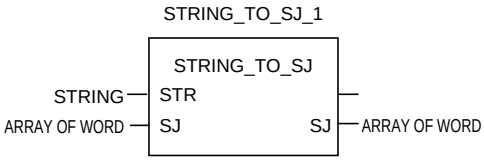
Instruction Symbol	Name	No. of steps	Page
READ_WORD_1  READ_BOOL_1  	Direct read READ_WORD Direct read READ_BOOL Note: These functions are not supported in SPS.	16 16	P2-180
WRITE_WORD_1  WRITE_BOOL_1  	Direct write READ WRITE_WORD Direct write WRITE_BOOL Note: These functions are not supported in SPS.	16 16	P2-181
R_READ_1  	Remote data read R_READ	16	P2-182
R_WRITE_1  	Remote data write R_READ	16	P2-183

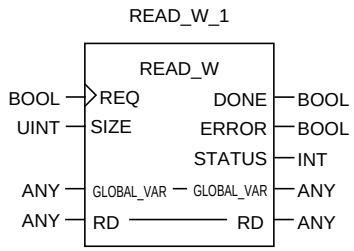
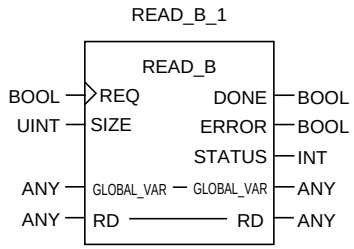
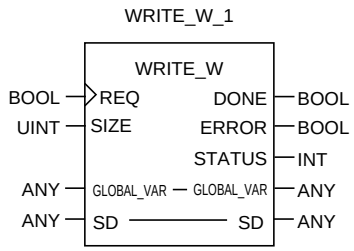
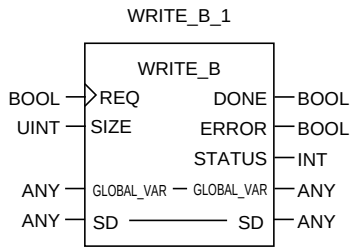
Instruction Symbol	Name	No. of steps	Page
	File data read F_READ	16	P2-186
	File data write F_WRITE	16	P2-187
	Extension test & set EXT_T_S Note: These functions are not supported in SPS.	14	P2-188
	Sequential file store FFST	12	P2-189
	Sequential file load first FIFO	12	P2-190
	Sequential file load last FILO	12	P2-191

Instruction Symbol	Name	No. of steps	Page
	Filter FILTER_DINT	10	P2-192
	Filter FILTER_REAL	10	P2-193
	Integrate INT_DINT	16	P2-194
	Integrate INT_REAL	16	P2-195
	Differentiate DIF_DINT	10	P2-196
	Differentiate DIF_REAL	10	P2-197
	Pulse count PULSE_CNT	12	P2-198

Instruction Symbol	Name	No. of steps	Page
	Pulse output PULSE_OUT	12	P2-198
	Pulse PWM	10	P2-199
	Hardware RTC (Real-time clock) HW_RTC Note: These functions are not supported in SPS.	10	P2-199
	Test & set T_S	6	P2-200
	Change bank BANK_CHG Note: These functions are not supported in SPS.	14	P2-201

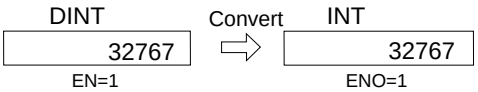
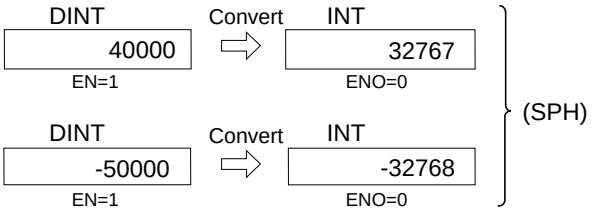
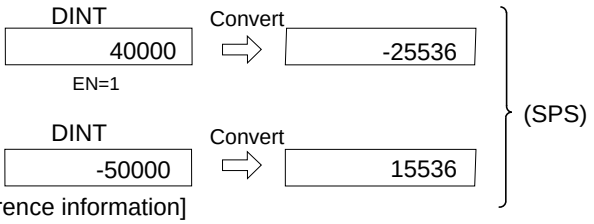
3) Only for SPS FBs (Function blocks)

Instruction Symbol	Name	Page
	File open FILE_OPEN	P2-204
	File data write FILE_WRITE	P2-205
	File data read FILE_READ	P2-206
	File pointer seek FILE_SEEK	P2-207
	Convert shift-JIS to string SJ_TO_STR	P2-208
	Convert string to shift-JIS STR_TO_SJ	P2-208

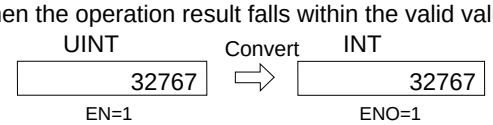
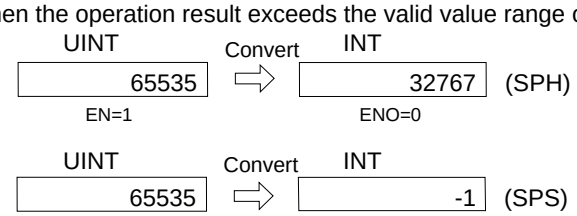
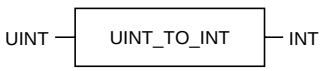
Instruction Symbol	Name	Page
	Direct read READ_W	P2-209
	Direct read READ_B	P2-210
	Direct write WRITE_W	P2-211
	Direct write WRITE_B	P2-212

2-5-3 Type conversion functions

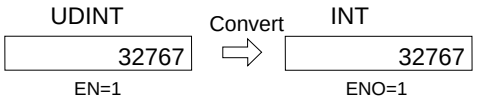
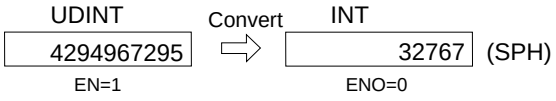
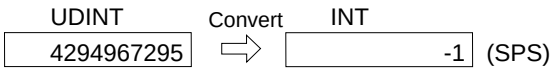
(1) Type conversion DINT_TO_INT

Name, Symbol, Function		Example
Name	DINT_TO_INT	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type   [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of INT type: -32,768 to 32,767
Symbol	 	
Function	(1) DINT_TO_INT converts DINT type data to INT type data. (2) When SPH is used, if the input DINT value exceeds the valid range of INT type after type conversion, the boundary value of INT type is output. (Boundary value: -32768, 32767) (3) When SPS is used, if the input DINT value exceeds the valid range of INT type after type conversion, no boundary processing is performed and lower 16 bits are output.	

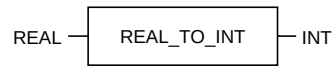
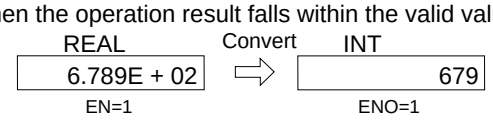
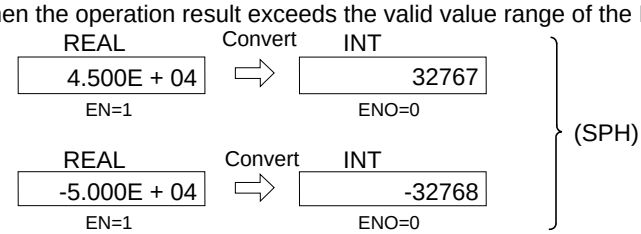
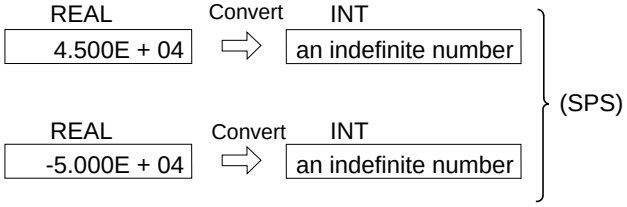
(2) Type conversion UINT_TO_INT

Name, Symbol, Function		Example
Name	UINT_TO_INT	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type  [Reference information] Value range of UINT type: 0 to 65,535 Value range of INT type: -32,768 to 32,767
Symbol	 	
Function	(1) UINT_TO_INT converts UINT type data to INT type data. (2) When SPH is used, if the input UINT value exceeds the valid range of INT type after type conversion, the upper limit value (32767) of INT type is output. (3) When SPS is used, even if the input UINT value exceeds the valid range of INT type after type conversion, no boundary processing is performed.	

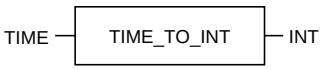
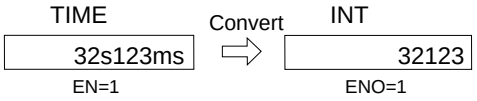
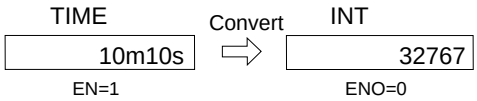
(3) Type conversion UDINT_TO_INT

Name, Symbol, Function	Example
Name UDINT_TO_INT **Symbol**  **Function** (1) UDINT_TO_INT converts UDINT type data to INT type data. (2) When SPH is used, if the input UDINT value exceeds the valid range of INT type after type conversion, the upper limit value (32767) of INT type is output. (3) When SPS is used, if the input UDINT value exceeds the valid range of INT type after type conversion, no boundary processing is performed and lower 16 bits are output.	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type   [Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of INT type: -32,768 to 32,767

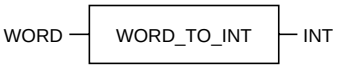
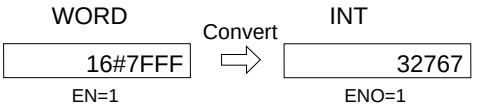
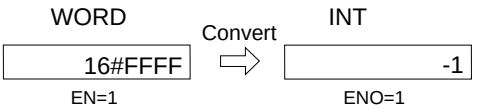
(4) Type conversion REAL_TO_INT

Name, Symbol, Function	Example
Name REAL_TO_INT **Symbol**  **Function** (1) REAL_TO_INT converts REAL type data to INT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of INT type after type conversion, the boundary value of INT type is output. (3) The fractional part is rounded off. (4) When SPS is used, if the input REAL value exceeds the valid range of INT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type   [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of INT type: -32,768 to 32,767

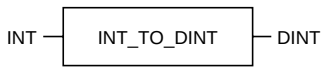
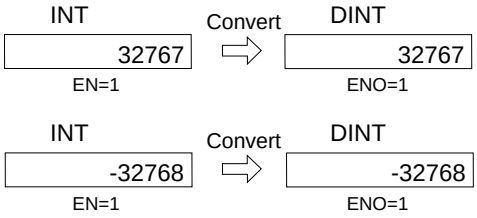
(5) Type conversion TIME_TO_INT

Name, Symbol, Function	Example
Name TIME_TO_INT **Symbol**  **Function** (1) TIME_TO_INT converts TIME type data to INT type data. (2) When SPH is used, if the input TIME value exceeds the valid range of INT type after type conversion, the upper limit value (32767) of INT type is output. (3) 1 ms is converted to 1. (4) When SPS is used, even if the input TIME value exceeds the valid range of INT type after type conversion, no boundary processing is performed.	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type  [Reference information] Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms) Value range of INT type: -32,768 to 32,767

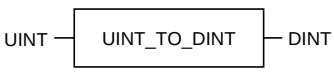
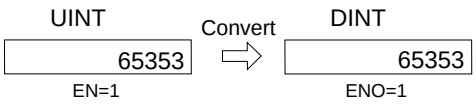
(6) Type conversion WORD_TO_INT

Name, Symbol, Function	Example
Name WORD_TO_INT **Symbol**  **Function** (1) WORD_TO_INT converts WORD type data to INT type data.	<Operation>   [Reference information] Value range of WORD type: 16#0000 to 16#FFFF Value range of INT type: -32,768 to 32,767

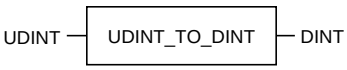
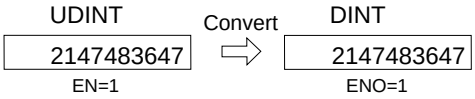
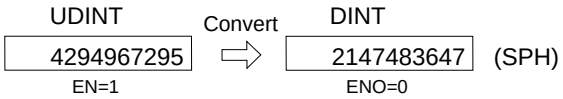
(7) Type conversion INT_TO_DINT

Name, Symbol, Function	Example
Name: INT_TO_DINT Symbol:  Function: (1) INT_TO_DINT converts INT type data to DINT type data.	<Operation>  [Reference information] Value range of INT type: -32,768 to 32,767 Value range of DINT type: -2,147,483,648 to 2,147,483,647

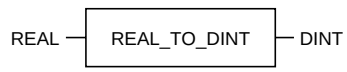
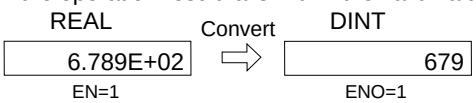
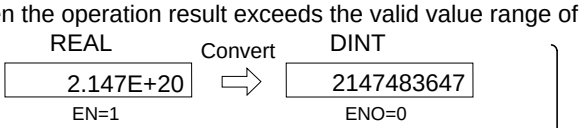
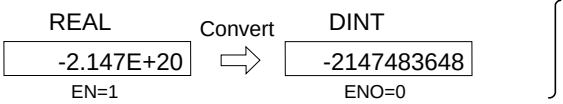
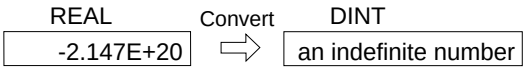
(8) Type conversion UINT_TO_DINT

Name, Symbol, Function	Example
Name: UINT_TO_DINT Symbol:  Function: (1) UINT_TO_DINT converts UINT type data to DINT type data.	<Operation>  [Reference information] Value range of UINT type: 0 to 65,535 Value range of DINT type: -2,147,483,648 to 2,147,483,647

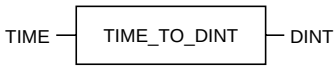
(9) Type conversion UDINT_TO_DINT

Name, Symbol, Function	Example
Name UDINT_TO_DINT **Symbol**  **Function** (1) UDINT_TO_DINT converts UDINT type data to DINT type data. (2) When SPH is used, if the input UDINT value exceeds the valid range of DINT type after type conversion, the upper limit value (2147483647) of DINT type is output. (3) When SPS is used, even if the input UDINT value exceeds the valid range of DINT type after type conversion, no boundary processing is performed.	<Operation> • When the operation result falls within the valid value range of the DINT type  • When the operation result exceeds the valid value range of the DINT type  (SPH)  (SPS) [Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of DINT type: -2,147,483,648 to 2,147,483,647

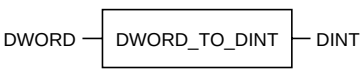
(10) Type conversion REAL_TO_DINT

Name, Symbol, Function	Example
Name REAL_TO_DINT **Symbol**  **Function** (1) REAL_TO_DINT converts REAL type data to DINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of DINT type after type conversion, the boundary value of DINT type is output. (Boundary value: -2147483648, 2147483647) (3) The fractional part is rounded off. (4) The number of significant digits after conversion is 6 digits. (5) When SPS is used, if the input REAL value exceeds the valid range of DINT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the DINT type  • When the operation result exceeds the valid value range of the DINT type   } (SPH)  (SPS) [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of DINT type: -2,147,483,648 to 2,147,483,647

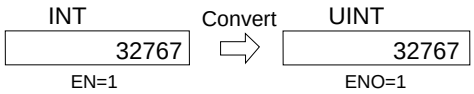
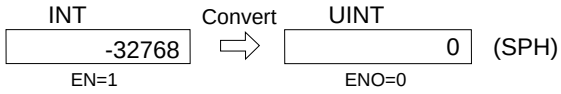
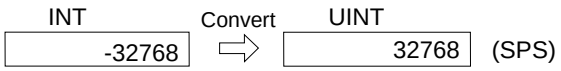
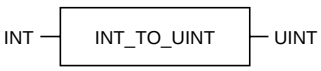
(11) Type conversion TIME_TO_DINT

Name, Symbol, Function		Example
Name	TIME_TO_DINT	<Operation> • When the operation result falls within the valid value range of the DINT type TIME 17h20m10s (EN=1) Convert ⇨ DINT 62410000 (ENO=1) • When the operation result exceeds the valid value range of the DINT type TIME 25d20h31m23s (EN=1) Convert ⇨ DINT 2147483647 (ENO=0) (SPH) TIME 25d20h31m23s Convert ⇨ DINT -2061084296 (SPS) [Reference information] Value range of TIME type: 0 to 4,294,967,295 (ms) Value range of DINT type: -2,147,483,648 to 2,147,483,647
Symbol		
Function	(1) TIME_TO_DINT converts TIME type data to DINT type data. (2) When SPH is used, if the input TIME value exceeds the valid range of DINT type after type conversion, the upper limit value (2147483647) of DINT type is output. (3) 1 ms is converted to 1. (4) When SPS is used, even if the input TIME value exceeds the valid range of DINT type after type conversion, no boundary processing is performed.	

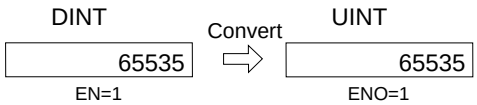
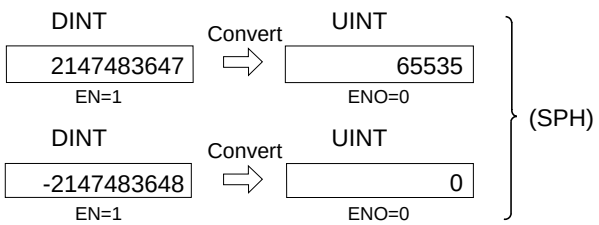
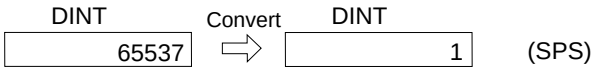
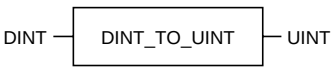
(12) Type conversion DWORD_TO_DINT

Name, Symbol, Function		Example
Name	DWORD_TO_DINT	<Operation> DWORD 16#7FFFFFFF (EN=1) Convert ⇨ DINT 2147483647 (ENO=1) DWORD 16#FFFFFFFF (EN=1) Convert ⇨ DINT -1 (ENO=1) [Reference information] Value range of DWORD type: 16#00000000 to 16#FFFFFFFF Value range of DINT type: -2,147,483,648 to 2,147,483,647
Symbol		
Function	(1) DWORD_TO_DINT converts DWORD type data to DINT type data.	

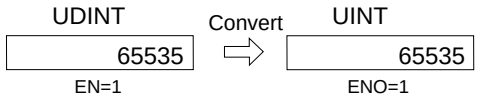
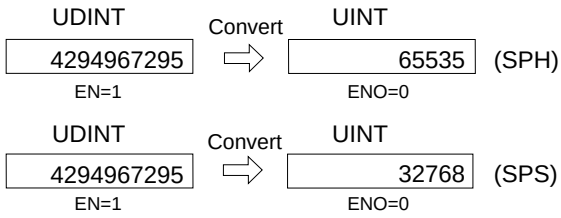
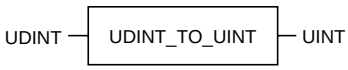
(13) Type conversion INT_TO_UINT

Name, Symbol, Function		Example
Name	INT_TO_UINT	<Operation> • When the operation result falls within the valid value range of the UINT type  • When the operation result exceeds the valid value range of the UINT type  (SPH)  (SPS) [Reference information] Value range of INT type: -32,768 to 32,767 Value range of UINT type: 0 to 65,535
Symbol	 	
Function	(1) INT_TO_UINT converts INT type data to UINT type data. (2) When SPH is used, if the input INT value exceeds the valid range of UINT type after type conversion, the lower limit value (UINT#0) of UINT type is output. (3) When SPS is used, even if the input INT value exceeds the valid range of UINT type after type conversion, no boundary processing is performed.	

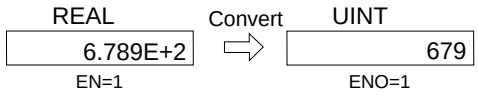
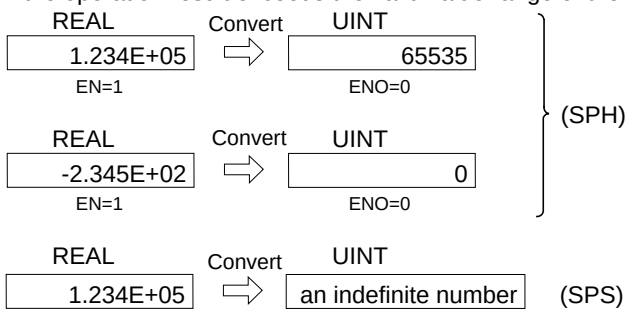
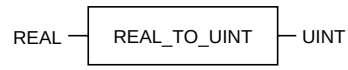
(14) Type conversion DINT_TO_UINT

Name, Symbol, Function		Example
Name	DINT_TO_UINT	<Operation> • When the operation result falls within the valid value range of the UINT type  • When the operation result exceeds the valid value range of the UINT type  (SPH)  (SPS) [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of UINT type: 0 to 65,535
Symbol	 	
Function	(1) DINT_TO_UINT converts DINT type data to UINT type data. (2) When SPH is used, if the input DINT value exceeds the valid range of UINT type after type conversion, the boundary value of UINT type is output. (Boundary value: 0, 65535) (3) When SPS is used, if the input DINT value exceeds the valid range of UINT type after type conversion, no boundary processing is performed and lower 16 bits are output.	

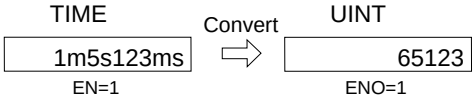
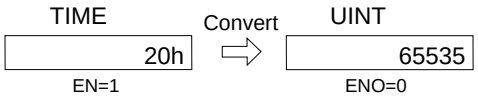
(15) Type conversion UDINT_TO_UINT

Name, Symbol, Function		Example
Name	UDINT_TO_UINT	<Operation> • When the operation result falls within the valid value range of the UINT type  • When the operation result exceeds the valid value range of the UINT type  [Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of UINT type: 0 to 65,535
Symbol	 	
Function	(1) UDINT_TO_UINT converts UDINT type data to UINT type data. (2) When SPH is used, if the input UDINT value exceeds the valid range of UINT type after type conversion, the upper limit value (65535) of UINT type is output. (3) When SPS is used, if the input UDINT value exceeds the valid range of UINT type after type conversion, no boundary processing is performed and lower 16 bits are output.	

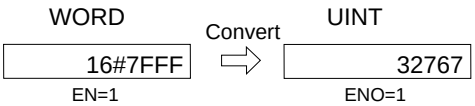
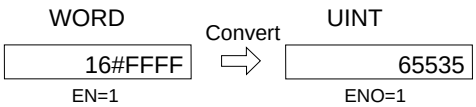
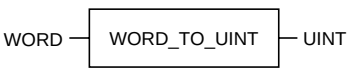
(16) Type conversion REAL_TO_UINT

Name, Symbol, Function		Example
Name	REAL_TO_UINT	<Operation> • When the operation result falls within the valid value range of the UINT type  • When the operation result exceeds the valid value range of the UINT type  [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of UINT type: 0 to 65,535
Symbol	 	
Function	(1) REAL_TO_UINT converts REAL type data to UINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of UINT type after type conversion, the boundary value of UINT type is output. (3) The fractional part is rounded off. (4) When SPS is used, if the input REAL value exceeds the valid range of UINT type after type conversion, an indefinite number is output.	

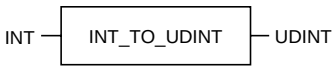
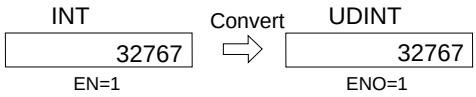
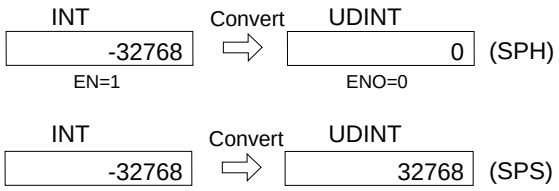
(17) Type conversion TIME_TO_UINT

Name, Symbol, Function	Example
Name TIME_TO_UINT	<Operation> • When the operation result falls within the valid value range of the UINT type
Symbol 	 • When the operation result exceeds the valid value range of the UINT type 
Function (1) TIME_TO_UINT converts TIME type data to UINT type data. (2) When SPH is used, if the input TIME value exceeds the valid range of UINT type after type conversion, the upper limit value (65535) of UINT type is output. (3) 1 ms is converted to 1. (4) When SPS is used, even if the input TIME value exceeds the valid range of UINT type after type conversion, no boundary processing is performed.	[Reference information] Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms) Value range of UINT type: 0 to 65,535

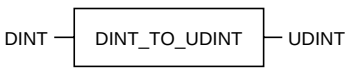
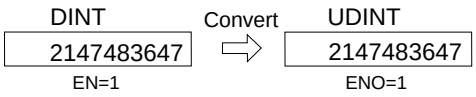
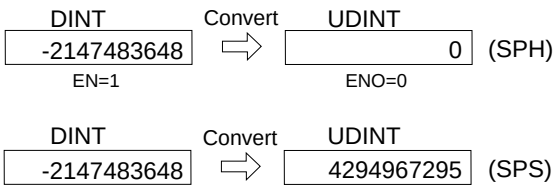
(18) Type conversion WORD_TO_UINT

Name, Symbol, Function	Example
Name WORD_TO_UINT	<Operation>  
Symbol 	
Function (1) WORD_TO_UINT converts WORD type data to UINT type data.	[Reference information] Value range of WORD type: 16#0000 to 16#FFFF Value range of UINT type: 0 to 65,535

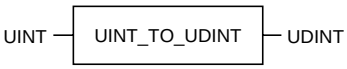
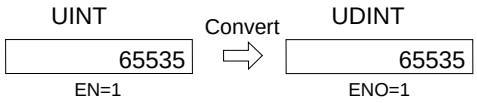
(19) Type conversion INT_TO_UDINT

Name, Symbol, Function	Example
Name INT_TO_UDINT **Symbol**  **Function** 1) INT_TO_UDINT converts INT type data to UDINT type data. (2) When SPH is used, if the input INT value exceeds the valid range of UDINT type after type conversion, the lower limit value (UDINT#0) of UDINT type is output. (3) When SPS is used, even if the input INT value exceeds the valid range of UDINT type after type conversion, no boundary processing is performed.	<Operation> • When the operation result falls within the valid value range of the UDINT type  • When the operation result exceeds the valid value range of the UDINT type  [Reference information] Value range of INT type: -32,768 to 32,767 Value range of UDINT type: 0 to 4,294,967,295

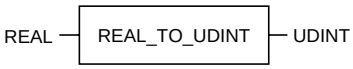
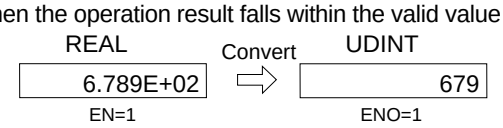
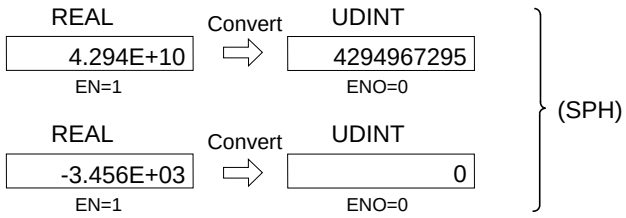
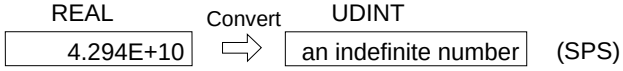
(20) Type conversion DINT_TO_UDINT

Name, Symbol, Function	Example
Name DINT_TO_UDINT **Symbol**  **Function** (1) DINT_TO_UDINT converts DINT type data to UDINT type data. (2) When SPH is used, if the input DINT value exceeds the valid range of UDINT type after type conversion, the lower limit value (UDINT#0) of UDINT type is output. (3) When SPS is used, even if the input DINT value exceeds the valid range of UDINT type after type conversion, no boundary processing is performed.	<Operation> • When the operation result falls within the valid value range of the UDINT type  • When the operation result exceeds the valid value range of the UDINT type  [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of UDINT type: 0 to 4,294,967,295

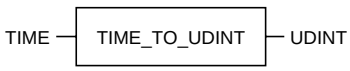
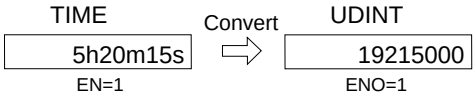
(21) Type conversion UINT_TO_UDINT

Name, Symbol, Function	Example
Name: UINT_TO_UDINT Symbol:  Function: (1) UINT_TO_UDINT converts UINT type data to UDINT type data.	<Operation>  [Reference information] Value range of UINT type: 0 to 65,535 Value range of UDINT type: 0 to 4,294,967,295

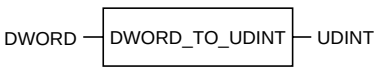
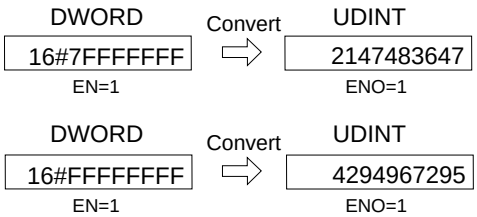
(22) Type conversion REAL_TO_UDINT

Name, Symbol, Function	Example
Name: REAL_TO_UDINT Symbol:  Function: (1) REAL_TO_UDINT converts REAL type data to UDINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of UDINT type after type conversion, the boundary value of UDINT type is output. (3) The fractional part is rounded off. (4) The number of significant digits after conversion is 6 digits. (5) When SPS is used, if the input REAL value exceeds the valid range of UDINT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the UDINT type  • When the operation result exceeds the valid value range of the UDINT type  (SPH)  (SPS) [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of UDINT type: 0 to 4,294,967,295

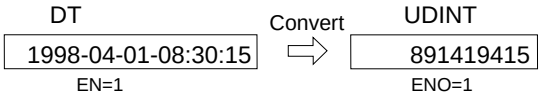
(23) Type conversion TIME_TO_UDINT

Name, Symbol, Function	Example
Name: TIME_TO_UDINT Symbol:  Function: (1) TIME_TO_UDINT converts TIME type data to UDINT type data. (2) 1 ms is converted to 1.	<Operation>  [Reference information] Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms) Value range of UDINT type: 0 to 4,294,967,295

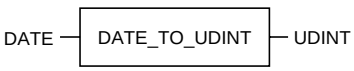
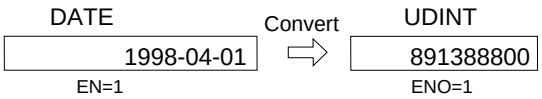
(24) Type conversion DWORD_TO_UDINT

Name, Symbol, Function	Example
Name: DWORD_TO_UDINT Symbol:  Function: (1) DWORD_TO_UDINT converts DWORD type data to UDINT type data.	<Operation>  [Reference information] Value range of DWORD type: 16#00000000 to 16#FFFFFFFF Value range of UDINT type: 0 to 4,294,967,295

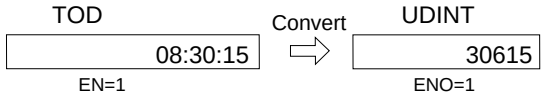
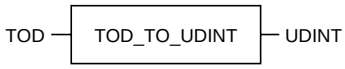
(25) Type conversion DT_TO_UDINT (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	DT_TO_UDINT	<Operation>
Symbol		
Function	(1) DT_TO_UDINT converts DT type data to UDINT type data. (2) The time is measured from 0:00:00 January 1st, 1970 and is specified in seconds.	

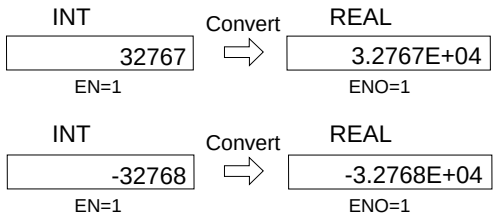
(26) Type conversion DATE_TO_UDINT (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	DATE_TO_UDINT	<Operation>
Symbol		
Function	(1) DATE_TO_UDINT converts DATE type data to UDINT type data. (2) The time is measured from 0:00:00 January 1st, 1970 and is specified in seconds (an integral multiple of 86400 seconds = 1 day).	

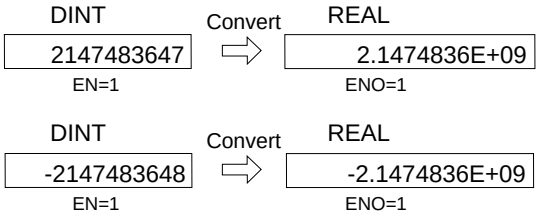
(27) Type conversion TOD_TO_UDINT (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	TOD_TO_UDINT	<Operation> 
Symbol		
Function	(1) TOD_TO_UDINT converts TOD type data to UDINT type data. (2) The time starts from 0:00:00 and is specified in seconds.	

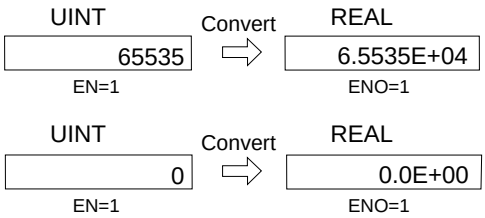
(28) Type conversion INT_TO_REAL

Name, Symbol, Function		Example
Name	INT_TO_REAL	<Operation> 
Symbol		
Function	(1) INT_TO_REAL converts INT type data to REAL type data. (2) The number of significant digits after conversion is 6 digits.	[Reference information] Value range of INT type: -32,768 to 32,767 Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$

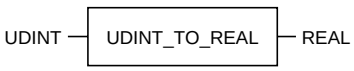
(29) Type conversion DINT_TO_REAL

Name, Symbol, Function	Example
Name: DINT_TO_REAL Symbol: 	<Operation> 
Function: (1) DINT_TO_REAL converts DINT type data to REAL type data. (2) The number of significant digits after conversion is 6 digits.	[Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$

(30) Type conversion UINT_TO_REAL

Name, Symbol, Function	Example
Name: UINT_TO_REAL Symbol: 	<Operation> 
Function: (1) UINT_TO_REAL converts UINT type data to REAL type data. (2) The number of significant digits after conversion is 6 digits.	[Reference information] Value range of UINT type: 0 to 65,535 Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$

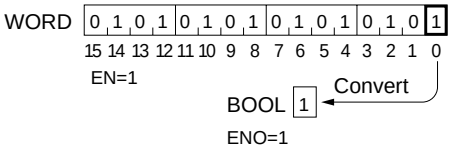
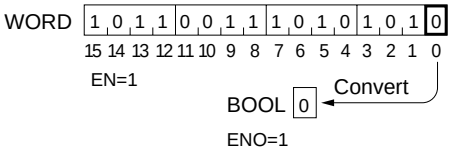
(31) Type conversion UDINT_TO_REAL

Name, Symbol, Function		Example
Name	UDINT_TO_REAL	<Operation> UDINT 4294967295 Convert REAL 4.2949673E+09 EN=1 ENO=1 UDINT 0 Convert REAL 0.0E+00 EN=1 ENO=1
Symbol		
Function	(1) UDINT_TO_REAL converts UDINT type data to REAL type data. (2) The number of significant digits after conversion is 6 digits.	[Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$

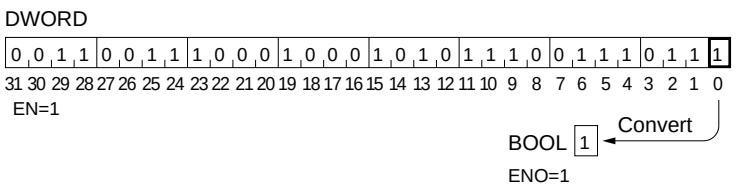
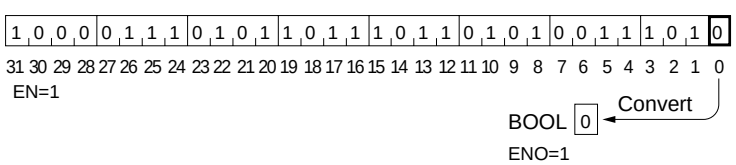
(32) Type conversion TIME_TO_REAL

Name, Symbol, Function		Example
Name	TIME_TO_REAL	<Operation> TIME 5h20m15s Convert REAL 1.92150E+07 EN=1 ENO=1
Symbol		
Function	(1) TIME_TO_REAL converts TIME type data to REAL type data. (2) 1 ms is converted to 1. (3) The number of significant digits after conversion is 6 digits.	[Reference information] Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms) Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$

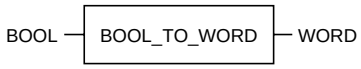
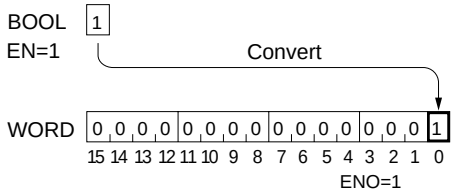
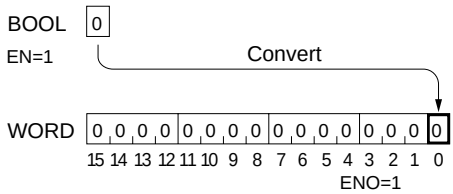
(33) Type conversion WORD_TO_BOOL

Name, Symbol, Function	Example
Name: WORD_TO_BOOL	<Operation> • When bit 0 of WORD type data is “1”
Symbol: WORD — WORD_TO_BOOL — BOOL	
Function: (1) WORD_TO_BOOL converts bit 0 of WORD type data to BOOL type data.	• When bit 0 of WORD type data is “0” 

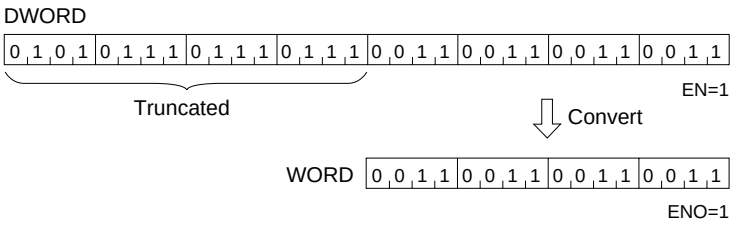
(34) Type conversion DWORD_TO_BOOL

Name, Symbol, Function	Example
Name: DWORD_TO_BOOL	<Operation> • When bit 0 of DWORD type data is “1”
Symbol: DWORD — DWORD_TO_BOOL — BOOL	
Function: (1) DWORD_TO_BOOL converts bit 0 of DWORD type data to BOOL type data.	• When bit 0 of DWORD type data is “0” 

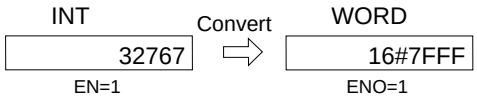
(35) Type conversion BOOL_TO_WORD

Name, Symbol, Function	Example
Name: BOOL_TO_WORD Symbol:  Function: (1) BOOL_TO_WORD converts BOOL type data to WORD type data. Bit 0 of the WORD data is loaded with the BOOL value and the other bits with a "0."	<Operation> • When BOOL data is "1"  • When BOOL data is "0" 

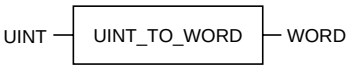
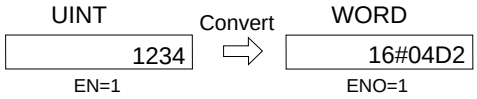
(36) Type conversion DWORD_TO_WORD

Name, Symbol, Function	Example
Name: DWORD_TO_WORD Symbol:  Function: (1) DWORD_TO_WORD converts DWORD type data to WORD type data. The lower-order 16 bits of the DWORD value are output.	<Operation>  [Reference information] Value range of DWORD type: 16#00000000 to 16#FFFFFFFF Value range of WORD type: 16#0000 to 16#FFFF

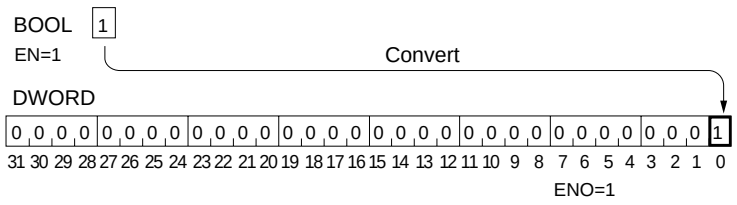
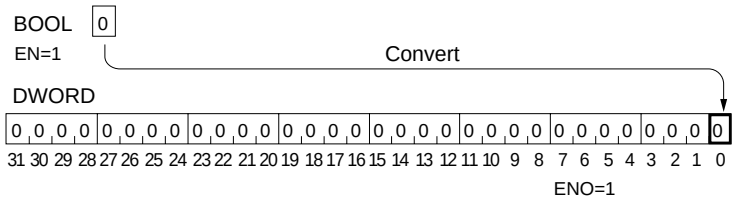
(37) Type conversion INT_TO_WORD

Name, Symbol, Function	Example
Name: INT_TO_WORD Symbol:  Function: (1) INT_TO_WORD converts INT type data to WORD type data.	<Operation>  [Reference information] Value range of INT type: -32,768 to 32,767 Value range of WORD type: 16#0000 to 16#FFFF

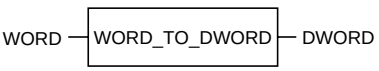
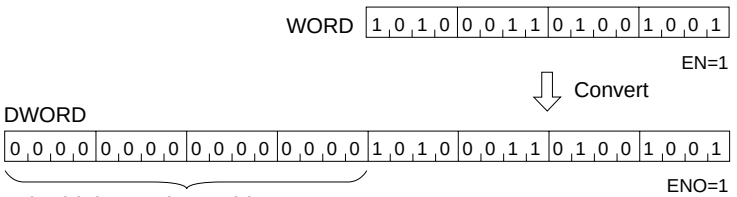
(38) Type conversion UINT_TO_WORD

Name, Symbol, Function	Example
Name: UINT_TO_WORD Symbol:  Function: (1) UINT_TO_WORD converts UINT type data to WORD type data.	<Operation>  [Reference information] Value range of UINT type: 0 to 65,535 Value range of WORD type: 16#0000 to 16#FFFF

(39) Type conversion BOOL_TO_DWORD

Name, Symbol, Function		Example
Name	BOOL_TO_DWORD	<Operation> • When BOOL data is “1”
Symbol		
Function	(1) BOOL_TO_DWORD converts BOOL type data to DWORD type data. Bit 0 of the DWORD type data is loaded with the BOOL value and the other bits with a “0.”	• When BOOL data is “0” 

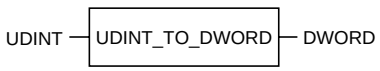
(40) Type conversion WORD_TO_DWORD

Name, Symbol, Function		Example
Name	WORD_TO_DWORD	<Operation>
Symbol		
Function	(1) WORD_TO_DWORD converts WORD type data to DWORD type data. The lower-order 16 bits of DWORD type data are loaded with the WORD type data and the higher-order 16 bits with a “0.”	[Reference information] Value range of WORD type: 16#0000 to 16#FFFF Value range of DWORD type: 16#00000000 to 16#FFFFFFFF

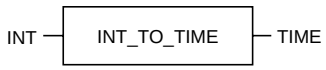
(41) Type conversion DINT_TO_DWORD

Name, Symbol, Function		Example
Name	DINT_TO_DWORD	<Operation> DINT 2147483647 (EN=1) Convert DWORD 16#7FFFFFFF (ENO=1) [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of DWORD type: 16#00000000 to 16#FFFFFFFF
Symbol		
Function	(1) DINT_TO_DWORD converts DINT type data to DWORD type data.	

(42) Type conversion UDINT_TO_DWORD

Name, Symbol, Function		Example
Name	UDINT_TO_DWORD	<Operation> UDINT 4294967295 (EN=1) Convert DWORD 16#FFFFFFFF (ENO=1) [Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of DWORD type: 16#00000000 to 16#FFFFFFFF
Symbol		
Function	(1) UDINT_TO_DWORD converts UDINT type data to DWORD type data.	

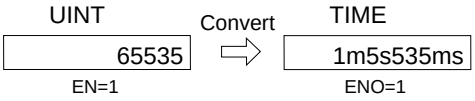
(43) Type conversion INT_TO_TIME

Name, Symbol, Function		Example
Name	INT_TO_TIME	<Operation> • When the operation result falls within the valid value range of the TIME type INT 32767 (EN=1) Convert ⇨ TIME 32s767ms (ENO=1) • When the operation result exceeds the valid value range of the TIME type INT -32768 (EN=1) Convert ⇨ TIME 0ms (ENO=0)
Symbol		
Function	(1) INT_TO_TIME converts INT type data to TIME type data. (2) When SPH is used, if the input INT value exceeds the valid range of TIME type after type conversion, the lower limit value (0ms) of TIME type is output. (3) 1 is converted to 1ms. (4) When SPS is used, even if the input INT value exceeds the valid range of TIME type after type conversion, no boundary processing is performed.	[Reference information] Value range of INT type: -32,768 to 32,767 Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms)

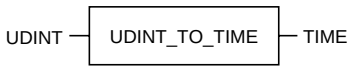
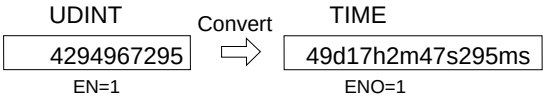
(44) Type conversion DINT_TO_TIME

Name, Symbol, Function		Example
Name	DINT_TO_TIME	<Operation> • When the operation result falls within the valid value range of the TIME type DINT 2147483647 (EN=1) Convert ⇨ TIME 24d20h31m23s647ms (ENO=1) • When the operation result exceeds the valid value range of the TIME type DINT -2147483648 (EN=1) Convert ⇨ TIME 0ms (ENO=0) (SPH) DINT -2147483648 Convert ⇨ TIME 2147483648 (SPS)
Symbol		
Function	(1) DINT_TO_TIME converts DINT type data to TIME type data. (2) When SPH is used, if the input DINT value exceeds the valid range of TIME type after type conversion, the lower limit value (0ms) of TIME type is output. (3) 1 is converted to 1ms. (4) When SPS is used, even if the input DINT value exceeds the valid range of TIME type after type conversion, no boundary processing is performed.	[Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms)

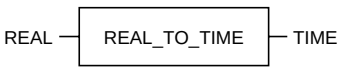
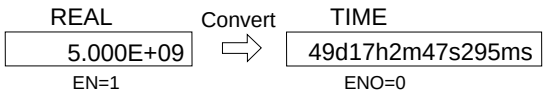
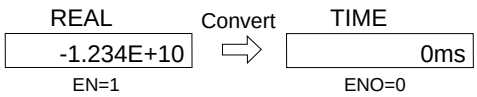
(45) Type conversion UINT_TO_TIME

Name, Symbol, Function	Example
Name: UINT_TO_TIME Symbol:  Function: (1) UINT_TO_TIME converts UINT type data to TIME type data. (2) 1 is converted to 1ms.	<Operation>  [Reference information] Value range of UINT type: 0 to 65,535 Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms)

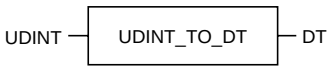
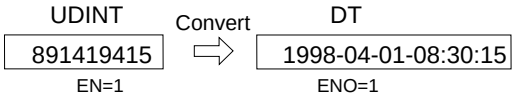
(46) Type conversion UDINT_TO_TIME

Name, Symbol, Function	Example
Name: UDINT_TO_TIME Symbol:  Function: (1) UDINT_TO_TIME converts UDINT type data to TIME type data. (2) 1 is converted to 1ms.	<Operation>  [Reference information] Value range of UDINT type: 0 to 4,294,967,295 Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms)

(47) Type conversion REAL_TO_TIME

Name, Symbol, Function	Example
Name REAL_TO_TIME **Symbol**  **Function** (1) REAL_TO_TIME converts REAL type data to TIME type data. (2) The appropriate limit value of the TIME type is generated if the result of converting the REAL value exceeds the valid value range of the TIME type. (3) The fractional part is rounded off. (4) 1 is converted to 1 ms. (5) The converted value contains some degree of error because the number of significant digits of the REAL type data is 6 digits.	<Operation> • When the operation result falls within the valid value range of the TIME type  • When the operation result exceeds the valid value range of the TIME type   [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of TIME type (SPH): 0 to 4,294,967,295 (ms) (SPS): 0 to 2,147,483,647 (ms)

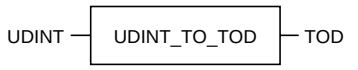
(48) Type conversion UDINT_TO_DT (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name UDINT_TO_DT **Symbol**  **Function** (1) UDINT_TO_DT converts UDINT type data to DT type data. (2) The UDINT type data is converted to DT type data which starts from 0:00:00 January 1st, 1970 and is specified in seconds.	<Operation> 

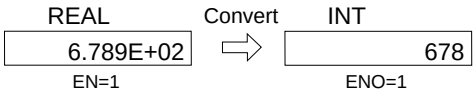
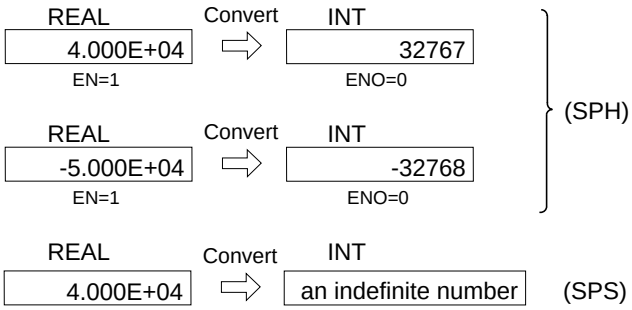
(49) Type conversion UDINT_TO_DATE (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	UDINT_TO_DATE	<Operation> UDINT 891419415 Convert DATE 1998-04-01 EN=1 ENO=1
Symbol		
Function	(1) UDINT_TO_DATE converts UDINT type data to DATE type data. (2) The UDINT type data is converted to DATE type data which starts from January 1st, 1970 and is specified in seconds (an integral multiple of 86400 seconds = 1 day). The time part of the DATE type data is truncated.	

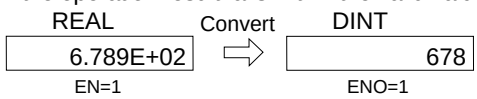
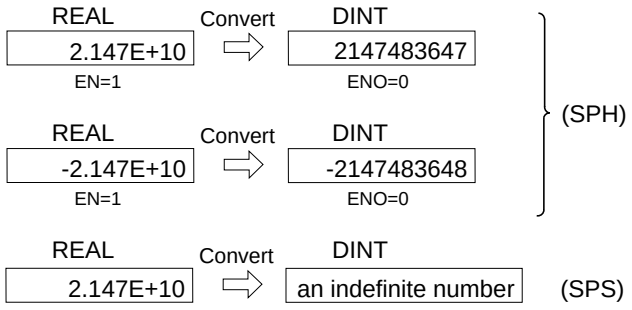
(50) Type conversion UDINT_TO_TOD (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	UDINT_TO_TOD	<Operation> UDINT 891419415 Convert TOD 08:30:15 EN=1 ENO=1
Symbol		
Function	(1) UDINT_TO_TOD converts UDINT type data to TOD type data. (2) The UDINT type data is converted to TOD type data which starts from 00:00:00 and is specified in seconds (the part exceeding 24:00:00 is truncated).	

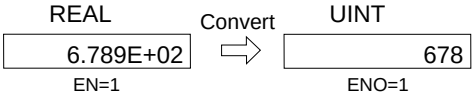
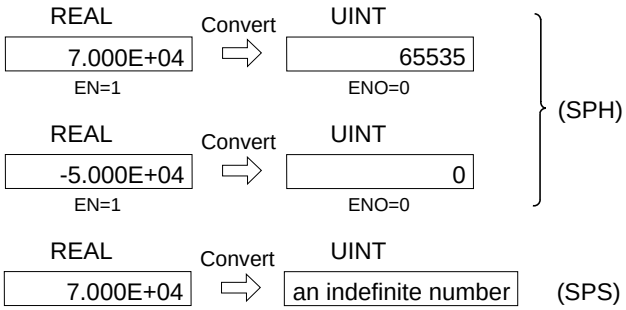
(51) Type conversion TRUNC_INT

Name, Symbol, Function	Example
Name TRUNC_INT **Symbol**  **Function** (1) TRUNC_INT converts REAL type data to INT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of INT type after type conversion, the boundary value of INT type is output. (3) The fractional part is truncated. (4) When SPS is used, if the input REAL value exceeds the valid range of INT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type  [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of INT type: -32,768 to 32,767

(52) Type conversion TRUNC_DINT

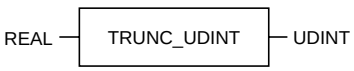
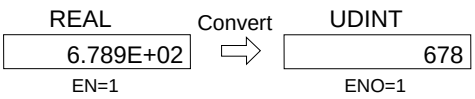
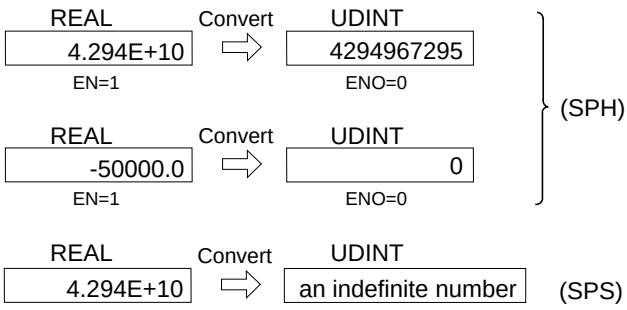
Name, Symbol, Function	Example
Name TRUNC_DINT **Symbol**  **Function** (1) TRUNC_DINT converts REAL type data to DINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of DINT type after type conversion, the boundary value of DINT type is output. (3) The fractional part is truncated. (4) The number of significant digits after conversion is 6 digits. (5) When SPS is used, if the input REAL value exceeds the valid range of DINT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the DINT type  • When the operation result exceeds the valid value range of the DINT type  [Reference information] Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$ Value range of DINT type: -2,147,483,648 to 2,147,483,647

(53) Type conversion TRUNC_UINT

Name, Symbol, Function	Example
Name: TRUNC_UINT Symbol:  Function: (1) TRUNC_UINT converts REAL type data to UINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of UINT type after type conversion, the boundary value of UINT type is output. (Boundary value: 0, 65535) (3) The fractional part is truncated. (4) When SPS is used, if the input REAL value exceeds the valid range of UINT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the UINT type  • When the operation result exceeds the valid value range of the UINT type 

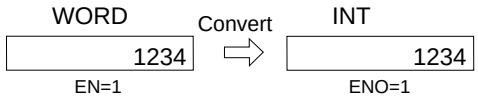
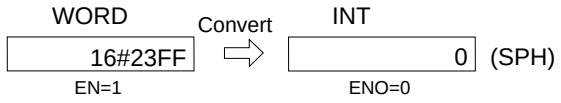
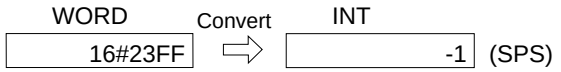
[Reference information]
Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$
Value range of UINT type: 0 to 65,535

(54) Type conversion TRUNC_UDINT

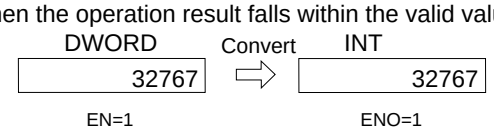
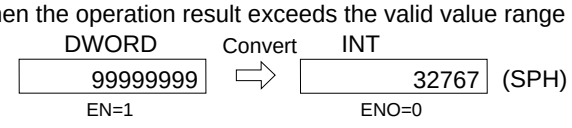
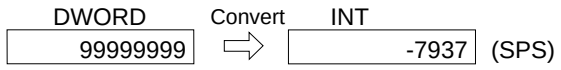
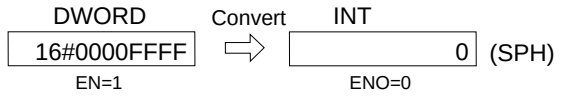
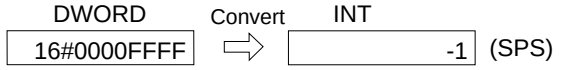
Name, Symbol, Function	Example
Name: TRUNC_UDINT Symbol:  Function: (1) TRUNC_UDINT converts REAL type data to UDINT type data. (2) When SPH is used, if the input REAL value exceeds the valid range of UDINT type after type conversion, the boundary value of UDINT type is output. (Boundary value: 0, 4294967295) (3) The fractional part is truncated. (4) The number of significant digits after conversion is 6 digits. (5) When SPS is used, if the input REAL value exceeds the valid range of UDINT type after type conversion, an indefinite number is output.	<Operation> • When the operation result falls within the valid value range of the UDINT type  • When the operation result exceeds the valid value range of the UDINT type 

[Reference information]
Value range of REAL type: $-2^{128} < N \leq -2^{-126}$, 0, $2^{-126} \leq N < 2^{128}$
Value range of UDINT type: 0 to 4,294,967,295

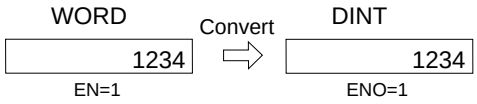
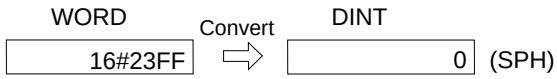
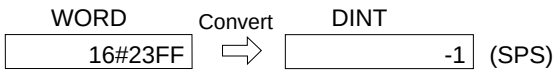
(55) Type conversion W_BCD_TO_INT

Name, Symbol, Function	Example
Name W_BCD_TO_INT **Symbol**  **Function** (1) W_BCD_TO_INT regards WORD type data as BCD code and converts it to INT type data. (2) When SPH is used, if the input WORD value is other than BCD code, “0” is output. (3) When SPS is used, if the input WORD value is other than BCD code, “-1” is output.	<Operation> • When the input value is BCD code  • When the input value is not BCD code   [Reference information] Value range of WORD type as interpreted as BCD code : WORD#0000 to WORD#9999 Value range of INT type: -32,768 to 32,767

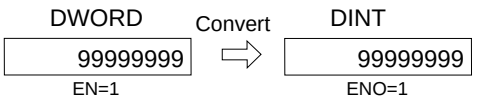
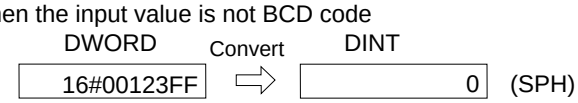
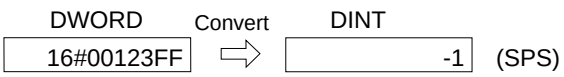
(56) Type conversion D_BCD_TO_INT

Name, Symbol, Function	Example
Name D_BCD_TO_INT **Symbol**  **Function** (1) D_BCD_TO_INT regards DWORD data as BCD code and converts it to INT type data. (2) When SPH is used, if the input DWORD value exceeds the valid range of INT type after type conversion, the upper limit value (32767) of INT type is output. (3) When SPH is used, if the input DWORD value is other than BCD code, “0” is output. (4) When SPS is used, if the input DWORD value exceeds the valid range of INT type after type conversion, no boundary processing is performed and lower 16 bits are output. (5) When SPS is used, if the input DWORD value is other than BCD code, “-1” is output.	<Operation> • When the operation result falls within the valid value range of the INT type  • When the operation result exceeds the valid value range of the INT type   • When the input value is not BCD code.   [Reference information] Value range of DWORD type as interpreted as BCD code : WORD#00000000 to WORD#99999999 Value range of INT type: -32,768 to 32,767

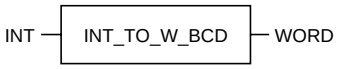
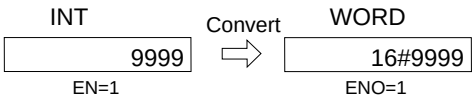
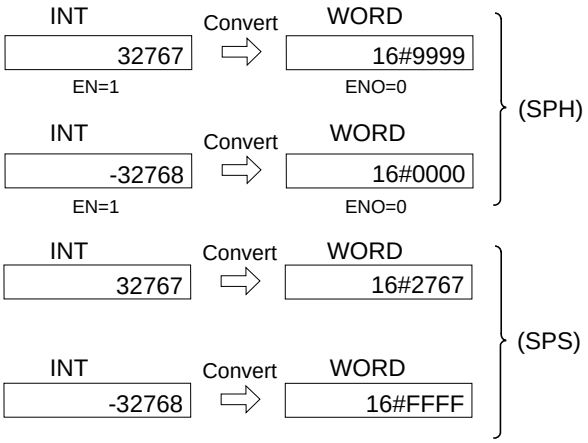
(57) Type conversion W_BCD_TO_DINT

Name, Symbol, Function		Example
Name	W_BCD_TO_DINT	<Operation> • When the input value is BCD code  • When the input value is not BCD code  (SPH)  (SPS) [Reference information] Value range of WORD type as interpreted as BCD code : 16#0000 to 16#9999 Value range of DINT type: -2,147,483,648 to 2,147,483,647
Symbol	 	
Function	(1) W_BCD_TO_DINT regards WORD data as BCD code and converts it to DINT type data. (2) When SPH is used, if the input WORD value is other than BCD code, “0” is output. (3) When SPS is used, if the input WORD value is other than BCD code, “-1” is output.	

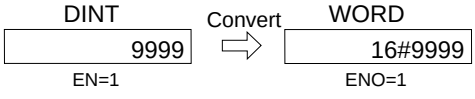
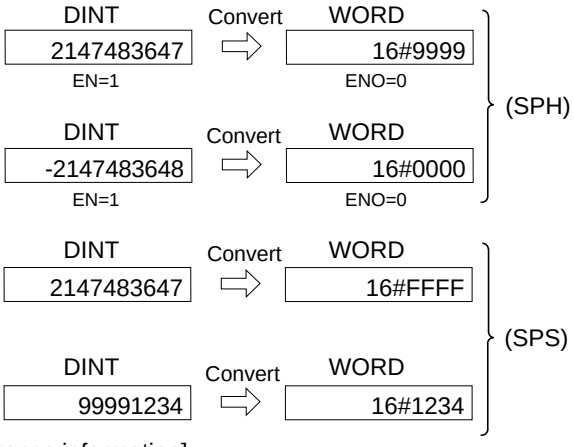
(58) Type conversion D_BCD_TO_DINT

Name, Symbol, Function		Example
Name	D_BCD_TO_DINT	<Operation> • When the operation result falls within the valid value range of the DINT type  • When the input value is not BCD code  (SPH)  (SPS) [Reference information] Value range of DWORD type as interpreted as BCD code : 16#00000000 to 16#99999999 Value range of DINT type: -2,147,483,648 to 2,147,483,647
Symbol	 	
Function	(1) D_BCD_TO_DINT regards DWORD data as BCD code and converts it to DINT type data. (2) A “0” is generated if the input WORD value is not BCD code. (3) When SPS is used, if the input DWORD value is other than BCD code, “-1” is output.	

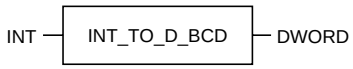
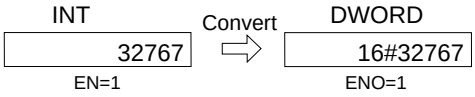
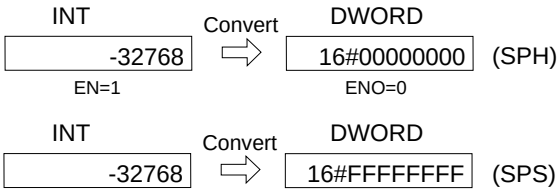
(59) Type conversion INT_TO_W_BCD

Name, Symbol, Function	Example
Name INT_TO_W_BCD **Symbol**  **Function** (1) INT_TO_W_BCD BCD-converts INT type data to WORD type data. (2) When SPH is used, if the input INT value exceeds the valid range of WORD type after type conversion, the boundary value of WORD type (BCD code) is output. (3) When SPS is used, if the input INT value exceeds 9999, the lower 4 digits of the INT value are output. If the input INT value is negative, “FFFF” is output.	<Operation> • When the operation result falls within the valid value range of the WORD type  • When the operation result exceeds the valid value range of the WORD type  [Reference information] Value range of INT type: -32,768 to 32,767 Value range of WORD type as interpreted as BCD code : 16#0000 to 16#9999

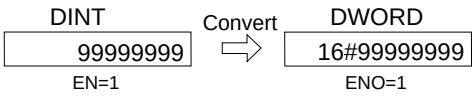
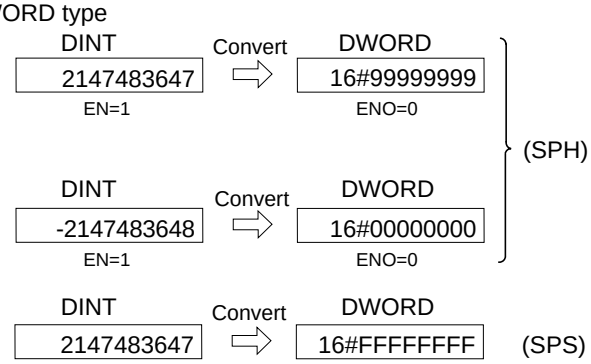
(60) Type conversion DINT_TO_W_BCD

Name, Symbol, Function	Example
Name DINT_TO_W_BCD **Symbol**  **Function** (1) DINT_TO_W_BCD BCD-converts DINT type data to WORD type data. (2) When SPH is used, if the input DINT value exceeds the valid range of WORD type after type conversion, the boundary value of WORD type (BCD code) is output. (3) When SPS is used, if the input DINT value is in the range from 10000 to 99999999, the lower 4 digits are output. If the input INT value exceeds 99999999 or is negative, “16#FFFF” is output.	<Operation> • When the operation result falls within the valid value range of the WORD type  • When the operation result exceeds the valid value range of the WORD type  [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of WORD type as interpreted as BCD code : 16#0000 to 16#9999

(61) Type conversion INT_TO_D_BCD

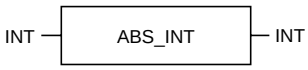
Name, Symbol, Function	Example
Name: INT_TO_D_BCD Symbol:  Function: (1) INT_TO_D_BCD BCD-converts INT type data to DWORD type data. (2) When SPH is used, if the input INT value exceeds the valid range of DWORD type after type conversion, the lower limit value of DWORD type (16#00000000) (BCD code) is output. (3) When SPS is used, if the input INT value is negative, “16#FFFFFFFF” is output.	<Operation> • When the operation result falls within the valid value range of the DWORD type  • When the operation result exceeds the valid value range of the DWORD type  [Reference information] Value range of INT type: -32,768 to 32,767 Value range of DWORD type as interpreted as BCD code : 16#00000000 to 16#99999999

(62) Type conversion DINT_TO_D_BCD

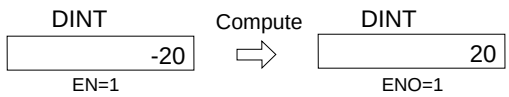
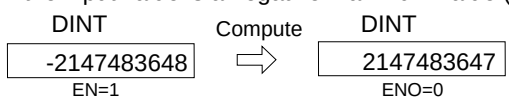
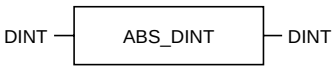
Name, Symbol, Function	Example
Name: DINT_TO_D_BCD Symbol:  Function: (1) DINT_TO_D_BCD BCD-converts DINT type data to DWORD type data. (2) When SPH is used, if the input DINT value exceeds the valid range of DWORD type after type conversion, the boundary value of DWORD type (BCD code) is output. (3) When SPS is used, if the input DINT value exceeds 99999999 or is negative, “16#FFFFFFFF” is output.	<Operation> • When the operation result falls within the valid value range of the DWORD type  • When the operation result exceeds the valid value range of the DWORD type  [Reference information] Value range of DINT type: -2,147,483,648 to 2,147,483,647 Value range of DWORD type as interpreted as BCD code : 16#00000000 to 16#99999999

2-5-4 Arithmetic functions

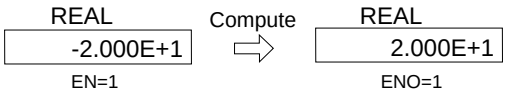
(1) Absolute value ABS_INT

Name, Symbol, Function		Example
Name	ABS_INT	<Operation>  • When the input value is a negative maximum value (-32768) 
Symbol	 	
Function	(1) ABS_INT computes and returns the absolute value of INT type data. (2) The positive maximum value (32767) of the output data type is generated if the input value is “-32768.”	

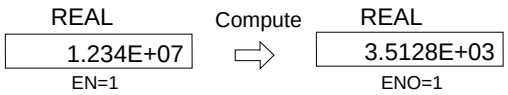
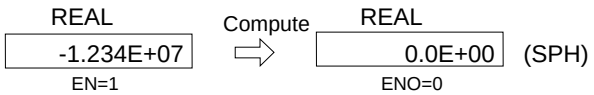
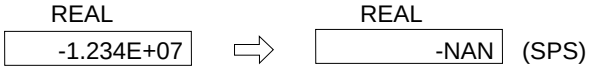
(2) Absolute value ABS_DINT

Name, Symbol, Function		Example
Name	ABS_DINT	<Operation>  • When the input value is a negative maximum value (-2147483648) 
Symbol	 	
Function	(1) ABS_DINT computes and returns the absolute value of DINT type data. (2) The positive maximum value (2147483647) of the output data type is generated if the input value is “-2147483648.”	

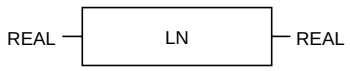
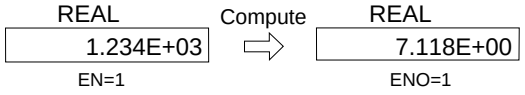
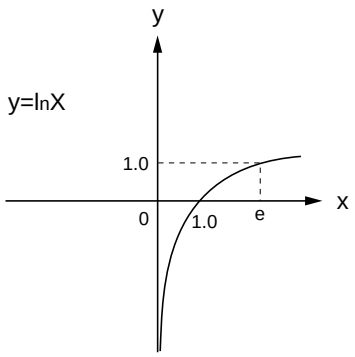
(3) Absolute value ABS_REAL

Name, Symbol, Function	Example
Name: ABS_REAL Symbol:  Function: (1) ABS_REAL computes the absolute value of REAL type data.	<Operation> 

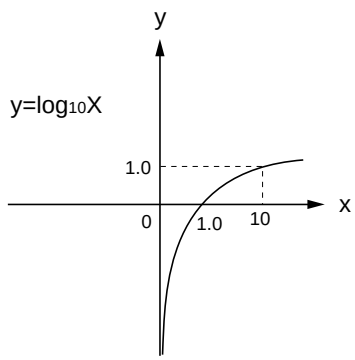
(4) Square root SQRT

Name, Symbol, Function	Example
Name: SQRT Symbol:  Function: (1) SQRT computes the square root of REAL type data. (2) When SPH is used, if the input value is negative, a 0 is output and ENO is set to 0. (3) The number of significant digits of the output is 5. (4) When SPS is used, if the input value is negative, “-NAN” is output.	<Operation>  • When the input value is negative  (SPH)  (SPS)

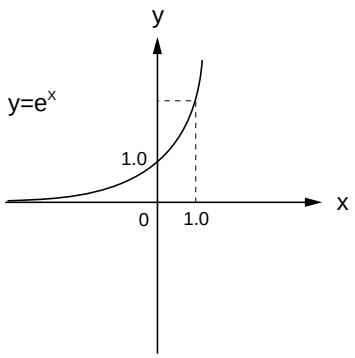
(5) Natural logarithm LN

Name, Symbol, Function		Example
Name	LN	<Operation>
Symbol		
Function	(1) LN computes the natural logarithm of REAL type data. (2) When SPH is used, if the input value is negative, a 0 is output and ENO is set to 0. (3) The output is the negative maximum value and ENO is set to 0 if the input value is 0. (4) The number of significant digits of the output is 4 or less. (5) When SPS is used, if the input value is negative, “-NAN” is output. If the input value is 0 (zero), “-INF” is output.	

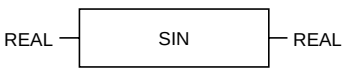
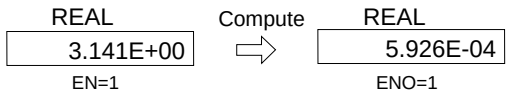
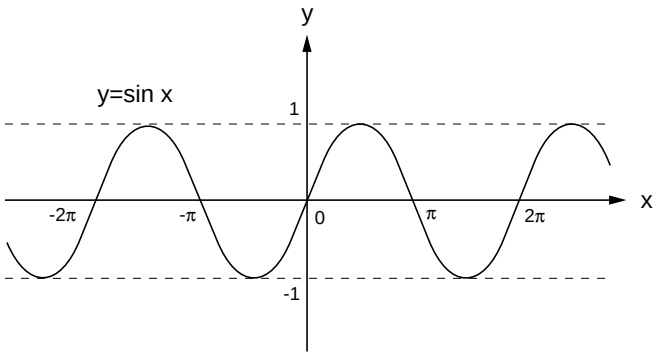
(6) Common logarithm LOG

Name, Symbol, Function		Example
Name	LOG	<Operation>
Symbol		
Function	(1) LOG computes the common logarithm of REAL type data. (2) When SPH is used, if the input value is negative, a 0 is output and ENO is set to 0. (3) The output is the negative maximum value and ENO is set to 0 if the input value is 0. (4) The number of significant digits of the output is 4 or less. (5) When SPS is used, if the input value is negative, “-NAN” is output. If the input value is 0 (zero), “-INF” is output.	

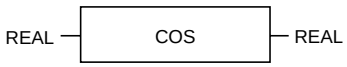
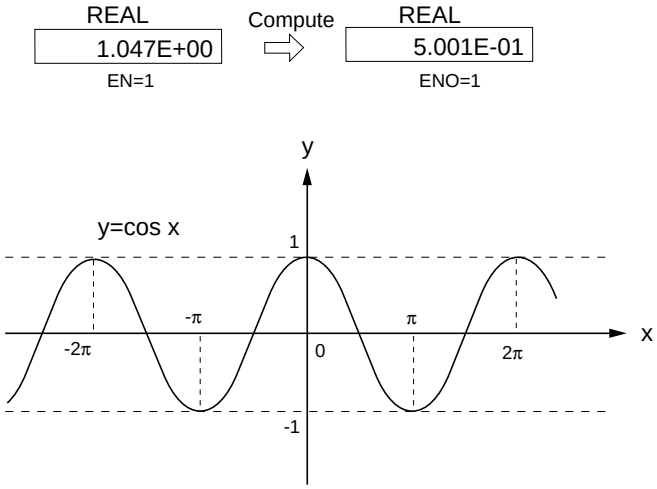
(7) Exponent EXP

Name, Symbol, Function	Example
Name EXP **Symbol**  **Function** (1) EXP computes the exponent of the input using the base (e) as 2.718281. (2) When SPH is used, if the operation result exceeds the boundary value of REAL type, the boundary value is output. (3) The number of significant digits of the output is as follows: • 4 digits if the input and output fall within -64 to 64. • In other cases, a large error results. (4) ENO is set to 0 if the output value exceeds the valid value range of the REAL type. If the output value is the REAL type and is so close to 0 that it cannot be expressed, the output is set to 0 and ENO to 1. (5) When SPS is used, if the operation result exceeds the boundary value of REAL type, "+INF" is output.	<Operation>  

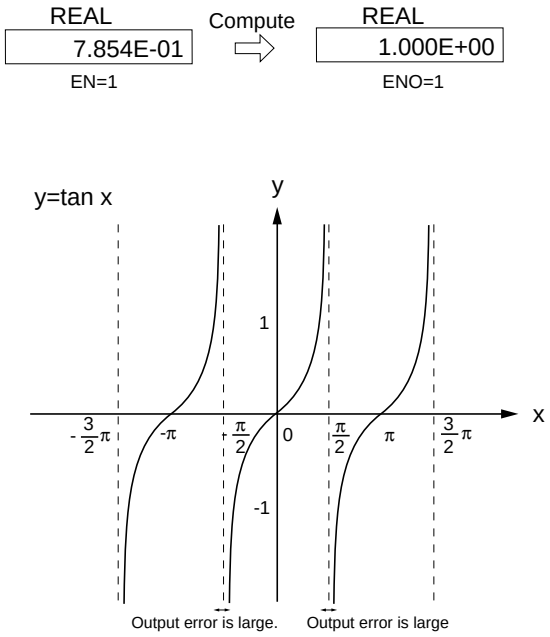
(8) Sine SIN

Name, Symbol, Function	Example
Name SIN **Symbol**  **Function** (1) SIN computes the sine of REAL type data. (2) The input must be given in radians (angle x π / 180). (3) The number of significant digits of the output is 5 if the input falls within -2π and 2π. (Calculated up to the fourth decimal place.) If the absolute value of the input is greater than or equal to 2π, computation is carried out but with a large error. (4) When an (input value) is < -2^{31} or > 2^{31} -1, an output value = 0 and ENO = 0.	<Operation>  

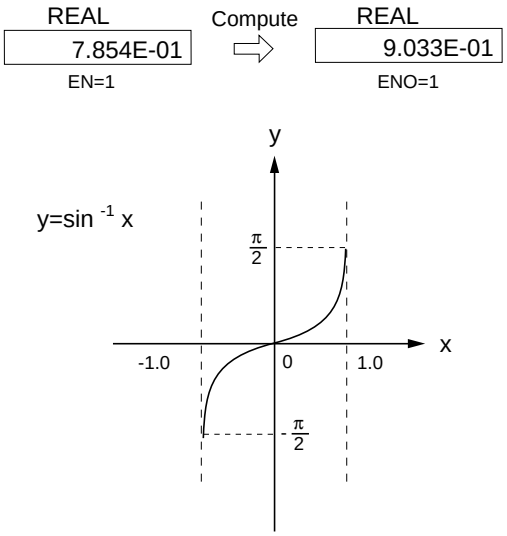
(9) Cosine COS

Name, Symbol, Function		Example
Name	COS	<Operation>
Symbol		
Function	(1) COS computes the cosine of REAL type data. (2) The input must be given in radians. (3) The number of significant digits of the output is 5 if the input falls within -2π and 2π. (Calculated up to the fourth decimal place.) If the absolute value of input is greater than or equal to 2π, computation is carried out but with a large error. (4) When an (input value) is < -2^{31} or > 2^{31} -1, an output value = 0 and ENO = 0.	

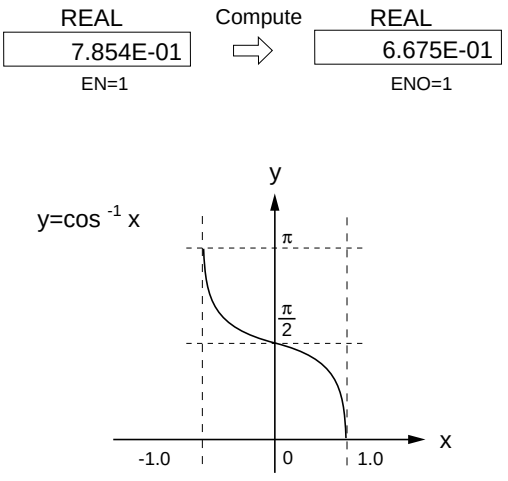
(10) Tangent TAN

Name, Symbol, Function		Example
Name	TAN	<Operation>
Symbol		
Function	(1) TAN computes the tangent of REAL type data. (2) The input must be given in radians. (3) The number of significant digits of the output is 4 if the input falls within -2π and 2π. The output has a large error if the absolute value of the input is greater if the input is approximately an integral multiple of π/2. If the absolute value of input is greater than or equal to 2π, computation is carried out but with a large error. (4) When an (input value) is < -2^{31} or > 2^{31} -1, an output value = 0 and ENO = 0.	

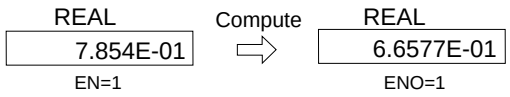
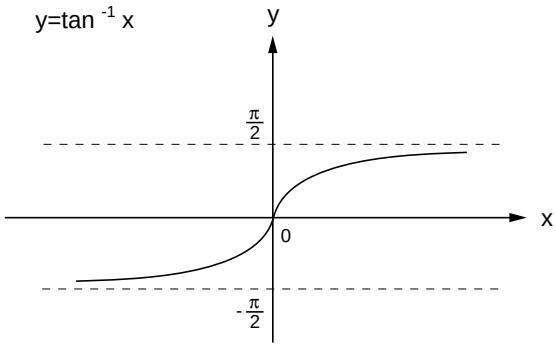
(11) Arc sine ASIN

Name, Symbol, Function		Example
Name	ASIN	<Operation>
Symbol		
Function	(1) ASIN computes the arc sine of REAL type data. (2) The output is given in radians. (3) The input must have a value range of -1.0 to +1.0 and the output must have a value range of -π/2 to π/2. A 0 is generated and ENO is set to 0 if the input exceeds this value range. (4) The number of significant digits of the output is as follows: • 4 digits if | input value | = 1.0 or 0.998999 or less • The error is large if | input value | = 0.999 and 0.999999 (5) When SPS is used, if the input value is outside the range from -1.0 to 1.0, "-NAN" is output.	

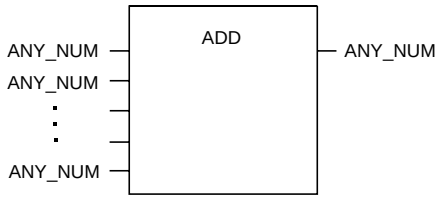
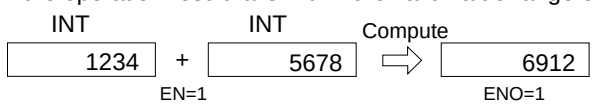
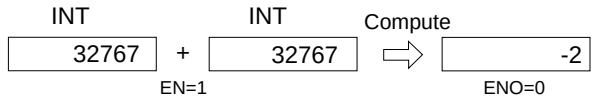
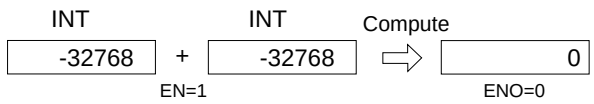
(12) Arc cosine ACOS

Name, Symbol, Function		Example
Name	ACOS	<Operation>
Symbol		
Function	(1) ACOS computes the arc cosine of REAL type data. (2) The output is given in radians. (3) The input must have a value range of -1.0 to +1.0 and the output must have a value range of π to 0. A 0 is generated and ENO is set to 0 if the input exceeds this value range. (4) The number of significant digits of the output is as follows: • 4 digits if | input value | = 1.0 or 0.998999 or less • The error is large if | input value | = 0.99 and 0.999999 (5) When SPS is used, if the input value is outside the range from -1.0 to 1.0, "-NAN" is output.	

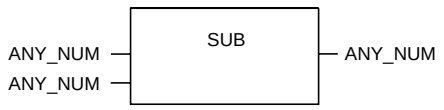
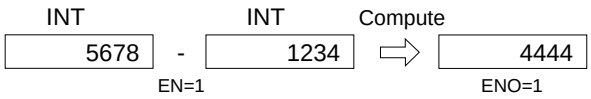
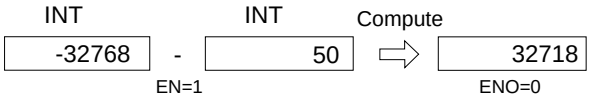
(13) Arc tangent ATAN

Name, Symbol, Function	Example
Name: ATAN **Symbol**:  **Function**: (1) ATAN computes the arc tangent of REAL type data. (2) The output is given in radians. (3) The input must have a value range of its negative maximum value to its positive maximum value and the output must have a value range of $-\pi/2$ to $\pi/2$. (4) The number of significant digits of the output is 5.	<Operation>  

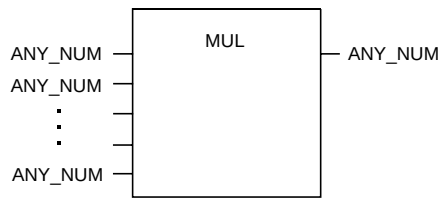
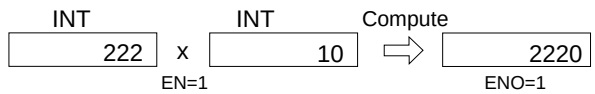
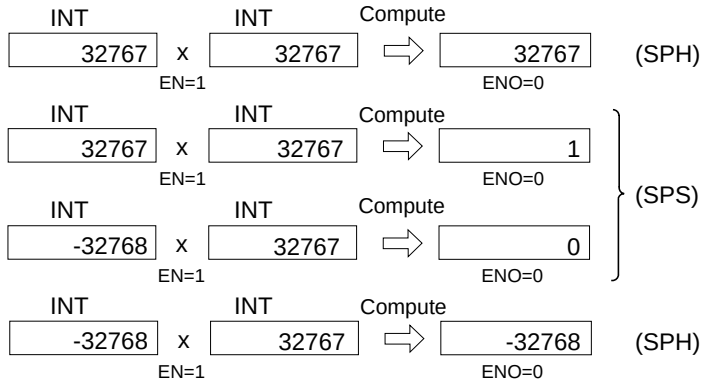
(14) Addition ADD

Name, Symbol, Function	Example
Name: ADD **Symbol**:  **Function**: (1) ADD adds the input data together. (2) The data type of the input and output operands must be the same. (3) Available data type is ANY_NUM type (REAL, INT, DINT, UINT, or UDINT). (4) ENO is set to 0 if the operation result exceeds the valid value range of the data type of the operands. (5) If a data value is approaching so closely to 0 (zero) that no REAL type data can represent it, an output value = 1 and ENO = 1 (only for SPH).	<Operation> • When the operation result falls within the valid value range of the operands  • When the operation result exceeds the valid value range of the operands   Note: 1) The CPU performs no boundary processing when the result of adding operands of types other than REAL type exceeds the valid value rangeof the operands. Make sure that the result of the addition does not exceed the valid value range of the operands. The CPU performs boundary processing for REAL type operands (only for SPH). Note: 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of the data type.

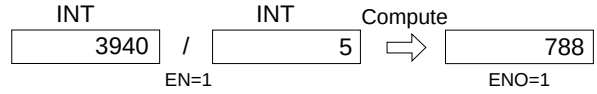
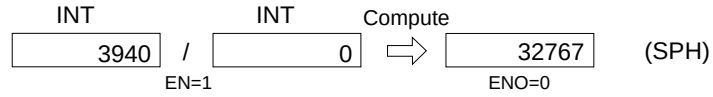
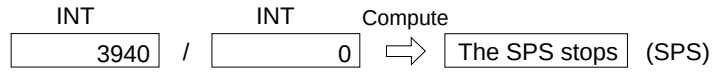
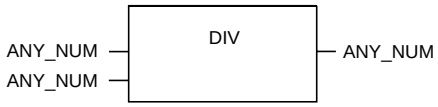
(15) Subtraction SUB

Name, Symbol, Function	Example
Name: SUB Symbol:  Function: (1) SUB subtracts the second input from the first input. (2) The data type of the input and output operands must be the same. (3) Available data type is ANY_NUM type (REAL, INT, DINT, UINT, or UDINT). (4) ENO is set to 0 if the operation result exceeds the valid value range of the data type of the operands (only for SPH). (5) If a data value is approaching so closely to 0 (zero) that no REAL type data can represent it, an output value = 1 and ENO = 1 (only for SPH).	<Operation> • When the operation result falls within the valid value range of the operands  • When the operation result exceeds the valid value range of the operands  Note: 1) The CPU performs no boundary processing when the result of subtracting operands of types other than REAL type exceeds the valid value range of the operands. Make sure that the result of the subtraction does not exceeds the valid value range of the operands. The CPU performs boundary processing for REAL type operands (only for SPH). Note: 2) When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of the data type.

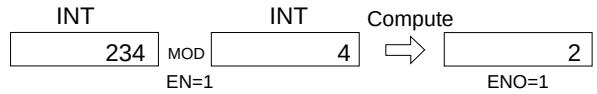
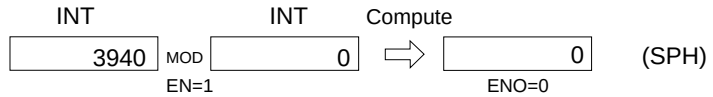
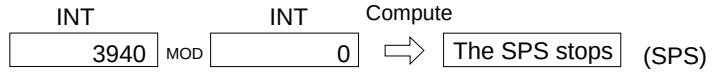
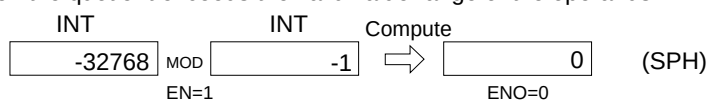
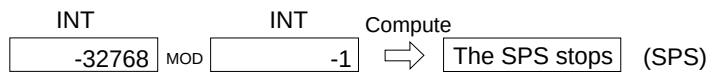
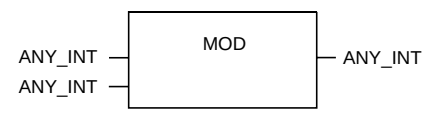
(16) Multiplication MUL

Name, Symbol, Function	Example
Name: MUL Symbol:  Function: (1) MUL multiplies the input data together. (2) The data type of the input and output operands must be the same. (3) ENO is set to 0 if the operation result exceeds the valid value range of the data type of the operands (only for SPH). (4) Available data type is ANY_NUM type (REAL, INT, DINT, UINT, or UDINT). (5) If a data value is approaching so closely to 0 (zero) that no REAL type data can represent it, an output value = 1 and ENO = 1 (only for SPH).	<Operation> • When the operation result falls within the valid value range of the operands  • When the operation result exceeds the valid value range of the operands  Note: When SPS is used, no boundary processing is performed even when the operation result exceeds the valid range of the data type.

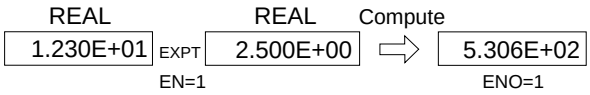
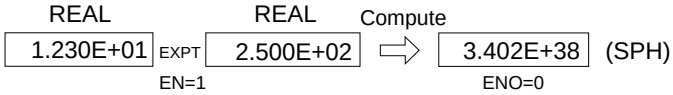
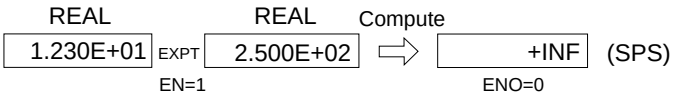
(17) Division DIV

Name, Symbol, Function		Example
Name	DIV	<Operation> • When the operation result falls within the valid value range of the operands  • When the divisor is 0  
Symbol	 	
Function	(1) DIV divides the first input by the second input. (2) The data type of the input and output operands must be the same. (3) ENO is set to 0 if the operation result exceeds the valid value range of the data type of the operands (only for SPH). (4) Available data type is ANY_NUM type (REAL, INT, DINT, UINT, or UDINT). (5) If a data value is approaching so closely to 0 (zero) that no REAL type data can represent it, an output value = 1 and ENO = 1. (6) If the divisor is 0, the maximum absolute value with the sign of the dividend is generated and ENO is set to 0 (only for SPH).	Notes: 1) When SPS is used, if the operation result exceeds the valid range of the data type, the SPS stops operation. Notes: 2) When SPS is used, if the divisor is 0 (zero) (except REAL type), the SPS stops operation. In the case of REAL type, "-NAN" or "±INF" is output.

(18) Division remainder MOD

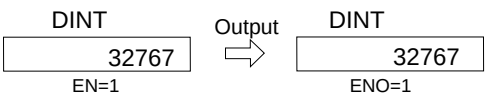
Name, Symbol, Function		Example
Name	MOD	<Operation> • When the operation result falls within the valid value range of the operands  • When the divisor is 0   • When the quotient exceeds the valid value range of the operands  
Symbol	 	
Function	(1) MOD divides the first input by the second input and generates the remainder. (2) The data type of the input and output operands must be the same. (3) The output is set to 0 and ENO is set to 0 if the quotient exceeds the valid value range of the data type of the operands (only for SPH). (4) Available data type is ANY_INT type (INT, DINT, UINT, or UDINT). (5) If the divisor is 0, a 0 is generated and ENO is set to 0 (only for SPH). (6) Remainder is calculated using the following equation: (Divisor) x (Quotient) + (Remainder) = (Dividend)	Note: When SPS is used, if the quotient of the division exceeds the boundary value of the data type, or if the divisor is 0 (zero), the SPS stops operation.

(19) Exponent EXPT

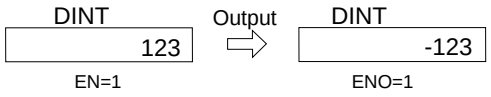
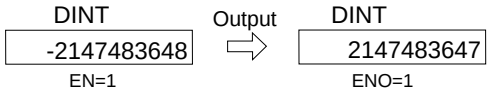
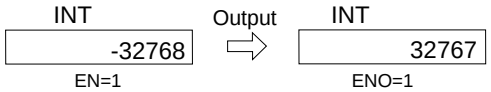
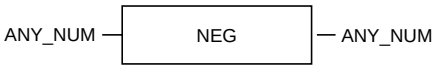
Name, Symbol, Function	Example
Name: EXPT Symbol:  Function: (1) EXPT computes the first input (base) to the power (exponent) of the second input. (2) When SPH is used, if the operation result exceeds the valid value range of the REAL type, the boundary value of REAL type is output. (3) When SPH is used, base ≥ 0. If base < 0, the output is set to 0 and ENO to 0. (4) The number of significant digits of the output is 4. (5) ENO is set to 0 if the output value exceeds the valid value range of the REAL type. If the output value is the REAL type and is so close to 0 that it cannot be expressed, the output is set to 0 and ENO to 1.	<Operation> • When the operation result falls within the valid value range of the operand  • When the operation result exceeds the valid value range of the operand  (SPH)  (SPS)

(6) When a base = 0 and an exponent = 0 (0°C), an output value = 1 and ENO = 1.
(7) When SPS is used, if the operation result exceeds the boundary value of REAL type, “+INF” is output.
(8) When SPS is used, base ≥ 0. If base < 0, “-NAN” is output.

(20) Move MOVE

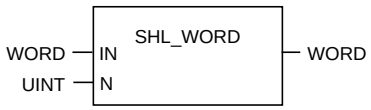
Name, Symbol, Function	Example
Name: MOVE Symbol:  Function: (1) MOVE outputs the input as it is. (2) Available data type is any data type (ANY type) of the MICREX-SX series.	<Operation>  Note: Since when a variable is a derived data type, transfer is repeated the same number of times as that of elements. If a variable has been defined in the processor bus space, pay attention to the number of accesses. Refer to “Appendix Accessing the Processor Bus.”

(21) Negation NEG

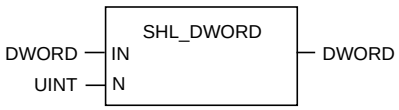
Name, Symbol, Function		Example
Name	NEG	<Operation>  For a negative value of DINT or INT type  
Symbol	 	
Function	(1) NEG outputs the input data with its sign negated. (2) Available data are types INT, DINT, or REAL. (Neither UINT nor UDINT must be specified.) (3) ENO is set to 0 if the result of negating an INT type data or DINT type data exceeds the valid value range of the respective data type.	

2-5-5 Bit string functions

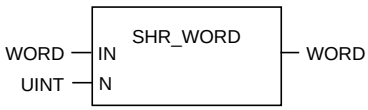
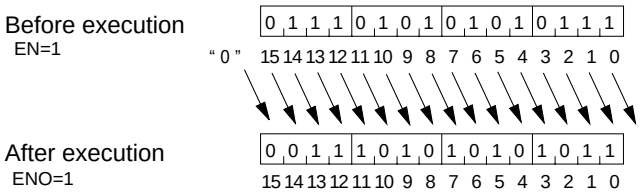
(1) Shift left SHL_WORD

Name, Symbol, Function	Example
Name: SHL_WORD	<Operation> Before execution EN=1 0 1 1 1 | 0 1 0 1 | 0 1 0 1 | 0 1 1 1 Bit addresses → 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 "0" After execution ENO=1 1 1 1 0 | 1 0 1 0 | 1 0 1 0 | 1 1 1 0 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Symbol: 	
Function: (1) SHL_WORD shifts the first input (WORD type data) the number of bit positions designated by the second input (UINT type data) to the left. Bit 0 is loaded with a "0" after the shift. The data in bit 15 is discarded. (2) The lowest-order 4 bits of the second input "N" are significant. For example, SHL_WORD shifts 0 bits when N = 16 and 1 bit when N = 17.	

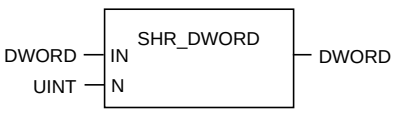
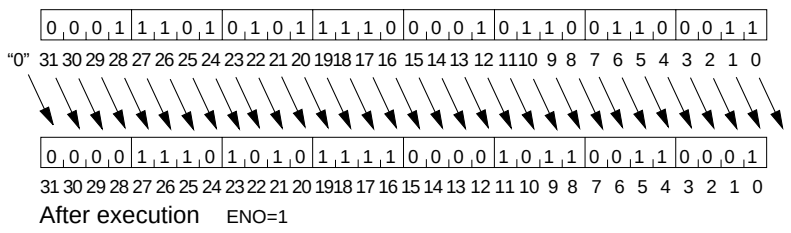
(2) Shift left SHL_DWORD

Name, Symbol, Function	Example
Name: SHL_DWORD	<Operation> Example of shifting 1 bit position Before execution 0 0 0 1 | 1 1 0 1 | 0 1 0 1 | 1 1 1 0 | 0 0 0 1 | 0 1 1 0 | 0 1 1 0 | 0 0 1 1 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 "0" 0 0 1 1 | 1 0 1 0 | 1 0 1 1 | 1 1 0 0 | 0 0 1 0 | 1 1 0 0 | 1 1 0 0 | 0 1 1 0 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 After execution ENO=1
Symbol: 	
Function: (1) SHL_DWORD shifts the first input (DWORD type data) the number of bit positions designated by the second input (UINT type data) to the left. Bit 0 is loaded with a "0" after the shift. The data in bit 31 is discarded. (2) The lowest-order 5 bits of the second input "N" are significant. For example, SHL_DWORD shifts 0 bits when N = 32 and 1 bit when N = 33.	

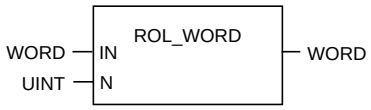
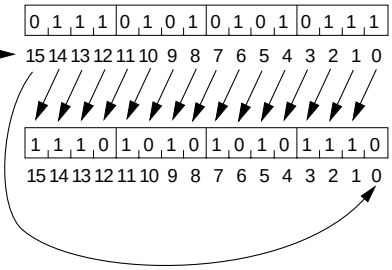
(3) Shift right SHR_WORD

Name, Symbol, Function		Example
Name	SHR_WORD	<Operation> Example of shifting 1 bit position
Symbol		
Function	(1) SHR_WORD shifts the first input (WORD type data) the number of bit positions designated by the second input (UINT type data) to the right. Bit 15 is loaded with a "0" after the shift. The data in bit 0 is discarded. (2) The lowest-order 4 bits of the second input "N" are significant. For example, SHR_WORD shifts 0 bits when N = 16 and 1 bit when N = 17.	

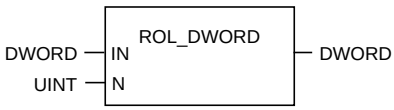
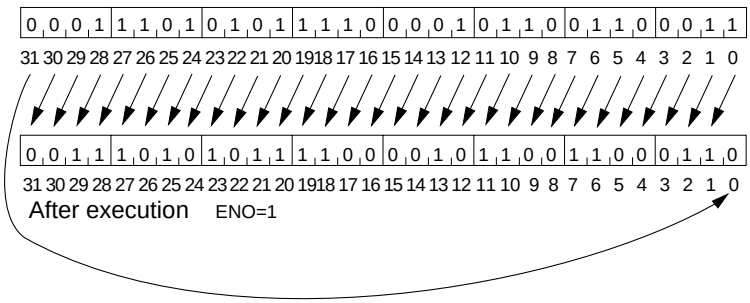
(4) Shift right SHR_DWORD

Name, Symbol, Function		Example
Name	SHR_DWORD	<Operation> Example of shifting 1 bit position Before execution EN=1
Symbol		 After execution ENO=1
Function	(1) SHR_DWORD shifts the first input (DWORD type data) the number of bit positions designated by the second input (UINT type data) to the right. Bit 31 is loaded with a "0" after the shift. The data in bit 0 is discarded. (2) The lowest-order 5 bits of the second input "N" are significant. For example, SHR_DWORD shifts 0 bits when N = 32 and 1 bit when N = 33.	

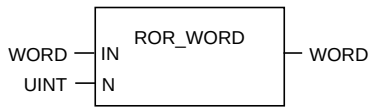
(5) Rotate left ROL_WORD

Name, Symbol, Function	Example
Name: ROL_WORD Symbol:  Function: (1) ROL_WORD rotates the first input (WORD type data) the number of bit positions designated by the second input (UINT type data) to the left. Bit 0 is loaded with the data from bit 15. (2) The lowest-order 4 bits of the second input “N” are significant. For example, ROL_WORD rotates 0 bits when N = 16 and 1 bit when N = 17.	<Operation> Example of rotating 1 bit Before execution EN=1 Bit addresses After execution ENO=1 

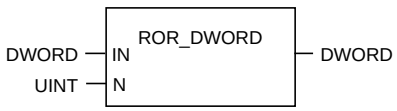
(6) Rotate left ROL_DWORD

Name, Symbol, Function	Example
Name: ROL_DWORD Symbol:  Function: (1) ROL_DWORD rotates the first input (DWORD type data) the number of bit positions designated by the second input (UINT type data) to the left. Bit 0 is loaded with the data from bit 31. (2) The lowest-order 5 bits of the second input “N” are significant. For example, ROL_DWORD rotates 0 bits when N = 32 and 1 bit when N = 33.	<Operation> Example of rotating 1 bit position Before execution EN=1  After execution ENO=1

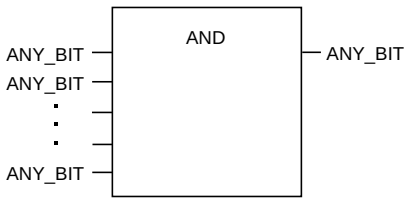
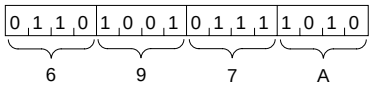
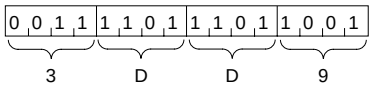
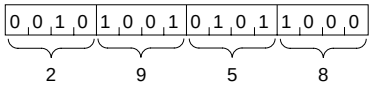
(7) Rotate right ROR_WORD

Name, Symbol, Function		Example
Name	ROR_WORD	<Operation> Example of rotating 1 bit Before execution EN=1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 1 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 After execution ENO=1 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 1 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Symbol		
Function	(1) ROR_WORD rotates the first input (WORD type data) the number of bit positions designated by the second input (UINT type data) to the right. Bit 15 is loaded with the data from bit 0. (2) The lowest-order 4 bits of the second input "N" are significant. For example, ROR_WORD rotates 0 bits when N = 16 and 1 bit when N = 17.	

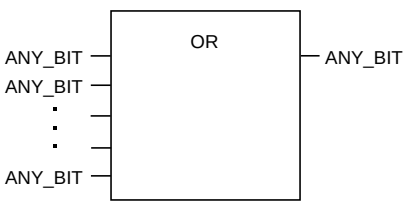
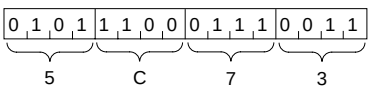
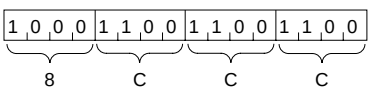
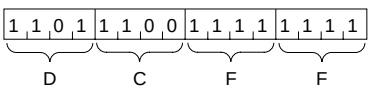
(8) Rotate right ROR_DWORD

Name, Symbol, Function		Function
Name	ROR_DWORD	<Operation> Example of rotating 1 bit position Before execution EN=1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 After execution ENO=1
Symbol		
Function	(1) ROR_DWORD rotates the first input (DWORD type data) the number of bit positions designated by the second input (UINT type data) to the right. Bit 31 is loaded with the data from bit 0. (2) The lowest-order 5 bits of the second input "N" are significant. For example, ROR_DWORD rotates 0 bits when N = 32 and 1 bit when N = 33.	

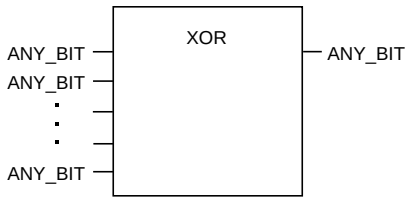
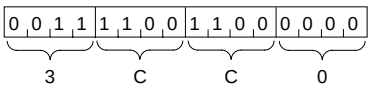
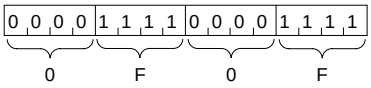
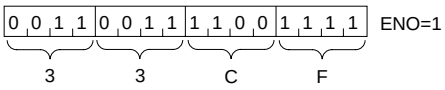
(9) Logical product AND

Name, Symbol, Function	Example
Name: AND Symbol:  Function: (1) AND generates the logical product AND of all inputs. (2) The data type of the input and output operands must all be the same. (3) Available data type is ANY_BIT type (BOOL, WORD, or DWORD).	<Operation> Input WORD  AND EN=1 Input WORD  Output WORD  ENO=1

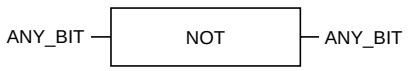
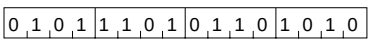
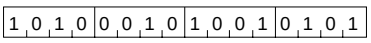
(10) Logical add OR

Name, Symbol, Function	Example
Name: OR Symbol:  Function: (1) OR generates the logical add OR of all inputs. (2) The data type of the input and output operands must all be the same. (3) Available data type is ANY_BIT type (BOOL, WORD, or DWORD).	<Operation> Input WORD  OR EN=1 Input WORD  Output WORD  ENO=1

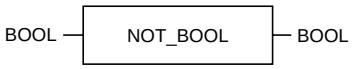
(11) Exclusive XOR

Name, Symbol, Function	Example
Name: XOR Symbol:  Function: (1) XOR generates the exclusive OR of all inputs. (2) The data type of the input and output operands must all be the same. (3) Available data type is ANY_BIT type (BOOL, WORD, or DWORD).	<Operation> Input WORD  XOR EN=1 Input WORD  Output WORD 

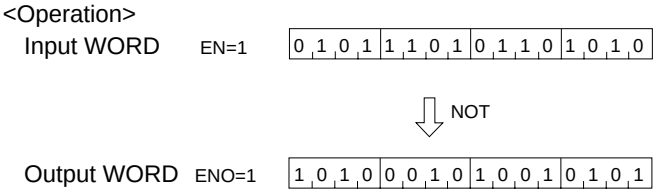
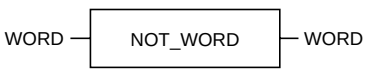
(12) Logical negation NOT

Name, Symbol, Function	Example
Name: NOT Symbol:  Function: (1) NOT generates an inverted bit string of the input. (2) Available data type is ANY_BIT type (BOOL, WORD, or DWORD).	<Operation> In the case of WORD type data Input word EN=1  NOT Output word ENO=1 

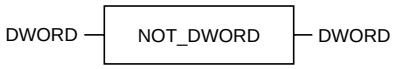
(13) Negation NOT_BOOL

Name, Symbol, Function		Example
Name	NOT_BOOL	<Operation> • When the input data has a value of “1”  • When the input data has a value of “0” 
Symbol	 	
Function	(1) NOT_BOOL generates an inverted value of BOOL data.	

(14) Negation NOT_WORD

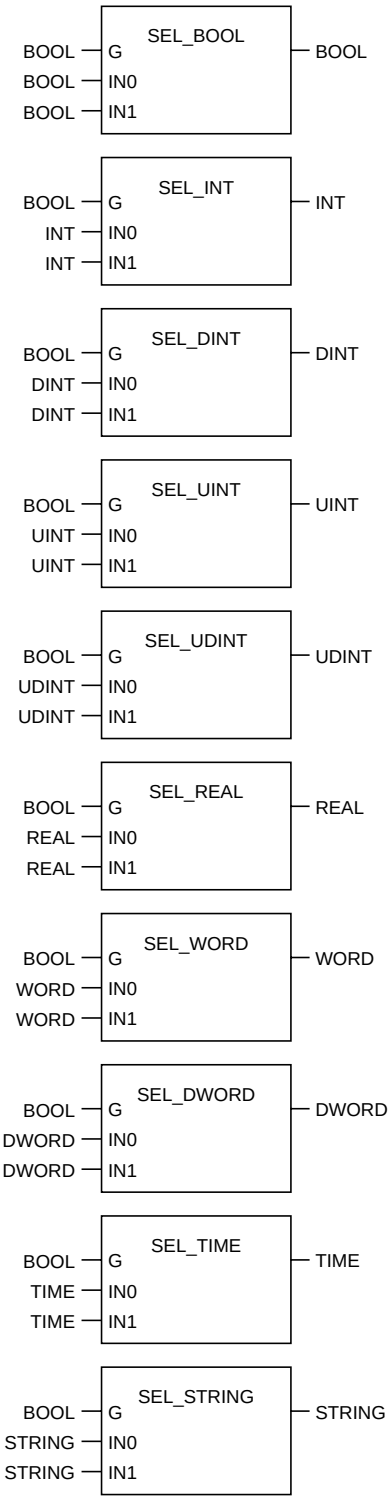
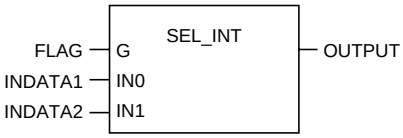
Name, Symbol, Function		Example
Name	NOT_WORD	<Operation> 
Symbol	 	
Function	(1) NOT_WORD inverts each bit of WORD data.	

(15) Negation NOT_DWORD

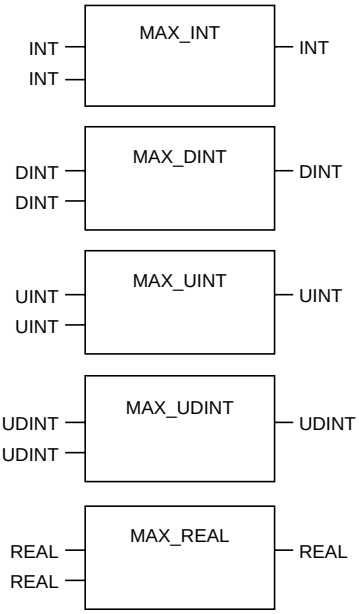
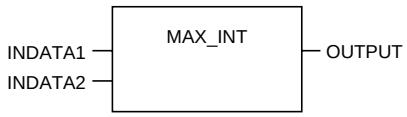
Name, Symbol, Function		Example
Name	NOT_DWORD	<Operation> Input DWORD EN=1 1 0 1 0 | 1 0 1 0 | 1 0 1 0 | 1 0 1 0 | 1 0 1 0 | 1 0 1 0 | 1 0 1 0 | 1 0 1 0 NOT Output DWORD ENO=1 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 1 0 1 | 0 1 0 1
Symbol		
Function	(1) NOT_DWORD inverts each bit of DWORD data.	

2-5-6 Selection/comparison functions

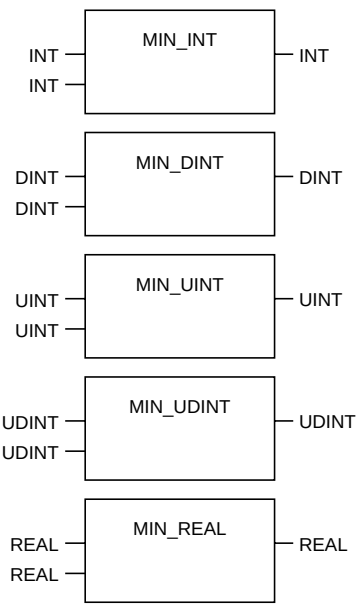
(1) Select SEL

Name, Symbol, Function	Example
Name: SEL_*	<Programming example>
Symbol: 	
Function: (1) The SEL_* outputs the value of the second input if the value of the first input is 0 and outputs the value of the third input if the value of the first input is 1.	<Operation> • When "FLAG" has a value of "0" INDATA1 12345 INDATA2 5000 ⇨ OUTPUT 12345 • When "FLAG" has a value of "1" INDATA1 12345 INDATA2 5000 ⇨ OUTPUT 5000 Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

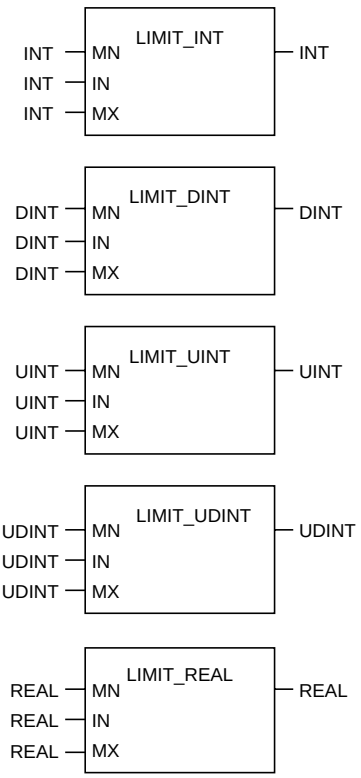
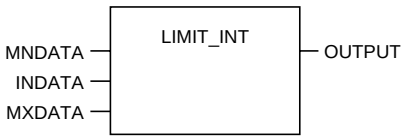
(2) Maximum value MAX

Name, Symbol, Function	Example
Name MAX_*	<Programming example>
Symbol 	 <Operation> INDATA1 1234 INDATA2 0 ⇨ OUTPUT 1234
Function (1) MAX_* outputs the maximum value of the inputs.	

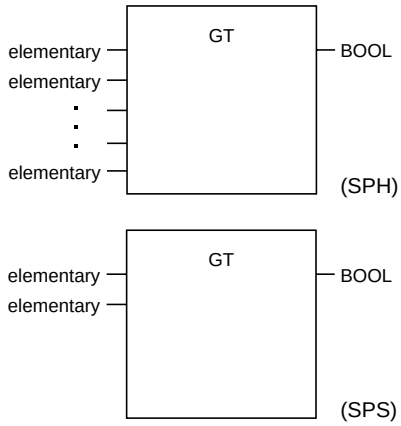
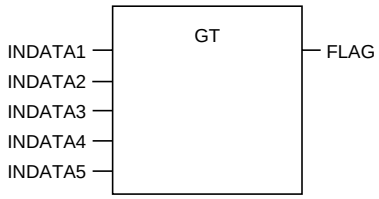
(3) Minimum value MIN

Name, Symbol, Function	Example
Name MIN_*	<Programming example>
Symbol 	 <Operation> INDATA1 1234 INDATA2 5678 ⇨ OUTPUT 1234
Function (1) MIN_* outputs the minimum value of the inputs.	

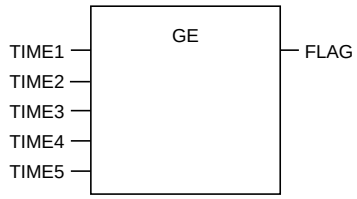
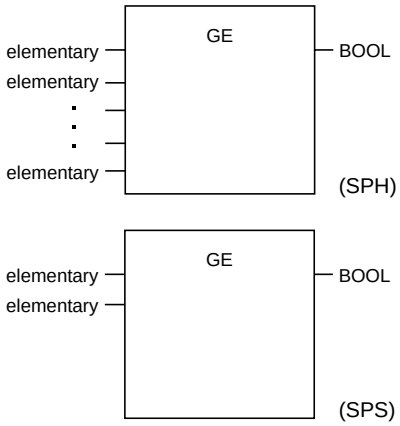
(4) Limit LIMIT

Name, Symbol, Function	Example
Name: LIMIT_*	<Programming example>
Symbol: 	
Function: (1) LIMIT_* outputs “MN” if “IN” is smaller than “MN.” It outputs “MX” if “IN” is greater than “MX.” In the other cases, LIMIT_* outputs “IN.” (2) LIMIT_* outputs “MX” if “MN” is greater than “MX.” Note: When SPS is used, if “MN” is greater than “MX” for LIMIT_UINT or LIMIT_UDINT, “MN” is output.	<Operation> • When “INDATA” ≤ “MNDATA” MNDATA 1000, INDATA 0 ⇨ OUTPUT 1000, MXDATA 5000 • When “MNDATA” < “INDATA” < “MXDATA” MNDATA 1000, INDATA 3000 ⇨ OUTPUT 3000, MXDATA 5000 • When “INDATA” ≥ “MXDATA” MNDATA 1000, INDATA 5500 ⇨ OUTPUT 5000, MXDATA 5000

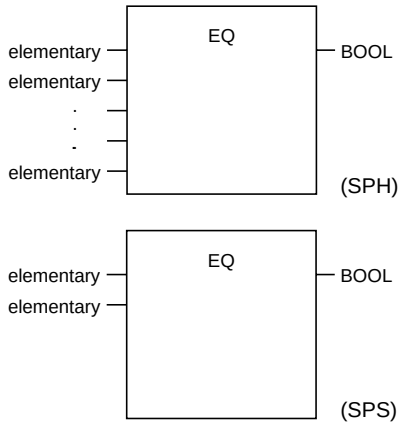
(5) Comparison (>) GT

Name, Symbol, Function	Example
Name GT **Symbol**  **Function** (1) GT outputs a 1 if the conditions first input > second input > third input > ... are met. GT outputs a 0 if the above conditions are not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of BOOL type and ANY_DATE type can be used. (4) The maximum number of inputs is 16 (only for SPH). When SPS is used, the input cannot be extended. Fixed to two inputs. Note: REAL type data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	<Programming example> Note: Example of a program for SPH  <Operation> • When the input is of INT type INDATA1 12345 INDATA2 10000 INDATA3 1000 INDATA4 900 INDATA5 30 Consequently, INDATA1 > INDATA2 > INDATA3 > INDATA4 > INDATA5 hold, and FLAG is set to 1.

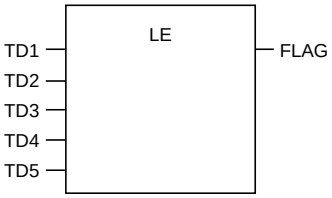
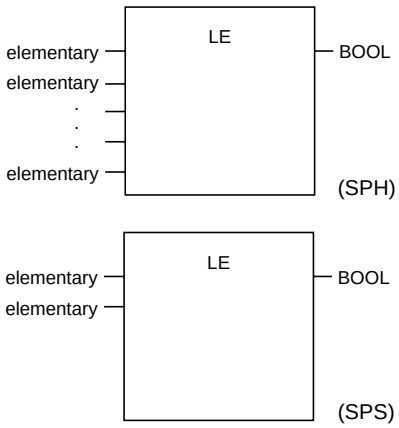
(6) Comparison (≥) GE

Name, Symbol, Function	Example
Name GE	<Programming example> Note: Example of a program for SPH 
Symbol 	<Operation> • When the input is of TIME type TIME1 10m5s TIME2 10m5s TIME3 9m10s TIME4 20s TIME5 0s
Function (1) GE outputs a 1 if the conditions first input ≥ second input ≥ third input ≥ ... are met. GE outputs a 0 if the above conditions are not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of BOOL type and ANY_DATE type can be used. (4) The maximum number of inputs is 16 (only for SPH). When SPS is used, the input cannot be extended. Fixed to two inputs. Note: REAL data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	This program results in TIME1 ≥ TIME2 ≥ TIME3 ≥ TIME4 ≥ TIME5, which sets a FLAG to 1.

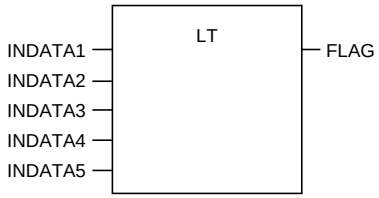
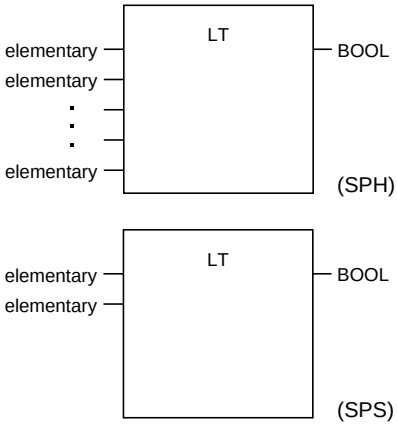
(7) Comparison (=) EQ

Name, Symbol, Function	Example
Name EQ **Symbol**  **Function** (1) EQ outputs a 1 if the conditions first input = second input = third input = ... are met. EQ outputs a 0 if the above conditions are not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of ANY_DATE type can be used. (4) The maximum number of inputs is 16 (only for SPH). When SPS is used, the input cannot be extended. Fixed to two inputs. Note: REAL data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	<Programming example> Note: Example of a program for SPH  <Operation> • When the input is of DATE type DATE1 1984-06-25 DATE2 1972-02-14 DATE3 1973-09-23 DATE4 1991-08-07 DATE5 1994-06-04 This program results in DATE1 ≠ DATE2 ≠ DATE3 ≠ DATE4 ≠ DATE5, which sets a FLAG to 1.

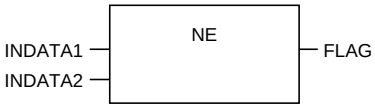
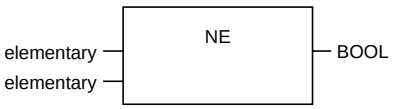
(8) Comparison (≤) LE

Name, Symbol, Function	Example
Name LE	<Programming example> Note: Example of a program for SPH 
Symbol 	<Operation> • When the input is of TOD type TD1 15:36:34 TD2 15:37:50 TD3 15:40:00 TD4 15:50:11 TD5 18:00:20 This program results in TD1 ≤ TD2 ≤ TD3 ≤ TD4 ≤ TD5, which sets a FLAG to 1.
Function (1) LE outputs a 1 if the conditions first input ≤ second input ≤ third input ≤ ... are met. LE outputs a 0 if the above conditions are not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of BOOL type and ANY_DATE type can be used. (4) The maximum number of inputs is 16 (only for SPH). When SPS is used, the input cannot be extended. Fixed to two inputs. Note: REAL data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	

(9) Comparison (<) LT

Name, Symbol, Function	Example
Name LT	<Programming example> Note: Example of a program for SPH 
Symbol 	<Operation> • When the input is of INT type INDATA1 -12345 INDATA2 -100 INDATA3 0 INDATA4 100 INDATA5 12345 This program results in INDATE1 < INDATE2 < INDATE3 < INDATE4 < INDATE5, which sets a FLAG to 1.
Function (1) LT outputs a 1 if the conditions first input < second input < third input < ... are met. LT outputs a 0 if the above conditions are not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of BOOL type and ANY_DATE type can be used. (4) The maximum number of inputs is 16 (only for SPH). When SPS is used, the input cannot be extended. Fixed to two inputs. Note: REAL data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	

(10) Comparison (≠) NE

Name, Symbol, Function	Example
Name NE	<Programming example>  <Operation> • When the input is of REAL type INDATA1 2.0E-10 INDATA2 2.0E+10 This program results in INDATE1 ≠ INDATE2 which sets a FLAG to 1.
Symbol  	
Function (1) NE outputs a 1 if the condition that the first input ≠ the second input is met. NE outputs a 0 if the above condition is not met. (2) The data type of the input operands must be the same. (3) Available data types are ANY_NUM type (REAL, INT, DINT, UINT, UDINT), ANY_BIT type (BOOL, WORD, DWORD), ANY_DATE type (DT, DATE, or TOD), and TIME type. When SPS is used, none of ANY_DATE type can be used. Note: REAL data generally involves an error. The error will get larger as computations are executed. Accordingly, the result of the REAL type data comparison operation may differ from the actual value. When using this instruction, consider the presence of this type of error.	

2-5-7 Character string functions

(1) Get length LEN

Name, Symbol, Function		Example
Name	LEN	<Operation>
Symbol		STRING A B C D E F G H I J EN=1 ⇩ Compute INT 10 ENO=1
Function	(1) LEN outputs an INT type data that indicates the number of characters in the input data of STRING type.	Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

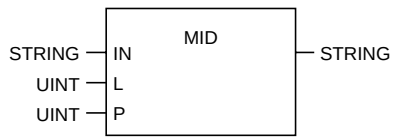
(2) Get left sub-string LEFT

Name, Symbol, Function		Example
Name	LEFT	<Operation>
Symbol		STRING A B C D E F G H I J UINT 5 EN=1 ⇩ Extract STRING A B C D E ENO=1
Function	(1) LEFT extracts a sub-string of the length designated by the second input from the left side of the STRING data designated by the first input. (2) If the number of characters “L” is greater than the number of characters in the input string, LEFT outputs the input string as is and sets ENO to 1. (3) Only NULL is output and ENO is set to 1 if the number of characters to extract is equal to 0.	Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

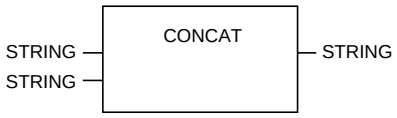
(3) Get right sub-string RIGHT

Name, Symbol, Function	Example
Name: RIGHT **Symbol**:  **Function**: (1) RIGHT extracts a sub-string of the length designated by the second input from the right side of the STRING type data designated by the first input. (2) If the number of characters “L” is greater than the number of characters in the input string, RIGHT outputs the input string as is and sets ENO to 1. (3) Only NULL is output and ENO is set to 1 if the number of characters to extract is equal to 0.	<Operation> STRING A B C D E F G H I J EN=1 UINT 5 ⇩ Extract STRING F G H I J ENO=1 Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

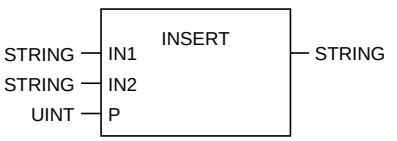
(4) Get middle sub-string MID

Name, Symbol, Function	Example
Name: MID **Symbol**:  **Function**: (1) MID extracts a sub-string of the length designated by the second input starting from the Pth character position (the third input) in the first input of STRING type. (2) When SPH is used, if P≥ 65 or P = 0, only NULL is output and ENO is set to 0. (3) When the number of input characters is < P and L ≠ 0, only NULL is output, resulting in ENO = 0. (4) The string starting at the first character position until the end of the input string is output and ENO is set to 1 if the starting position “P” is smaller than the length of the string and the sum of the starting position P and the number of characters “L” exceeds the length of the input string. (5) Only NULL is output and ENO is set to 1 if the number of extracted characters = 0. (6) When SPS is used, if P ≥ 81 or P=0, only NULL is output.	<Operation> STRING A B C D E F G H I J EN=1 UINT 5 (Second input) UINT 3 (Third input) ⇩ Extract STRING C D E F G ENO=1 Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

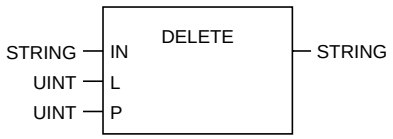
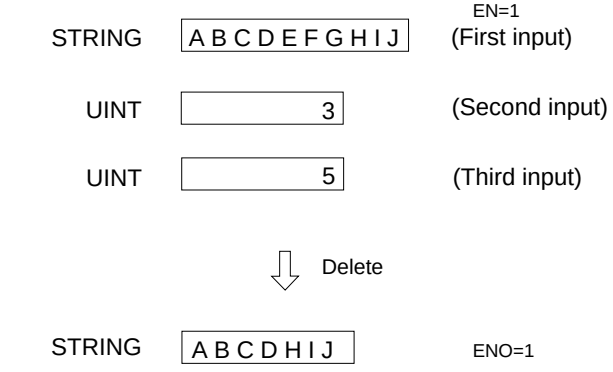
(5) Concatenate CONCAT

Name, Symbol, Function	Example
Name CONCAT	<Operation> STRING A B C D E F G H I J (First input) EN=1 STRING X Y Z (Second input) ⇩ Concatenate STRING A B C D E F G H I J X Y Z ENO=1
Symbol  	
Function (1) CONCAT concatenates the string designated by the second input to the first input of STRING type. (2) When SPH is used, if the number of characters to be concatenated exceeds 64 characters, only the first 64 characters are output and ENO is set to 0. (3) Only NULL is output and ENO is set to 0 if the number of concatenating characters = 0. (4) When SPS is used, if the number of concatenated characters exceeds 80, the first 80 characters are output.	Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

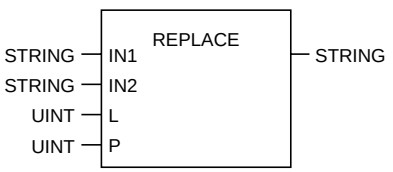
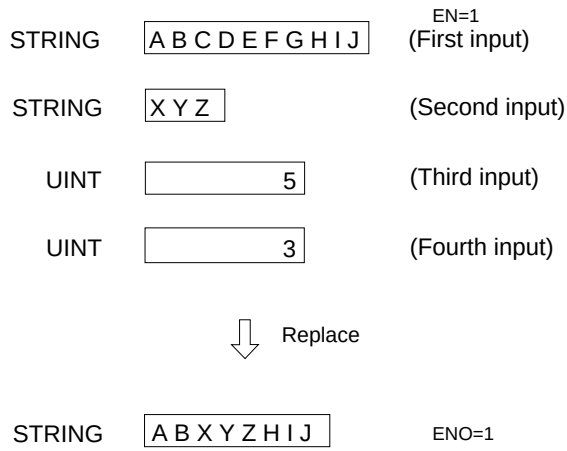
(6) Insert string INSERT

Name, Symbol, Function	Example
Name INSERT	<Operation> STRING A B C D E F G H I J (First input) EN=1 STRING X Y Z (Second input) UINT 4 (Third input) ⇩ Insert STRING A B C D X Y Z E F G H I J ENO=1
Symbol  	
Function (1) INSERT inserts the string designated by the second input into the first input of STRING type at the Pth character position designated by the third input. (2) When SPH is used, if the number of characters to be inserted exceeds 64 characters, only the first 64 characters are output and ENO is set to 0. (3) The first input is output and ENO is set to 0 if the number of characters of the first input < P. (4) If P = 0, the second input is inserted before the first input and ENO is set to 1. (5) When SPS is used, if the number of combined characters exceeds 80, the first 80 characters after insertion are output.	Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

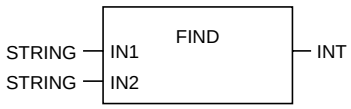
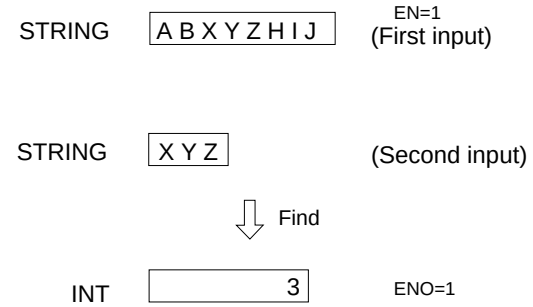
(7) Delete string DELETE

Name, Symbol, Function	Example
Name DELETE **Symbol**  **Function** (1) DELETE deletes the number of characters designated by the second input from the first input of STRING type starting at the Pth position (third input). (2) When SPH is used, if P ≥ 65 or P = 0, the output is NULL and ENO is set to 0. (3) If the number of characters of the first input < P, the first input is output and ENO is set to 0. (4) When SPH is used, if the output is longer than 65 characters, the first 64 characters are output and ENO is set to 0. (5) If L = 0, the first input is output as is and ENO is set to 1. (6) When SPS is used, if P ≥ 81 or P=0, the output is NULL. (7) When SPS is used, if 81 characters or more are to be output, only the first 80 characters are output.	<Operation>  Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

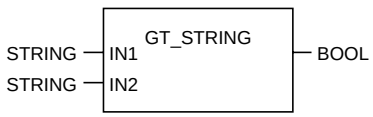
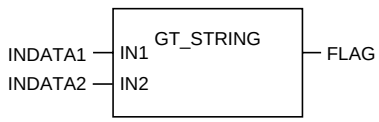
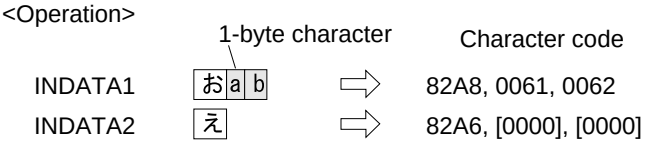
(8) Replace string REPLACE

Name, Symbol, Function	Example
Name REPLACE **Symbol**  **Function** (1) Deletes the indicated number of charactersin the third input starting from the Pth one (the fourth input) of the STRING type data in the first input and inserts the string in the second input. (2) When SPH is used, if the number of replaced characters exceeds 64 characters, the first 64 characters are stored and ENO is set to 0. (3) When SPH is used, if P ≥ 65 or P = 0, the output is NULL and ENO is set to 0. (4) If the number of characters of the first input < P, the first input is output and ENO is set to 0. (5) If L = 0, the second input is inserted starting at the Pth character position and ENO is set to 1. (6) When SPS is used, if the number of characters exceeds 80 after replacement, only the first 80 characters are output.	<Operation>  Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

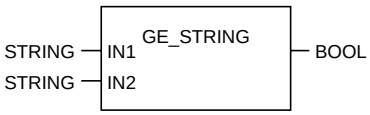
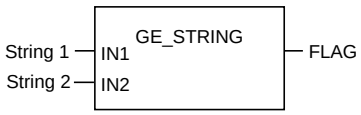
(9) Find string FIND

Name, Symbol, Function	Example
Name: FIND Symbol:  Function: (1) FIND searches the first input of STRING type for the same string designated by the second input and outputs the position of the first occurrence of the string. (2) A “0” is output if no matching string is found.	<Operation>  Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

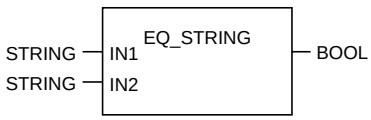
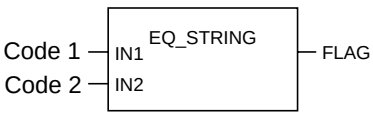
(10) Compare string (>) GT_STRING

Name, Symbol, Function	Example
Name: GT_STRING Symbol:  Function: (1) GT_STRING outputs a 1 if the condition first input > second input is met. It outputs a 0 if the above condition is not met. (2) The result of executing a compare instruction is BOOL type data (1 or 0). (3) Available data type is STRING type. (4) GT_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Programming example>  <Operation>  Consequently, INDATA1 > INDATA2 holds, and FLAG is set to 1. Note: 1) The function regards a 1-byte character as a 16-bit code by placing an 8-bit “00” in the higher-order byte position of the 16-bit code. 2) If the strings to compare have different lengths, “0000” is appended to the shorter string from the right side. 3) The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

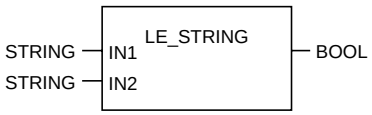
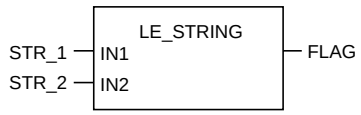
(11) Compare string (≥) GE_STRING

Name, Symbol, Function	Example
Name: GE_STRING	<Programming example>
Symbol: 	
Function: (1) GE_STRING outputs a 1 if the condition first input ≥ second input is met. It outputs a 0 if the above condition is not met. (2) Available data type is STRING type. (3) GE_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Operation> • When the input data is of TIME type Character code String 1 はる ⇨ 82CD82E9 String 2 なつ ⇨ 82C882C2 Consequently, String 1 ≥ String 2 holds, and FLAG is set to 1. Note: The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

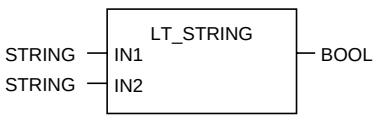
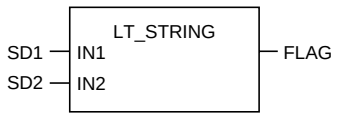
(12) Compare string (=) EQ_STRING

Name, Symbol, Function	Example
Name: EQ_STRING	<Programming example>
Symbol: 	
Function: (1) EQ_STRING outputs a 1 if the condition first input = second input is met. It outputs a 0 if the above condition is not met. (2) Available data type is STRING type. (3) GT_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Operation> Code 1 ABCD Code 2 ABCD Consequently, Code 1 = Code 2 holds, and FLAG is set to 1. Note: The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

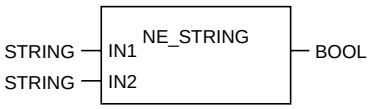
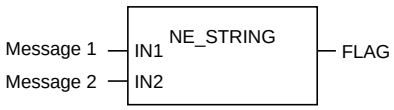
(13) Compare string (≤) LE_STRING

Name, Symbol, Function	Example
Name LE_STRING **Symbol**  **Function** (1) LE_STRING outputs a 1 if the condition first input ≤ second input is met. It outputs a 0 if the above condition is not met. (2) Available data type is STRING type. (3) LE_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Programming example>  <Operation> STR_1 あお ⇨ 82A0, 82A7 STR_2 あか ⇨ 82A0, 82A9 Consequently, STR_1 ≤ STR_2 holds, and FLAG is set to 1. Note: The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

(14) Compare string (<) LT_STRING

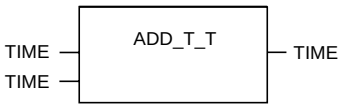
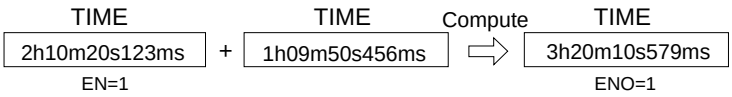
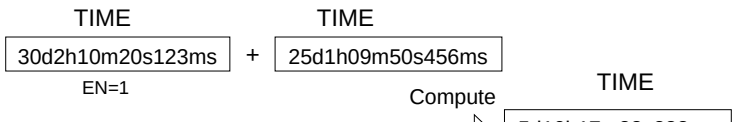
Name, Symbol, Function	Example
Name LT_STRING **Symbol**  **Function** (1) LT_STRING outputs a 1 if the condition first input < second input is met. It outputs a 0 if the above condition is not met. (2) Available data type is STRING type. (3) LT_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Programming example>  <Operation> SD1 D ⇨ 8263 SD2 E ⇨ 8264 Consequently, SD1 < SD2 holds, and FLAG is set to 1. Note: The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

(15) Compare string (≠) NE_STRING

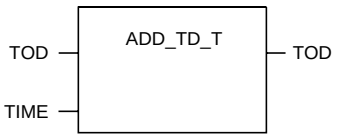
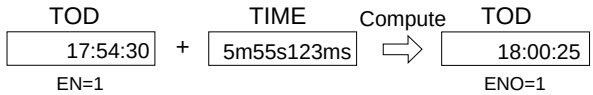
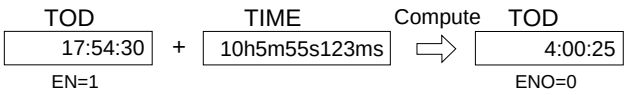
Name, Symbol, Function		Example
Name	NE_STRING	<Programming example>
Symbol		
Function	(1) NE_STRING outputs a 1 if the condition first input ≠ second input is met. It outputs a 0 if the above condition is not met. (2) Available data type is STRING type. (3) LE_STRING converts the string into a sequence of character codes and compares the character codes as numeric data.	<Operation> Message 1 ON Compute ⇨ FLAG 1 Message 2 OFF Consequently, Message 1 ≠ Message 2 holds, and FLAG is set to 1. Note: The SPS compares characters by shifted JIS code. Note: No. of characters for STRING type when SPH is used: 0 to 64 (single-byte, double-byte) No. of characters for STRING type when SPS is used: 0 to 80 (single-byte), 0 to 40 (double-byte)

2-5-8 Time type data functions

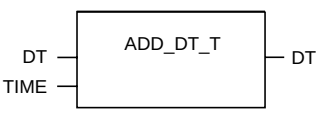
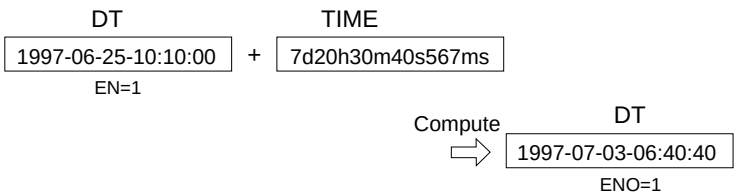
(1) Add time ADD_T_T

Name, Symbol, Function	Example
Name: ADD_T_T	<Operation> • When the operation results falls within the limit value range of the data type
Symbol: 	
Function: (1) ADD_T_T adds together TIME type data. (2) The overflow condition is ignored and ENO is set to 0 if the operation result exceeds the limit value range of the TIME type.	• When the operation result exceeds the limit value range of the data type  [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

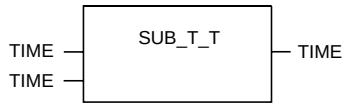
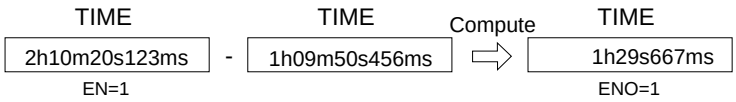
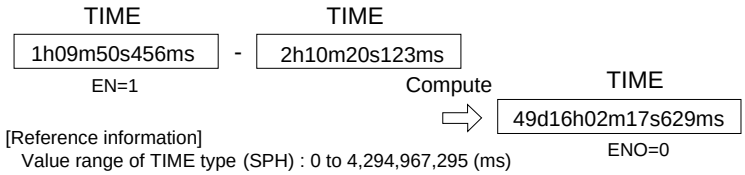
(2) Add time ADD_TD_T (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name: ADD_TD_T	<Operation> • When the operation results falls within the limit value range of the data type
Symbol: 	
Function: (1) ADD_TD_T adds together TOD type data and TIME type data. (2) The ms fraction of the TIME type data is truncated. (3) If the operation result exceeds the limit value range of the TOD type, its 24-hour system equivalent is output and ENO is set to 0. (Limit values: 0, 23:59:59)	• When the operation result exceeds the limit value range of the data type 

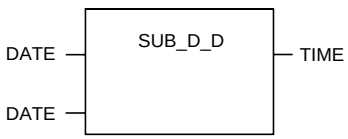
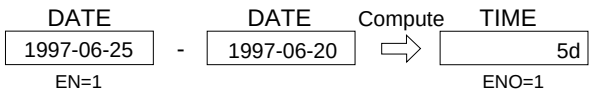
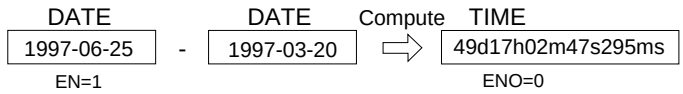
(3) Add time ADD_DT_T (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name: ADD_DT_T	<Operation>
Symbol: 	
Function: (1) ADD_DT_T adds together DT type data and TIME type data. (2) The ms fraction of the TIME type data is truncated. (3) The overflow condition is ignored and ENO is set to 0 if the operation result exceeds the limit value range of the DT type.	

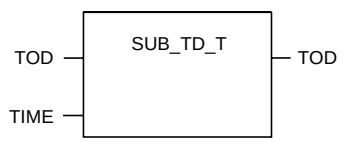
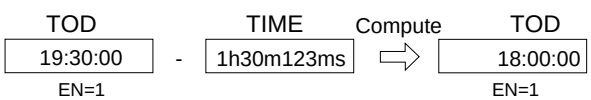
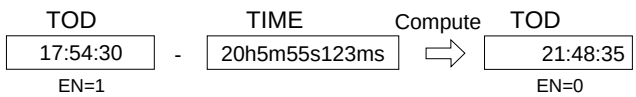
(4) Subtract time SUB_T_T

Name, Symbol, Function	Example
Name: SUB_T_T Symbol:  Function: (1) SUB_T_T performs a subtraction on TIME type data. (2) The underflow condition is ignored and ENO is set to 0 if the operation result exceeds the limit value range of the TIME type. (Limit values: 0, 49d17h02m47s295ms)	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type  [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

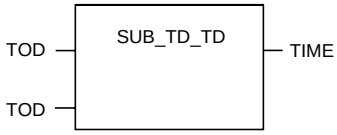
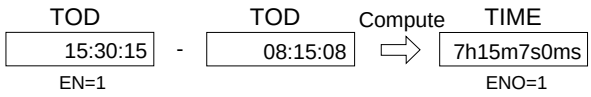
(5) Subtract time SUB_D_D (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name: SUB_D_D Symbol:  Function: (1) SUB_D_D performs a subtraction on DATE type data and outputs TIME type data. (2) If the operation result exceeds the positive limit value of the data type, the limit value of the data type is output and ENO is set to 0. (3) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case.	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type 

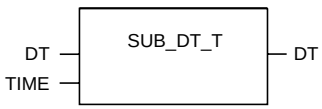
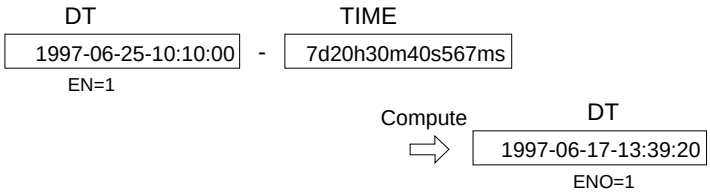
(6) Subtract time SUB_TD_T (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name: SUB_TD_T Symbol:  Function: (1) SUB_TD_T performs a subtraction on TOD type data and TIME type data and outputs TOD type data. (2) The ms fraction of the TIME data is truncated. (3) If the operation result exceeds the limit value range of the TOD type, its 24-hour system equivalent is output and ENO is set to 0. (Limit values: 00:00:00, 23:59:59)	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type 

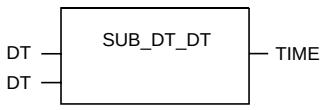
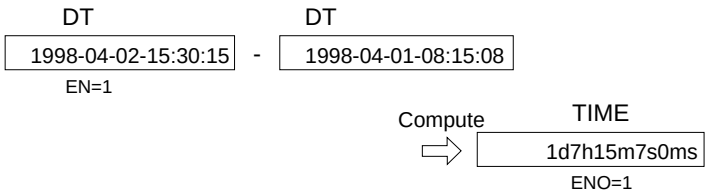
(7) Subtract time SUB_TD_TD (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	SUB_TD_TD	<Operation>
Symbol		
Function	(1) SUB_TD_TD performs a subtraction on TOD type data and outputs TIME type data. (2) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case.	

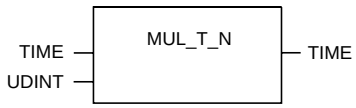
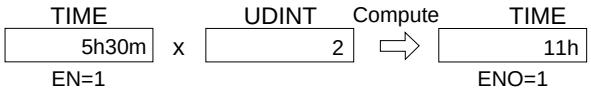
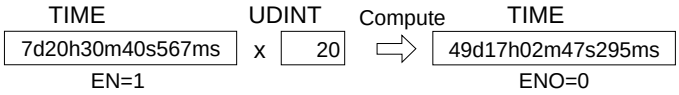
(8) Subtract time SUB_DT_T (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	SUB_DT_T	<Operation>
Symbol		
Function	(1) SUB_DT_T performs a subtraction on DT type data and TIME type data and outputs DT type data. (2) The ms fraction of the TIME data is truncated. (3) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case.	

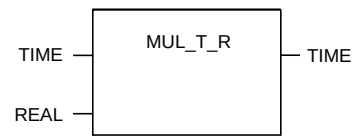
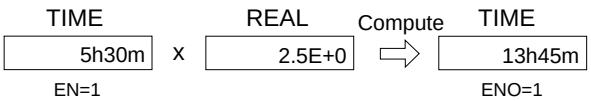
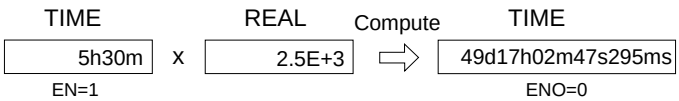
(9) Subtract time SUB_DT_DT (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	SUB_DT_DT	<Operation>
Symbol		
Function	(1) SUB_DT_DT performs a subtraction on DT type data and outputs TIME type data. (2) When the result exceeds the positive data type boundary value, a TIME type boundary value is output, resulting in ENO = 0. (3) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case.	

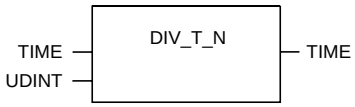
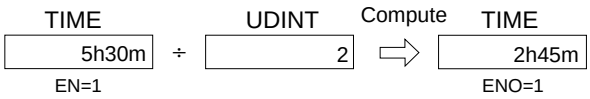
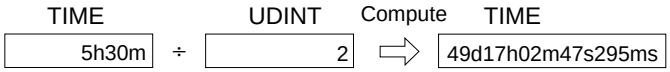
(10) Multiply time MUL_T_N

Name, Symbol, Function	Example
Name: MUL_T_N Symbol:  Function: (1) MUL_T_N multiplies TIME type data by the second input data. (2) If the operation result exceeds the limit value range of the data type, the limit value of the data type is output and ENO is set to 0.	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type  Note: When SPS is used, even if the operation result exceeds the boundary value of the data type, no boundary processing is performed. Be careful so that no overflow will occur. [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

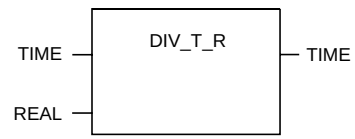
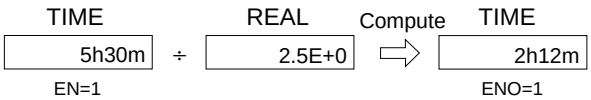
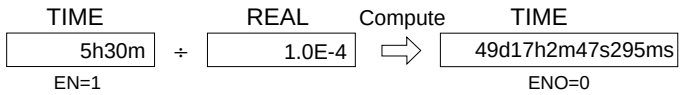
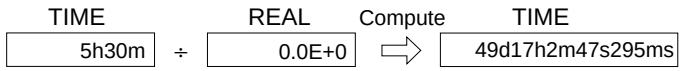
(11) Multiply time MUL_T_R

Name, Symbol, Function	Example
Name: MUL_T_R Symbol:  Function: (1) MUL_T_R multiplies TIME type data by the second input data. (2) When the result exceeds the positive data type boundary value, a TIME type boundary value is output, resulting in ENO = 0. (3) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case.	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type  Note: In this multiplication, TIME type data is converted to REAL type data and the result is converted to TIME type again. Consequently, the precision of the operation result is identical to that of REAL type. [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

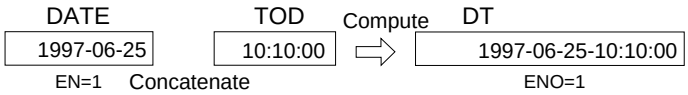
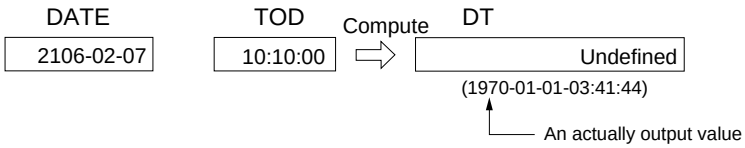
(12) Divide time DIV_T_N

Name, Symbol, Function	Example
Name: DIV_T_N Symbol:  Function: (1) DIV_T_N performs a division on TIME type data. (2) Any digits in the result less than ms-unit digits are truncated. (3) When a divider is 0, a TIME type of boundary value is output, resulting in ENO = 0. Note: When SPS is used, if divisor is 0 (zero), the SPS stops operation.	<Operation> • When the operation results falls within the limit value range of the data type  • When a divider is 0  [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

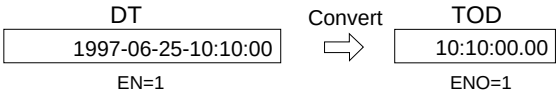
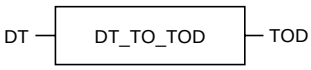
(13) Divide time DIV_T_R

Name, Symbol, Function	Example
Name: DIV_T_R Symbol:  Function: (1) DIV_T_R multiplies TIME type data by the second input data. (2) If the operation result exceeds the limit value range of the data type, the limit value of the data type is output and ENO is set to 0. (Limit value: 49d17h02m47s295ms) (3) Correct output value is not guaranteed if the operation result is negative. ENO is set to 0 in this case. (4) The ms fraction of the operation result is rounded off. (5) When a divider is 0, a TIME type of boundary value is output, resulting in ENO = 0.	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type  Note: In this multiplication, TIME type data is converted to REAL type data and the result is converted to TIME type again. Consequently, the precision of the operation result is identical to that of REAL type. • When a divider is 0  [Reference information] Value range of TIME type (SPH) : 0 to 4,294,967,295 (ms) (SPS) : 0 to 2,147,483,647 (ms)

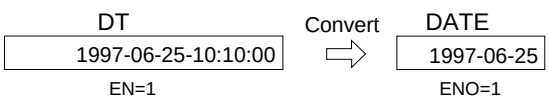
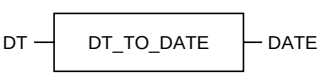
(14) Concatenate time CONCAT_D_D (These functions are not supported in SPS.)

Name, Symbol, Function		Example
Name	CONCAT_D_D	<Operation> • When the operation results falls within the limit value range of the data type  • When the operation result exceeds the limit value range of the data type 
Symbol	 	
Function	(1) CONCAT_D_D concatenates TOD type data to DATE type data and outputs DT type data. (2) Correct output value is not guaranteed if the operation result exceeds the limit value range of the data type. ENO is set to 0 in this case.	

(15) Convert DT to TOD DT_TO_TOD (These functions are not supported in SPS.)

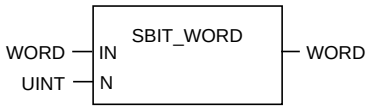
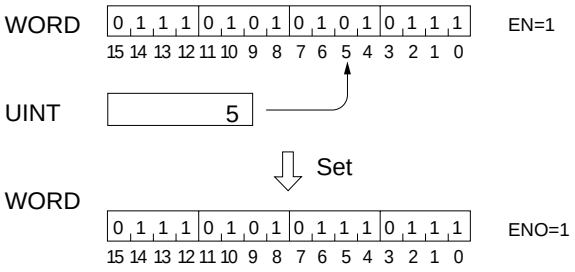
Name, Symbol, Function		Example
Name	DT_TO_TOD	<Operation> 
Symbol	 	
Function	(1) DT_TO_TOD extracts the time part of DT type data and outputs TOD type data.	

(16) Convert DT to DATE DT_TO_DATE (These functions are not supported in SPS.)

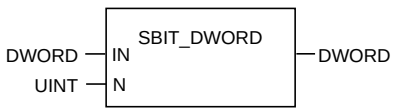
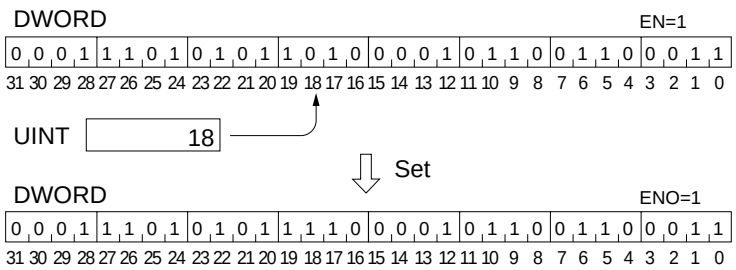
Name, Symbol, Function		Example
Name	DT_TO_DATE	<Operation> 
Symbol	 	
Function	(1) DT_TO_DATE extracts the date part of DT type data and outputs DATE type data.	

2-5-9 Original FCTs (Functions)

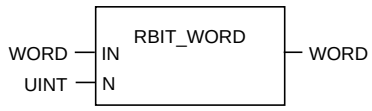
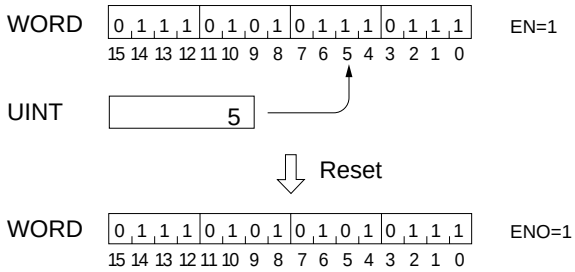
(1) Set bit SBIT_WORD

Name, Symbol, Function		Example
Name	SBIT_WORD	<Operation> • When bit “N” is read normally
Symbol		
Function	(1) SBIT_WORD sets the “Nth” bit of “IN” (WORD type) and outputs the resulting WORD type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 4 bits of “N” are significant. For example, bit 0 is set when N = 16 and bit 1 is set when N = 17.	

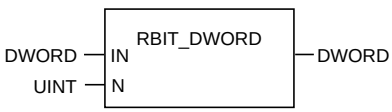
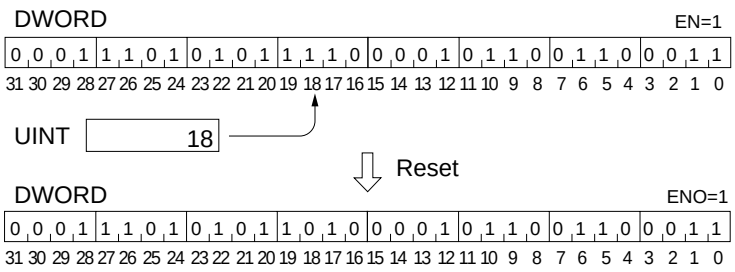
(2) Set bit SBIT_DWORD

Name, Symbol, Function		Example
Name	SBIT_DWORD	<Operation> • When bit “N” is read normally
Symbol		
Function	(1) SBIT_DWORD sets the “Nth” bit of “IN” (WORD type) and outputs the resulting DWORD type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 5 bits of “N” are significant. For example, bit 0 is set when N = 32 and bit 1 is set when N =33.	

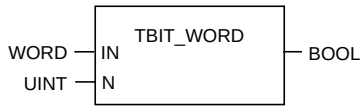
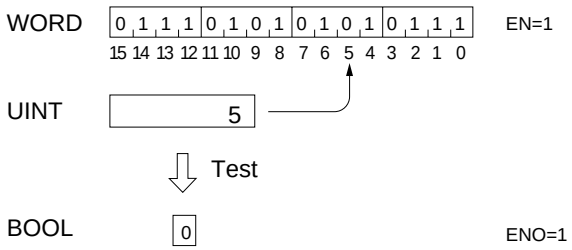
(3) Reset bit RBIT_WORD

Name, Symbol, Function	Example
Name RBIT_WORD	<Operation> • When bit “N” is read normally
Symbol 	
Function (1) RBIT_WORD resets the “Nth” bit of “IN” (WORD type) and outputs the resulting WORD type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 4 bits of “N” are significant. For example, bit 0 is reset when N = 16 and bit 1 is reset when N =17.	

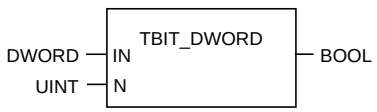
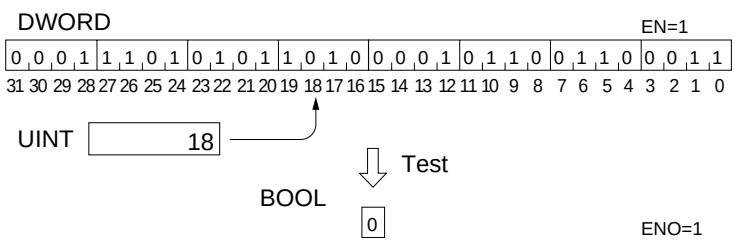
(4) Reset bit RBIT_DWORD

Name, Symbol, Function	Example
Name RBIT_DWORD	<Operation> • When bit “N” is read normally
Symbol 	
Function (1) RBIT_DWORD resets the “Nth” bit of “IN” (WORD type) and outputs the resulting DWORD type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 5 bits of “N” are significant. For example, bit 0 is reset when N = 32 and bit 1 is reset when N =33.	

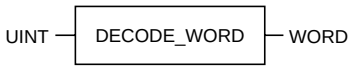
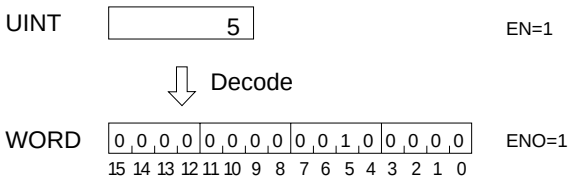
(5) Test bit TBIT_WORD

Name, Symbol, Function		Example
Name	TBIT_WORD	<Operation> • When bit “N” is read normally
Symbol		
Function	(1) TBIT_WORD tests (1 or 0) the “Nth” bit of “IN” (WORD type) and outputs a BOOL type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 4 bits of “N” are significant. For example, bit 0 is tested when N = 16 and bit 1 is tested when =17.	

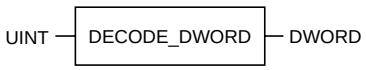
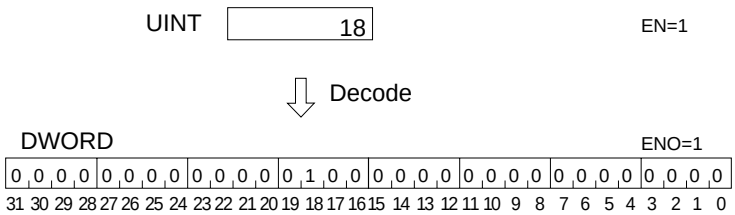
(6) Test bit TBIT_DWORD

Name, Symbol, Function		Example
Name	TBIT_DWORD	<Operation> • When bit “N” is read normally
Symbol		
Function	(1) TBIT_DWORD tests (1 or 0) the “Nth” bit of “IN” (DWORD type) and outputs a BOOL type data. (2) Available data type for “IN” is UNIT type. (3) The lowest-order 5 bits of “N” are significant. For example, bit 0 is tested when N = 32 and bit 1 is tested when N = 33.	

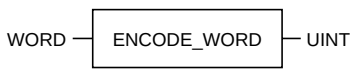
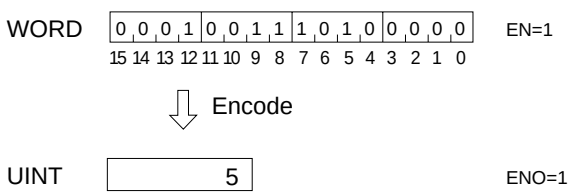
(7) Decode DECODE_WORD

Name, Symbol, Function		Example
Name	DECODE_WORD	<Operation>
Symbol		
Function	(1) DECODE_WORD outputs WORD type data with the bit position designated by the input (UINT type data) set on. (2) The lowest-order 4 bits of input are significant. For example, bit 0 is set on when the input is 16 and bit 1 is set on when the input is 17.	

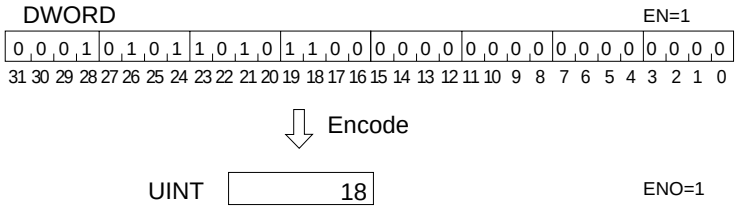
(8) Decode DECODE_DWORD

Name, Symbol, Function		Example
Name	DECODE_DWORD	<Operation>
Symbol		
Function	(1) DECODE_DWORD outputs DWORD type data with the bit position designated by the input (UINT type data) set on. (2) The lowest-order 5 bits of the input are significant. For example, bit 0 is set on when the input is 32 and bit 1 is set on when the input is 33.	

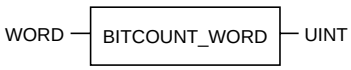
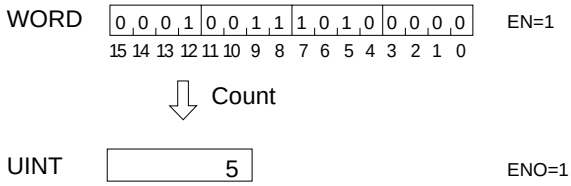
(9) Encode ENCODE_WORD

Name, Symbol, Function		Example
Name	ENCODE_WORD	<Operation>
Symbol		
Function	(1) ENCODE_WORD outputs a UINT type data that indicates the lowest-order bit position of the input (WORD type data) that is set on. (2) 16 is output if there is no bit in the input word that is set on.	

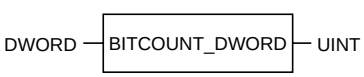
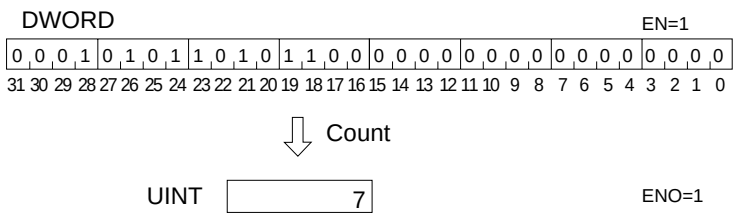
(10) Encode ENCODE_DWORD

Name, Symbol, Function		Example
Name	ENCODE_DWORD	<Operation>
Symbol		
Function	(1) ENCODE_DWORD outputs a UINT type data that indicates the lowest-order bit position of the input (DWORD type data) that is set on. (2) 32 is output if there is no bit in the input word that is set on.	

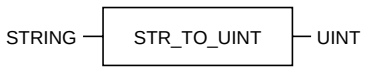
(11) Bit count BITCOUNT_WORD

Name, Symbol, Function		Example
Name	BITCOUNT_WORD	<Operation>
Symbol		
Function	(1) BITCOUNT_WORD outputs a UINT type data that indicates the number of on bits in the input (WORD type data).	

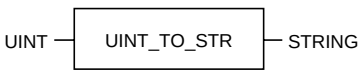
(12) Bit count BITCOUNT_DWORD

Name, Symbol, Function		Example
Name	BITCOUNT_DWORD	<Operation>
Symbol		
Function	(1) BITCOUNT_DWORD outputs a UINT type data that indicates the number of on bits in the input (DWORD type data).	

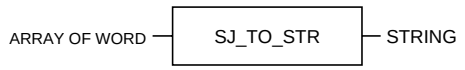
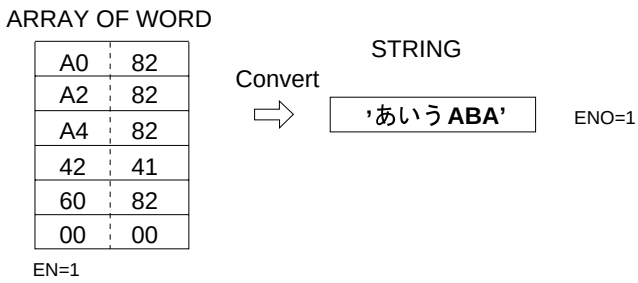
(13) Convert string to number STR_TO_UINT

Name, Symbol, Function	Example
Name: STR_TO_UINT	<Operation> STRING '1234' EN=1 ⇩ Convert UINT 1234 ENO=1 • When the input data contains a non-numeric character STRING '123F' EN=1 ⇩ Convert UINT 0 ENO=0 • When the input data exceeds the limit value range of UINT type STRING '123456' EN=1 ⇩ Convert UINT 65535 ENO=0
Symbol: 	
Function: (1) STR_TO_UINT converts STRING type data to UINT type data. (2) The STRING type data must be made up of 1-byte numerals from “0” to “9.” A 0 is output and ENO is set to 0 if the STRING type data contains a character other than “0” through “9.” (3) If the conversion result exceeds the limit value range of the UINT type, the upper limit value of the UINT type is output and ENO is set to 0. (Upper limit value: 65535) (4) 2-byte characters are not regarded as numerals. The output is set to 0 and ENO is set to 0 if a 2-byte numeral is encountered.	

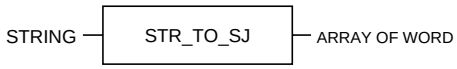
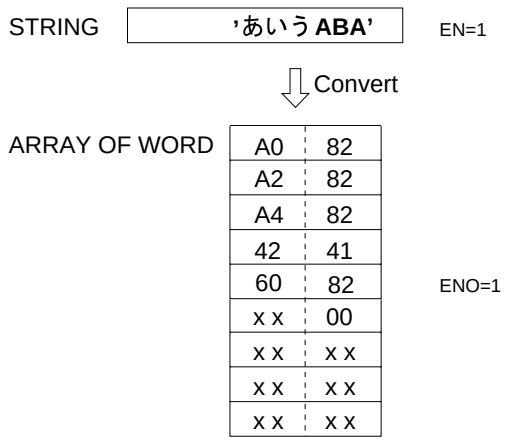
(14) Convert number to string UINT_TO_STR

Name, Symbol, Function	Example
Name: UINT_TO_STR	<Operation> UINT 1234 EN=1 ⇩ Convert STRING '1234' ENO=1
Symbol: 	
Function: (1) UINT_TO_STR converts UINT type data to STRING type data (1-byte numerals).	

(15) Convert shift-JIS to string SJ_TO_STR (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name SJ_TO_STR **Symbol**  **Function** (1) SJ_TO_STR converts Shift-JIS code defined as an array of WORD type data to STRING type data. (2) The Shift-JIS code input must be terminated by a NULL code (00 or 00 00). (3) If 64 or more Shift-JIS codes are input, the first 64 codes are converted to STRING type data and ENO is set to 0.	<Operation>  Note: No check is made to determine whether the input code is a sequence of valid Shift-JIS codes. Setting of ENO = 1 remains unchanged and an output is undefined.

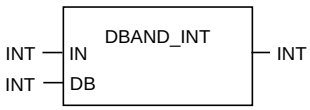
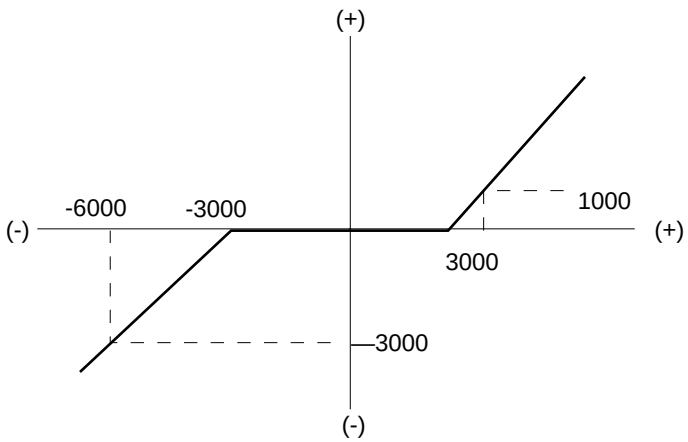
(16) Convert string to shift-JIS STR_TO_SJ (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name STR_TO_SJ **Symbol**  **Function** (1) STR_TO_SJ converts STRING type data to Shift-JIS code which is defined as an array of WORD type data. A null code is appended to the end of the output. (2) The destination array area may contain garbage data after the null code if the size of input is smaller than that of the array area. (3) If the size of the destination array area is smaller than that of the input, as many Shift-JIS codes as fit in the array area are output with a null code at the end of the array. ENO is also set to 0.	<Operation>  Note: Note: Since a null code is placed at the end of the destination array area, the size of the array area needs to be greater than the number of input characters by 1 word.

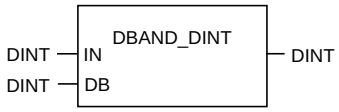
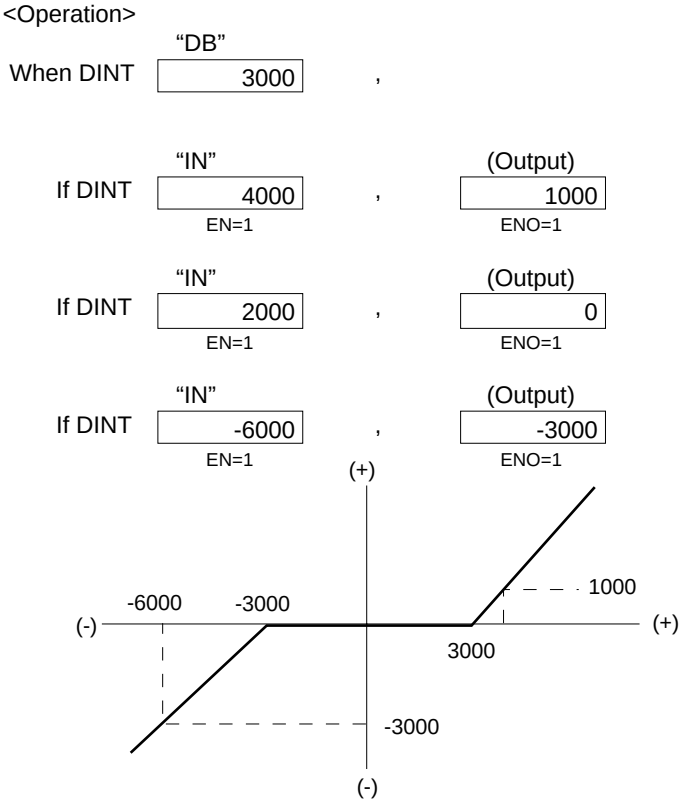
(17) Byte length BYTE_LEN

Name, Symbol, Function	Example
Name: BYTE_LEN Symbol:  Function: (1) BYTE_LEN outputs an INT type data that indicates the number of bytes of STRING type data as converted to Shift-JIS code.	<Operation> STRING ' あいう ab えお' EN=1 ⇩ Convert INT 12 ENO=1 Note: The size of a 1-byte character is 1 and that of a 2-byte character is 2.

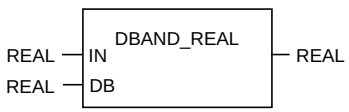
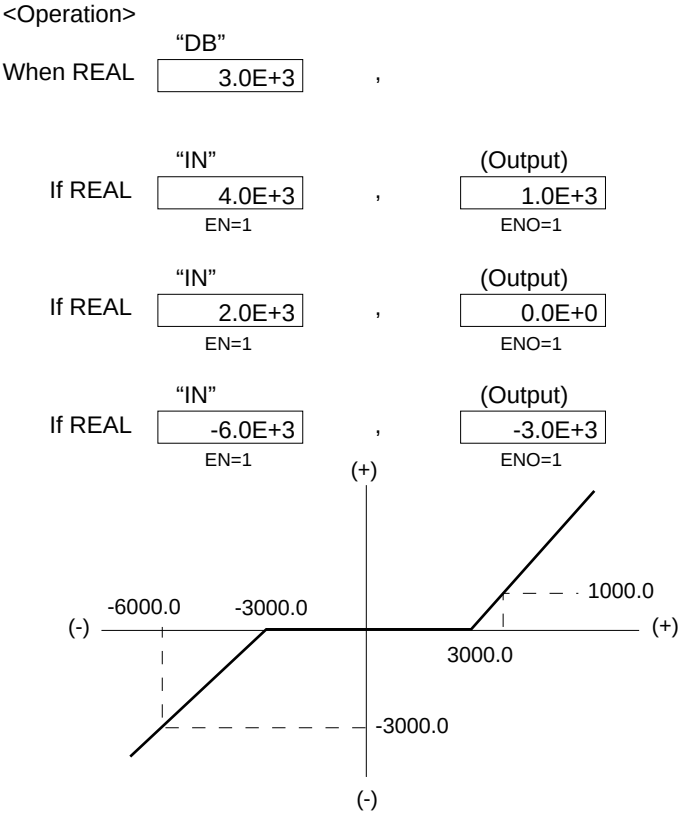
(18) Dead band DBAND_INT

Name, Symbol, Function	Example
Name: DBAND_INT Symbol:  Function: (1) DBAND_INT processes the input “DB” (INT type) as representing a dead band. It outputs IN - |DB| if IN > |DB|, IN + |DB| if IN < -|DB|, and a “0” in other cases.	<Operation> “DB” When INT 3000 “IN” (Output) if INT 4000 EN=1 , 1000 ENO=1 if INT “IN” 2000 EN=1 , (Output) 0 ENO=1 if INT “IN” -6000 EN=1 , (Output) -3000 ENO=1 

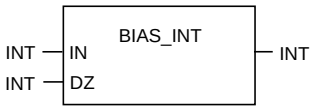
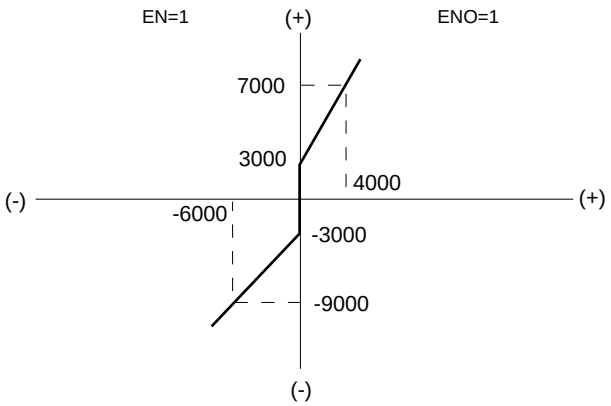
(19) Dead band DBAND_DINT

Name, Symbol, Function		Example
Name	DBAND_DINT	<Operation>
Symbol		
Function	(1) DBAND_DINT processes the input "DB" (DINT type) as representing a dead band. It outputs IN - |DB| if IN > |DB|, IN + |DB| if IN < -|DB|, and a "0" in the other cases.	

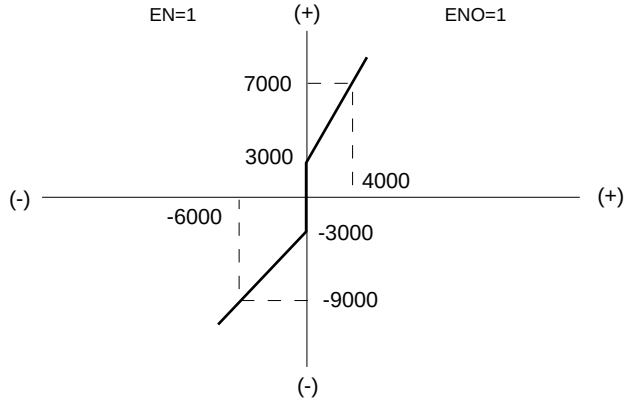
(20) Dead band DBAND_REAL

Name, Symbol, Function		Example
Name	DBAND_REAL	<Operation>
Symbol		
Function	(1) DBAND_REAL processes the input "DB" (REAL type) as representing a dead band. It outputs IN - |DB| if IN > |DB|, IN + |DB| if IN < -|DB|, and a "0" in the other cases. (2) The output value has an error which is similar to those which occur during REAL data additions or subtractions.	

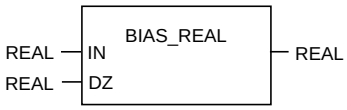
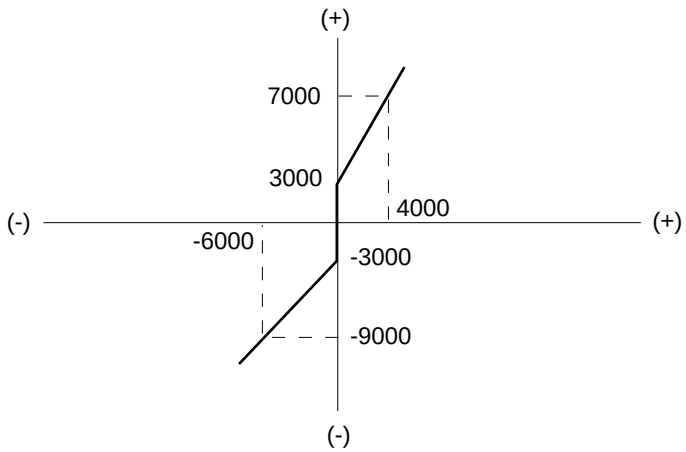
(21) Bias BIAS_INT

Name, Symbol, Function	Example
Name: BIAS_INT Symbol:  Function: (1) BIAS_INT processes the “DZ” (INT type) as representing a bias. It outputs IN + |DZ| if IN > 0, IN - |DZ| if IN < 0, and a “0” in the other cases. (2) The upper limit value of INT type is output if the operation result exceeds the upper limit value of INT type and the lower limit value is output if the operation result falls below the lower limit value. In these cases, ENO is set to 0.	<Operation> When INT “DZ” 3000 , If INT “IN” 4000 EN=1 , (Output) 7000 ENO=1 If INT “IN” 0 EN=1 , (Output) 0 ENO=1 If INT “IN” -6000 EN=1 , (Output) -9000 ENO=1 

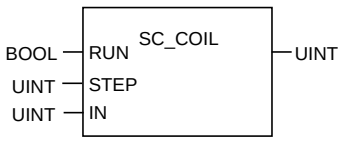
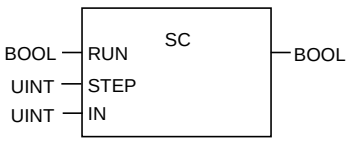
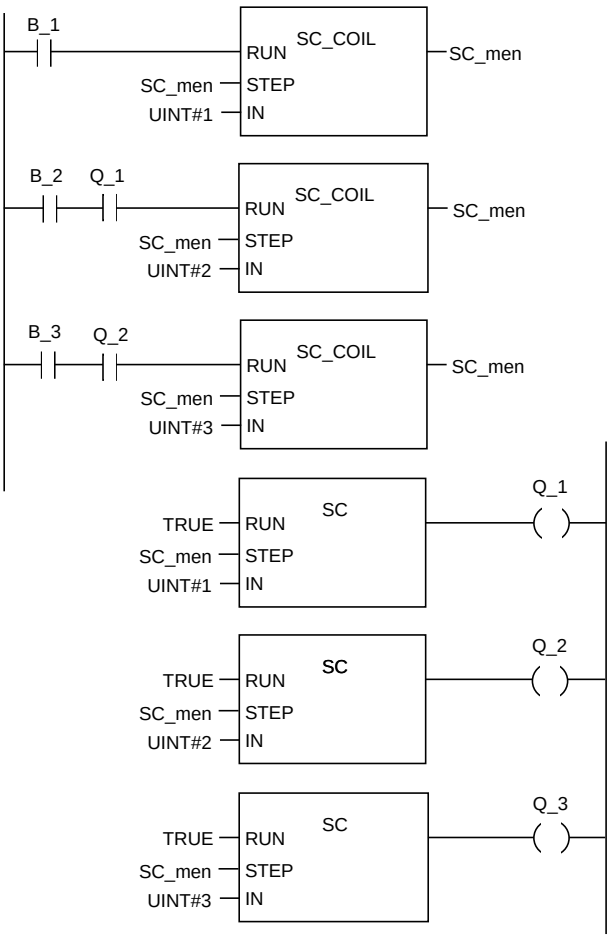
(22) Bias BIAS_DINT

Name, Symbol, Function	Example
Name: BIAS_DINT Symbol:  Function: (1) BIAS_DINT processes the “DZ” (DINT type) as representing a bias. It outputs IN + |DZ| if IN > 0, IN - |DZ| if IN < 0, and a “0” in the other cases. (2) The upper limit value of DINT type is output if the operation result exceeds the upper limit value of DINT type and the lower limit value is output if the operation result falls below the lower limit value. In these cases, ENO is set to 0.	<Operation> When DINT “DZ” 3000 , If DINT “IN” 4000 EN=1 , (Output) 7000 ENO=1 If DINT “IN” 0 EN=1 , (Output) 0 ENO=1 If DINT “IN” -6000 EN=1 , (Output) -9000 ENO=1 

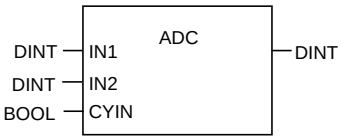
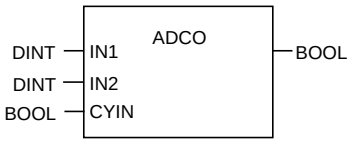
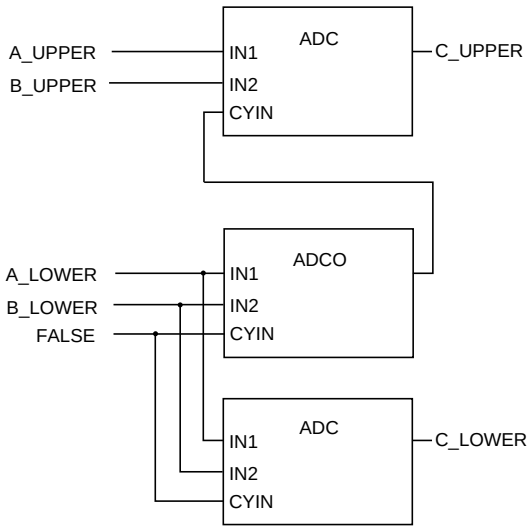
(23) Bias BIAS_REAL

Name, Symbol, Function	Example
Name BIAS_REAL	<Operation>
Symbol 	When REAL "DZ" 3.0E+3 ,
Function (1) BIAS_REAL processes the "DZ" (REAL type) as representing a bias. It outputs IN + |DZ| if IN > 0, IN - |DZ| if IN < 0, and a "0" in the other cases. (2) The upper limit value of REAL type is output if the operation result exceeds the upper limit value of REAL type and the lower limit value is output if the operation result falls below the lower limit value. In these cases, ENO is set to 0. (3) The output value has an error which is similar to those which occur during REAL data additions or subtractions.	If REAL "IN" 4.0E+3 EN=1 , (Output) 7.0E+3 ENO=1 If REAL "IN" 0.0E+0 EN=1 , (Output) 0.0E+0 ENO=1 If REAL "IN" -6.0E+3 EN=1 , (Output) -9.0E+3 ENO=1 

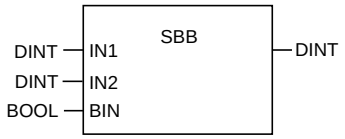
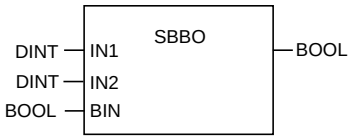
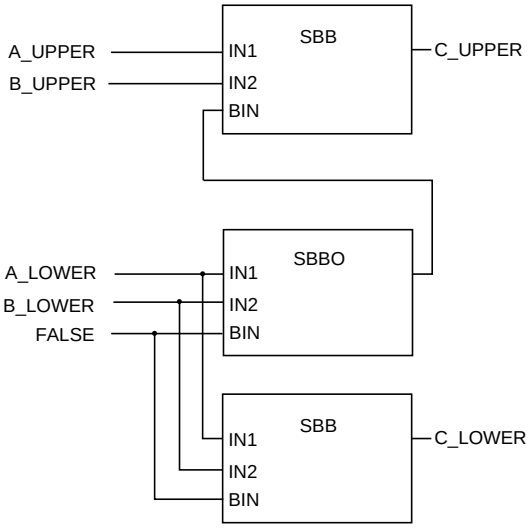
(24) Step sequence SC_COIL/SC

Name, Symbol, Function	Example
Name SC_COIL/SC	<Programming example> An example of creating a sequential circuit is shown below. Step execution proceeds in the order of Q1, Q2, and Q3.
Symbol <Step sequence coil>  <Step sequence bit>  	
Function Step sequence controls have four characteristics: self holding, interlock, power-off step retention, and subsequence priority. <Step sequence coil> (1) If “RUN” is 0 the value of “STEP” is output. If “RUN” is 1, the value of “IN” is output. <Step sequence bit> (1) If “RUN” is 1, “STEP” and “IN” are matched and a 1 is output if a match occurs. A 0 is output if no match occurs. Note: To implement the power-off step retention function, it is necessary to assign the variable “STEP” to retain memory.	

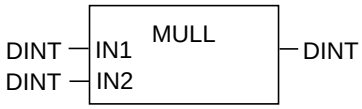
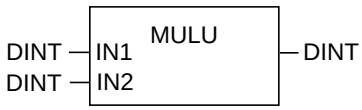
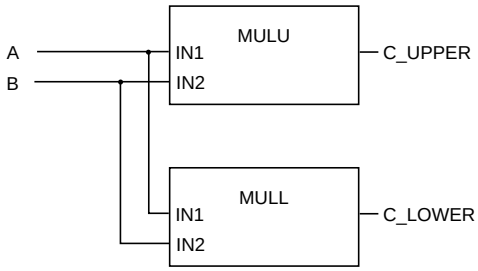
(25) 32-bit addition with carry ADC/ADCO

Name, Symbol, Function	Example
Name ADC/ADCO	<Programming example> A sample program for adding 64-bit operands is shown below.
Symbol <32-bit addition with carry>  <32-bit carry>  	 The maximum length of data that can be manipulated in a MICREX-SX system is 32 bits. Consequently, when adding 64-bit data items, it is necessary to regard each data item as consisting of two 32-bit data items. In the above figure, 64-bit data items A and B are added and the result is output in C. The upper-order 32 bits of A, B, and C are assigned to A_UPPER, B_UPPER, and C_UPPER, respectively, and their lower-order 32 bits are assigned to A_LOWER, B_LOWER, and C_LOWER, respectively. The most significant bit of the lower-order 32 bits is handled not as a sign but as a numeric value. [Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.
Function The ADC and ADCO instructions are used in combination to add together 64-bit or wider data. These instructions cannot be used independently. <32-bit addition with carry> (1) ADC adds DINT type “IN1” and “IN2” with carry “CYIN.” <32-bit carry> (1) ADCO adds DINT type “IN1” and “IN2” with carry“CYIN” and outputs a carry.	

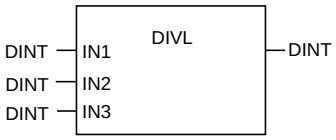
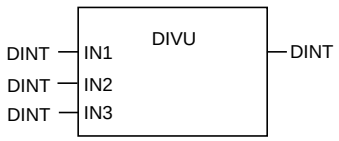
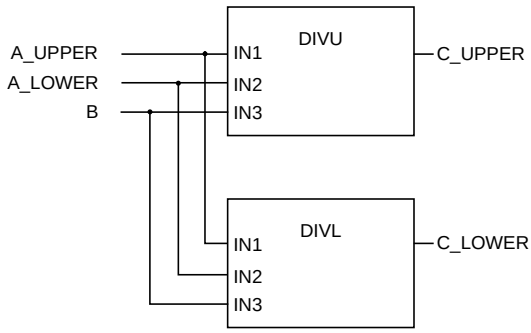
(26) 32-bit subtraction with borrow SBB/SBBO

Name, Symbol, Function	Example
Name SBB/SBBO	<Programming example> A sample program for performing a subtraction on 64-bit operands is shown below.
Symbol <32-bit subtraction with borrow>  <32-bit borrow>  	 The maximum length of data that can be manipulated in a MICREX-SX system is 32 bits. Consequently, when performing a subtraction 64-bit data items, it is necessary to regard each data item as consisting of two 32-bit data items. In the above figure, a subtraction is performed on 64-bit data items A and B and the result is output in C. The upper-order 32 bits of A, B, and C are assigned to A_UPPER, B_UPPER, and C_UPPER, respectively, and their lower-order 32 bits are assigned to A_LOWER, B_LOWER, and C_LOWER, respectively. The most significant bit of the lower-order 32 bits is handled not as a sign but as a numeric value. [Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.
Function The SBB and SBBO instructions are used in combination to perform subtractions on 64-bit or wider data. These instructions cannot be used independently. <32-bit subtraction with borrow> (1) SBB performs a subtraction on DINT type “IN1” and “IN2” with borrow “BIN.” <32-bit borrow> (1) SBBO performs a subtraction on DINT type “IN1” and “IN2” with borrow “BIN” and outputs a borrow.	

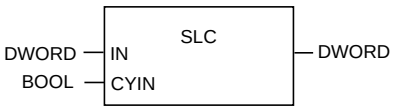
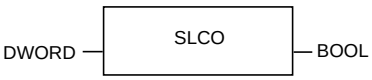
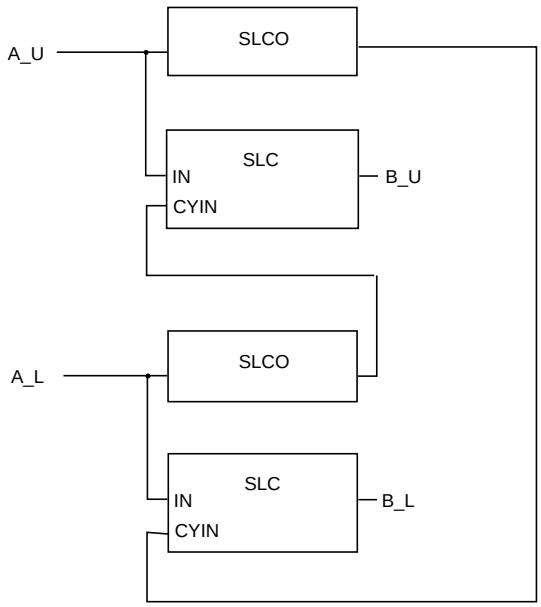
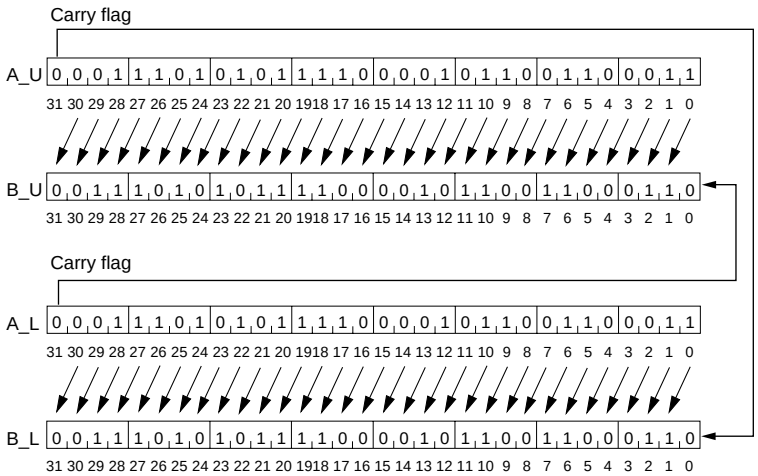
(27) 64-bit multiplication (MULL/MULU)

Name, Symbol, Function	Example
Name MULL/MULU	<Programming example>
Symbol <Lower-order digit in 64-bit multiplication>  <Upper-order digit in 64-bit multiplication>  	 In the above figure, multiplication is performed on 32-bit data items A and B and the result is output in C. The upper-order 32 bits of C are assigned to C_UPPER and the lower-order 32 bits to C_LOWER. The most significant bit of the lower-order 32 bits is handled not as a sign but as a numeric value. [Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.
Function The MULL and MULU instructions are used in combination when performing arithmetic operations that result in 64-bit data. These instructions cannot be used independently. <Lower-order digit in 64-bit multiplication> (1) MULL performs a multiplication of DINT type “IN1” by “IN2” and outputs the lower-order 32 bits of the 64-bit result. <Upper-order digit in 64-bit multiplication> (1) MULU performs a multiplication of DINT type “IN1” by “IN2” and outputs the upper-order 32 bits of the 64-bit result.	

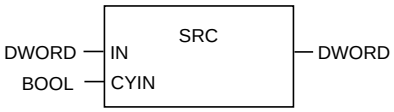
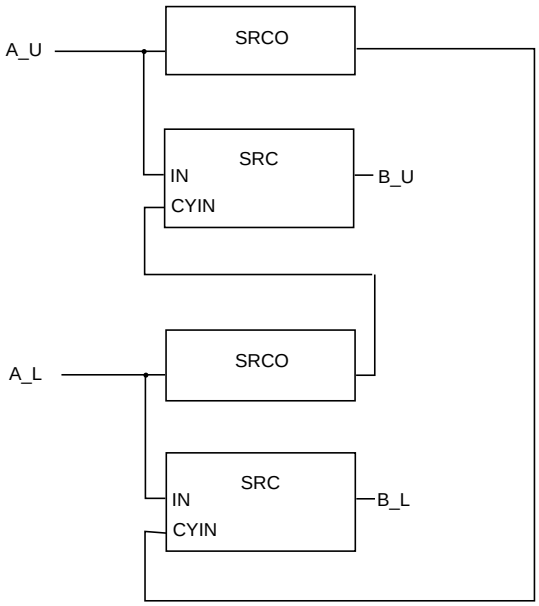
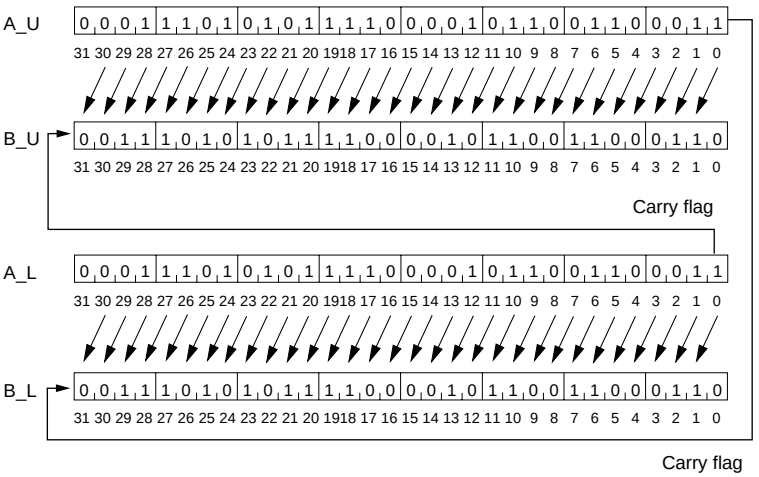
(28) 64-bit division (DIVL/DIVU)

Name, Symbol, Function		Example
Name	DIVL/DIVU	<Programming example>
Symbol	<Lower-order digit in 64-bit division>  <Upper-order digit in 64-bit division>  	 In the above figure, 64-bit data A is divided by B and the result is output in C. The upper-order 32 bits of C are assigned to C_UPPER and the lower-order 32 bits to C_LOWER. The most significant bit of the lower-order 32 bits is handled not as a sign but as a numeric value.
Function	The DIVL and DIVU instructions are used in combination when performing a division on a 64-bit data item. <Lower-order digit in 64-bit division> (1) DIVL divides 64-bit data which is divided into the upper-order part “IN1” of the dividend and the lower-order part “IN2” by the divisor “IN3” and outputs the lower-order 32 bits of the result. <Upper-order digit in 64-bit division> (1) DIVU divides 64-bit data which is divided into the upper-order part “IN1” of the dividend and the lower-order part “IN2” by the divisor “IN3” and outputs the upper-order 32 bits of the result. <Common> (1) If the divisor is 0, the maximum value with the same sign as the dividend is output and ENO is set to 0.	[Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.

(29) Shift left 32 bits with carry SLC/SLCO

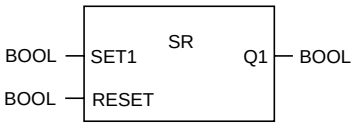
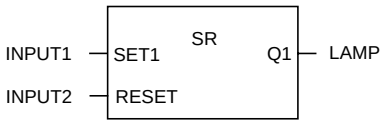
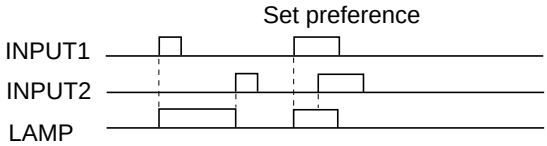
Name, Symbol, Function	Example
Name SLC/SLCO	<Programming example> A sample program for rotating 64-bit data is shown below.
Symbol <32-bit shift left with carry>  <32-bit carry>  	 In the above figure, 64-bit data A is shifted to the left with a carry and the result is output in B. Since a carry is moved, the operation is equivalent to a rotation instruction. <Operation of the above program>  [Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.
Function <32-bit shift left with carry> (1) SLC shifts DWORD type“IN1” 1 bit to the left with a carry. Enter a carry flag data into an “CYIN.” <32-bit carry> (1) SLCO adds DWORD type “IN1” 1 bit to the left and outputs a carry.	

(30) Shift right 32 bits with carry SRC/SRCO

Name, Symbol, Function	Example
Name SRC/SRCO	<Programming example> A sample program for rotating 64-bit data is shown below.
Symbol <32-bit shift right with carry>  <32-bit carry>  	
Function <32-bit shift right with carry> (1) SLC shifts DWORD type “IN1” 1 bit to the right with a carry. Enter a carry flag data into an “CYIN.” <32-bit carry> (1) SLCO adds DWORD type “IN1” 1 bit to the right and outputs a carry.	In the above figure, 64-bit data A is shifted to the right with a carry and the result is output in B. Since a carry is moved, the operation is equivalent to a rotation instruction. <Operation of the above program>  [Reference] It is recommended, to monitor 64-bit data, that you use the hexadecimal notation.

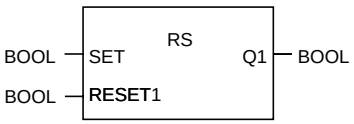
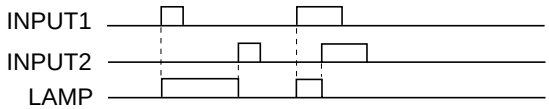
2-5-10 Standard FBs (Function Blocks)

(1) Set reset flip-flop SR

Name, Symbol, Function	Example
Name: SR Symbol:  Function: (1) SR implements a set reset flip-flop. When SET1 and RESET inputs are supplied at the same time, “SET” (Set Input) takes precedence over RESET.	<Programming example>  <Operation> 

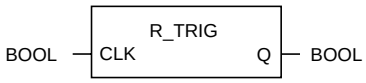
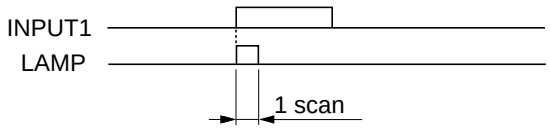
SET1	RESET	Q
1	*	1
0	1	0
0	0	Previous output

(2) Reset set flip-flop RS

Name, Symbol, Function	Example
Name: RS Symbol:  Function: (1) RS implements a reset set flip-flop. When SET1 and RESET inputs are supplied at the same time, “RESET1” (Reset Input) takes precedence over SET1.	<Programming example>  <Operation> 

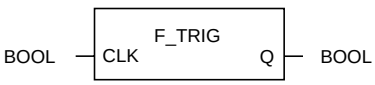
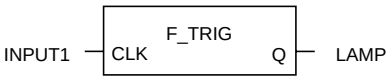
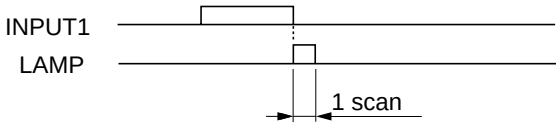
RESET1	SET	Q
1	*	0
0	1	1
0	0	Previous output

(3) Rising edge trigger R_TRIG

Name, Symbol, Function	Example
Name: R_TRIG Symbol:  Function: (1) R_TRIG senses the rising edge of the input and sets and holds the output as (“1”) for one scan period.	<Programming example>  <Operation> 

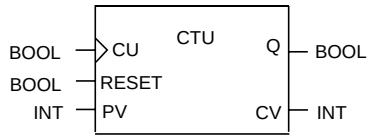
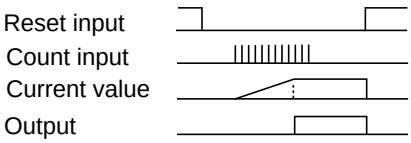
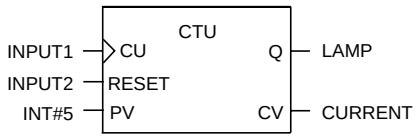
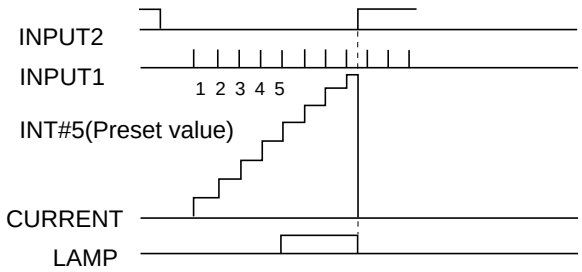
Note: 1) The old value for an input “CLK” has a retain attribute assigned.
This means that at start-up (warm start), the retained old value is used.
At initial start-up (cold start) or download, it is reset to 0 (only for SPH).
Note: 2) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

(4) Falling edge trigger F_TRIG

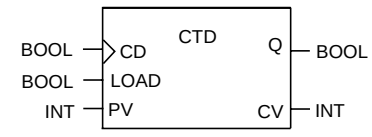
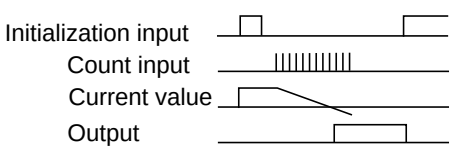
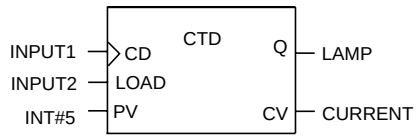
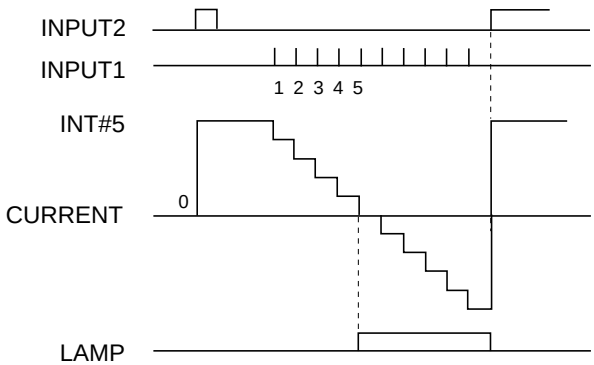
Name, Symbol, Function	Example
Name: F_TRIG Symbol:  Function: (1) F_TRIG senses the falling edge of the input and sets and holds the output at (“1”) for one scan period.	<Programming example>  <Operation> 

Note: 1) The old value for an input “CLK” has a retain attribute assigned.
This means that at start-up (warm start), the retained old value is used.
At initial start-up (cold start) or download, it is reset to 0 (only for SPH).
Note: 2) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

(5) Up Counter CTU

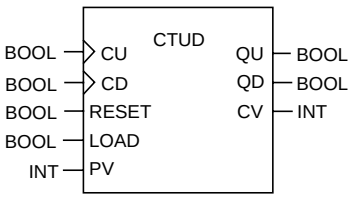
Name, Symbol, Function	Example
Name CTU **Symbol**  **Function** (1) CTU increments the current value “CV” by 1 on the rising edge of the count input “CU.” (2) When the current value “CV” exceeds the preset value “PV,” an “Q” is set to “1.”  (3) The preset value can take on a value from 0 to 32767. (4) Counting continues up to the maximum value (32767). This value will not be exceeded. (5) It is necessary to initially reset the count value by supplying a RESET signal.	<Programming example>  <Operation>  Note: 1) The current value for a count input has a retain attribute assigned. This means that at start-up (warm start), counting continues from the retained old value. At initial start-up (cold start) or download, it is reset to 0 (only for SPH). 2) An “Q” value is the result of a comparison between PV and CV values. When “RESET” = 0, the counted-up “Q” value is reset to 0 if a “PV” value is modified (increased). 3) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

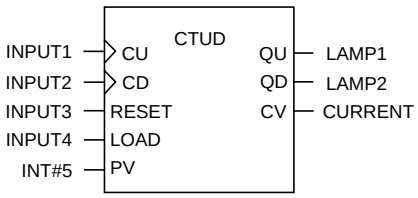
(6) Down counter CTD

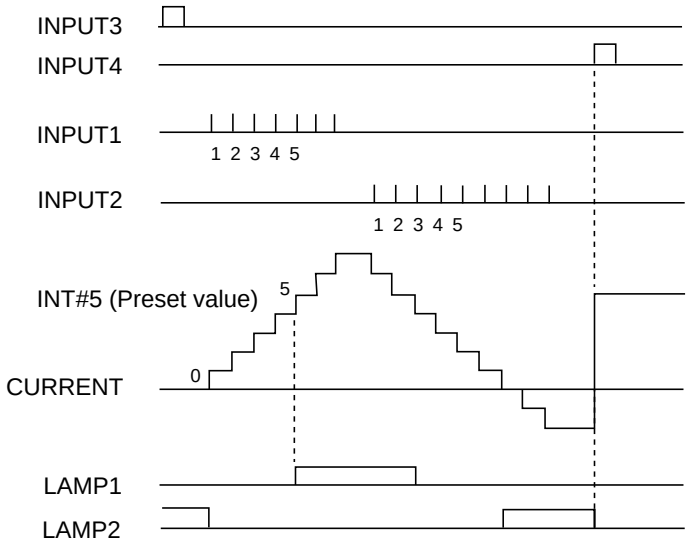
Name, Symbol, Function	Example
Name CTD **Symbol**  **Function** (1) CTD decrements the current value “CV” by 1 on the falling edge of the count input “CD.” (2) The “Q” is turned on when the current value “CV” reaches the preset value “PV.”  (3) The preset value can take on a value from 0 to 32767. (4) Counting continues up to the minimum value (-32768). This value will not be exceeded in the negative direction. (5) It is necessary to initialize the count value initially by supplying a LOAD signal.	<Programming example>  <Operation>  Note: 1) The current value for a count input has a retain attribute assigned. This means that at start-up (warm start), counting continues from the retained old value. At initial start-up (cold start) or download, it is reset to 0 (only for SPH). 2) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

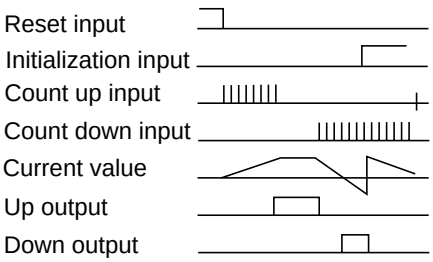
(7) Up down counter CTUD

Name, Symbol, Function	Example
Name CTUD	<Programming example>
Symbol	
Function (1) CTUD increments the current value “CV” by 1 on the rising edge of the count input “CU” and decrements it by 1 on the rising edge of the counter input “CD.” (2) When the current value “CV” exceeds the preset value “PV,” a count-up output “QU” is set to “1.” When the “CV” value is 0, the count-down output “QD” is set to “1.” (3) The preset value can take on a value from 0 to 32767. (4) Count-up continues up to the maximum value (32767) and countdown continues down to the minimum value (-32768). This value range will not be exceeded. (5) It is necessary to reset the current value to 0 by supplying a RESET input, and the “PV” value by supplying an initialization input “LOAD.”	<Operation>









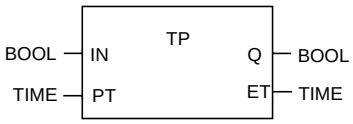
Note: 1) The current value for a count input has a retain attribute assigned. This means that at start-up (warm start), counting continues from the retained old value. At initial start-up (cold start) or download, it is reset to 0 (only for SPH).
2) The priority of processing is shown below.

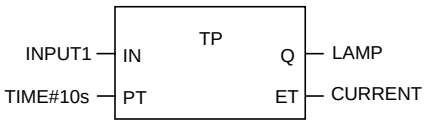
RESET > LOAD > CU↑ > CD↑

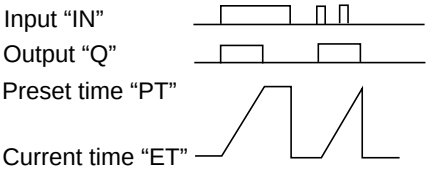
3) An “Q” value is the result of a comparison between “PV” and “CV” values. When “RESET” = 0, the counted-up “Q” value is reset to OFF if a “PV” value is modified (increased).
4) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

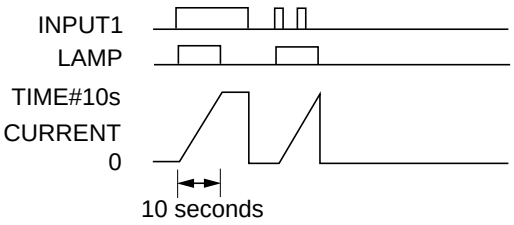
(8) Pulse TP

Name, Symbol, Function	Example
Name TP	<Programming example>
Symbol	
Function (1) TP starts its timer when "IN" is turned on and holds the "Q" on until the preset time "PT" is reached. The current time is output in "ET." (2) The time base is 1 ms. (3) The preset time is as below. SPH: 0 to 4294967295ms SPS: 0 to 2147483647ms (4) The "Q" will not turn on if the preset time is set to 0.	<Operation> Note: 1) Precision of timer instructions When a timer instruction is executed, a +0 to +2 scan time error occurs to update the elapsed time. 2) The timer compares between "PT" and "ET" values and outputs the result into "Q" whenever "IN" is set to "1." When the "PT" value is modified (increased) after time-up, "Q" is set to "1." The timer continues its operation from the current value.



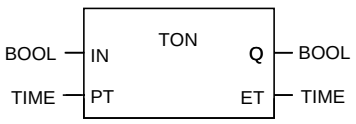


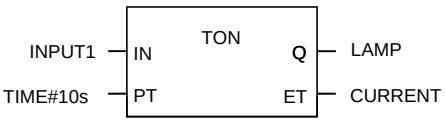


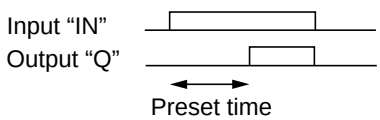


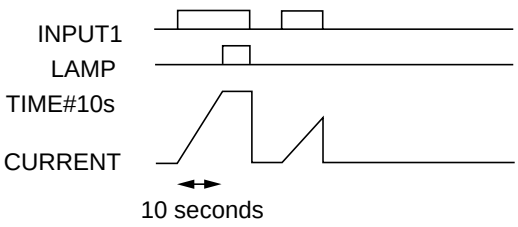
(9) On-delay timer TON

Name, Symbol, Function	Example
Name TON	<Programming example>
Symbol	
Function (1) TON starts its on-delay timer when the "IN" turns on and turns on the "Q" when the preset time "PT" is reached. The current time is output in "ET." (2) When the "IN" turns off, the current timer value is set to 0 and the "Q" to OFF. (3) The time base is 1ms. (4) The preset time is as below. SPH: 0 to 4294967295ms SPS: 0 to 2147483647ms (5) If the preset time is set to 0, the "Q" turns on immediately when the "IN" turns on.	<Operation> Note: 1) Precision of timer instructions When a timer instruction is executed, a +0 to +2 scan time error occurs to update the elapsed time. 2) The timer compares between "PT" and "ET" values and outputs the result into "Q" whenever "IN" is set to "1." When the "PT" value is modified (increased) after time-up, "Q" is set to "1." The timer continues its operation from the current value.

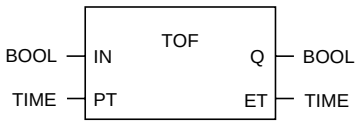
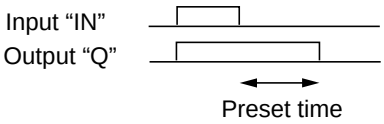
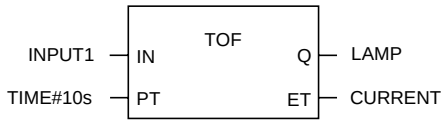
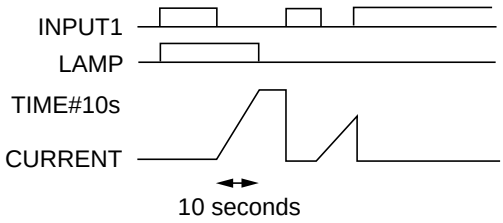




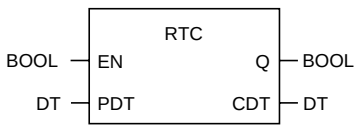
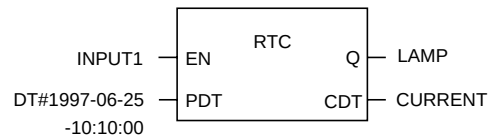
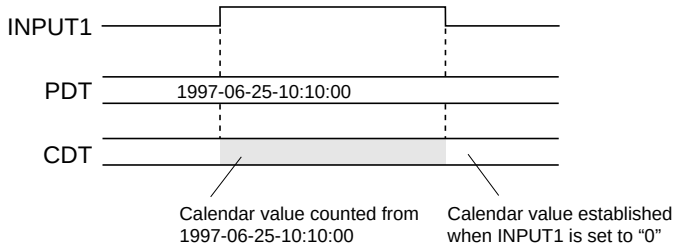




(10) Off-delay timer TOF

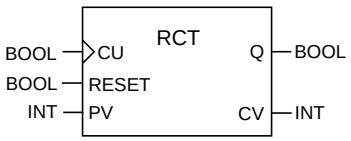
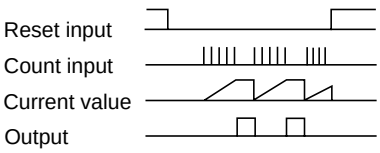
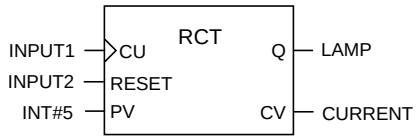
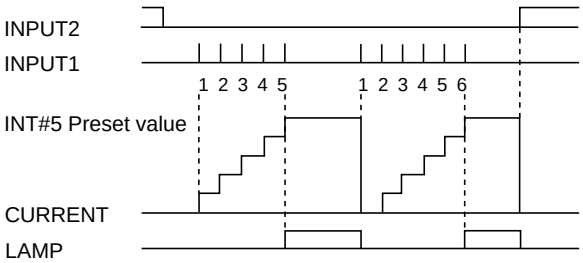
Name, Symbol, Function	Example
Name: TOF Symbol:  Function: (1) TOF turns on its “Q” to reset the current value to 0 when the “IN” turns on. When the input signal turns off, TOF starts counting and turns off its output when the preset time is reached.  (2) The time base is 1ms. (3) The preset time is as below. SPH: 0 to 4294967295ms SPS: 0 to 2147483647ms (4) If the preset time is set to 0, the “Q” turns off immediately when the “IN” turns off.	<Programming example>  <Operation>  Note: 1) Precision of timer instructions When a timer instruction is executed, a +0 to +2 scan time error occurs to update the elapsed time. 2) Under a condition of “IN” = 0, when “PT” exceeds “ET,” “Q” is set to “1,” causing the timer to operate. When “ET” = “PT,” “Q” is set to “0.”

(11) Real-time clock RTC (These functions are not supported in SPS.)

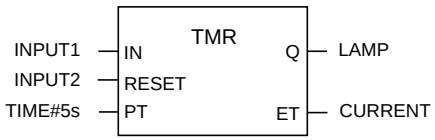
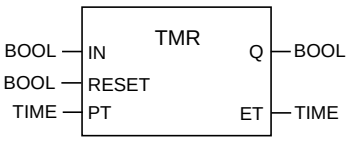
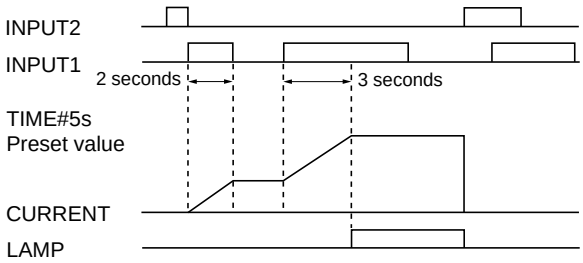
Name, Symbol, Function	Example
Name: RTC Symbol:  Function: (1) Fetches the preset value “PDT” at the rising edge on an “EN.” The current date and time relative to the current value “PDT” is output into the current value “CDT.” The same value is output to the “Q” as that of an input signal. (2) This FB does not set the real-time clock of the system.	<Programming example>  <Operation>  Note: The current value from this FB runs in conjunction with the system's real-time clock. When the system's real-time clock is advanced by 1 minute, for example, the current value of this FB also advances one minute.

2-5-11 Original FBs (Function Blocks)

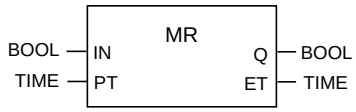
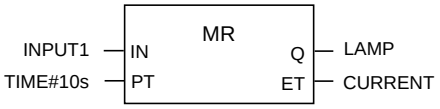
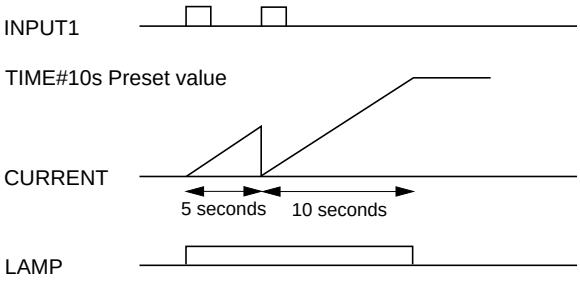
(1) Ring Counter RCT

Name, Symbol, Function	Example
Name RCT	<Programming example>
Symbol	
Function (1) RCT increments the current value “CV” by 1 on the rising edge of the count input “CU.” (2) The output signal “Q” is turned on when the current value “CV” reaches the preset value “PV.” The current value and output signal are turned off on the rising edge of the next count input.  (3) The preset value can take on a value from 0 to 32767. (4) It is necessary to reset the count value initially by supplying a RESET signal.	 <Operation>  Note: 1) The current value for a count input has a retain attribute assigned. This means that at start-up (warm start), counting continues from the retained old value. At initial start-up (cold start) or download, it is reset to 0 (only for SPH). 2) When a "PV" value is modified during operation, the output “Q” outputs the result of comparison between ”PV” and “CV” values. If the modified “PV” value lowers the “CV” value, “Q” is set to “1” and the “CV” value is decreased to the “PV” value. Next time a “CU” value is entered, “CV” is set to 0. 3) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

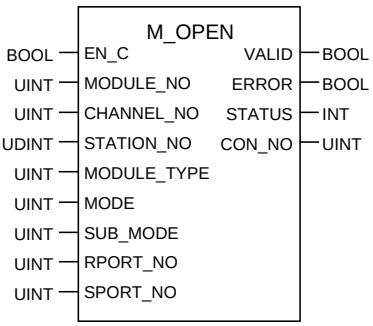
(2) Integrating timer TMR

Name, Symbol, Function	Example
Name TMR	<Programming example> 
Symbol 	<Operation> 
Function (1) TMR starts counting if an OFF-to-ON transition of input “IN” occurs when “RESET” is off, and suspends counting when the input signal turns off. TMR resumes counting when the input is turned on again. (2) The output “Q” is turned on when the current value “ET” reaches the preset value “PT.” (3) The time base is 1ms. (4) The preset time is as below. SPH: 0 to 4294967295ms SPS: 0 to 2147483647ms	Note: 1) The current value for a count input has a retain attribute assigned. This means that at start-up (warm start), counting continues from the retained old value. At initial start-up (cold start) or download, it is reset to 0 (only for SPH). 2) Precision of timer instructions When a timer instruction is executed, a +0 - +2 scan time error occurs to update the elapsed time. 3) Under a condition of “IN” = 1, when “PT” is modified, an output “Q” compares between new “PT” and “ET” values and outputs the result regardless of its current output value. Setting “PT” smaller than “ET” causes “Q” to be set to “1” and the “ET” value is the same as for “PT.” 4) When SPS is used, FB instance memories are all volatile. Therefore, when started up (warm start) just after the system is turned on, the old value is kept cleared to zero.

(3) Retriggerable timer MR

Name, Symbol, Function		Example
Name	MR	<Programming example>
Symbol		 <Operation> 
Function	(1) The timer starts operation at a rising edge at an “IN” and an “Q” remains set to “1” until the timer reaches the preset time. (2) Counting restarts when the input signal is turned on again (rising edge) during counting. (3) The current value “PT,” when reaching the preset “PT” value, becomes undefined. (0 or preset value retained) (4) The time base is 1ms. (5) The preset time is as below. SPH: 0 to 4294967295ms SPS: 0 to 2147483647ms	Note: When a preset value “PT” is modified, the result of comparison between the modified “PT” and “ET” values is output into an output “Q.” When the “PT” value is larger than the “ET” value, “Q” is set to 0. With “IN” set to 1, “ET” is the same value as that for “PT,” and with “IN” set to 0, it is 0.

(4) Open channel M_OPEN

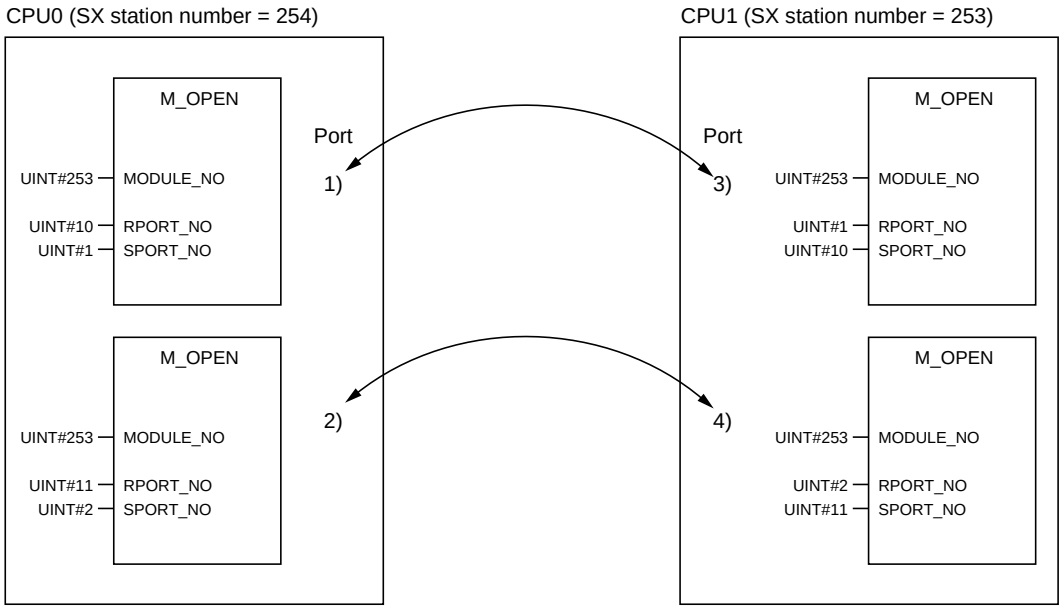
Name, Symbol, Function	Example
Name M_OPEN	Normally, M_OPEN is used in combination with M_SEND and M_RECEIVE. <Terminal description> Input EN_C: Requests open processing (a 0 to 1 transition requests open processing and a 1 to 0 transition requests close processing). MODULE_NO: SX bus station number of the communication module through which messages are to be transferred for out-of-configuration communication, or the SX bus station number of the destination CPU with which messages are to be transferred. CHANNEL_NO: Channel number within the communication module (0 if the module has only one channel). STATION_NO: Station number of the destination module to communicate with on the network (e.g., an IP address on an Ethernet or a P-link station number on a P-link). Has no meaning for in-configuration communication. MODULE_TYPE: Identifies the type of the destination communication module. 0: Message communication with a module within the configuration 1: Message communication with a module outside the configuration MODE: Communications mode (set in the communication module). For settings, refer to the manual for the communication module. SUB_MODE: Communications sub-mode (set in the communication module). For settings, refer to the manual for the communication module. RPORT_NO: Port number of the destination module to communicate (*2). SPORT_NO: The number of the receiving port to be defined on the SX bus for in-configuration communication (*1) and also the number of the receiving port to be defined on the network for out-of-configuration communication (*2). Output VALID: Open processing enabled (a 1 indicates the enabled state.) ERROR: Error reception output (Set and held at 1 during one scan period when an error occurs.) STATUS: Error status CON_NO: Connection number (serial number for a completed open process). *1: The user can specify a port number from 1 to 127 on the SX bus with this FB. *2: An offset is added to the specified destination port number and network port number if the type of the communication module through which messages are to be transferred in the out-of-configuration communications mode is a PC card interface module. For details, refer to the manual for the PC card interface. <Status (STATUS) summary> The status codes related to the M_OPEN FB are described below. Refer to page 2-187 for the common status codes for FBs related with message transmission. (1) Parameter error (code: 177) • When any input is out of the given range. • When “MODULE_NO” exceeds the range of SX bus station numbers (1-254). • When the value input to “MODULE_TYPE” is not the predefined type. • When “MODULE_NO” indicates self station number. (2) Channel open error (code: 193) • An invalid value is specified in “STATION_NO” (the invalid value range varies between different communication modules). • An invalid value is specified in the communications mode (the invalid value range varies between different communication modules). • Neither destination station number (IP address) nor PORT_NO exist on the network when the communications mode is set to the active side (sending side) (valid only with the PC interface module). (3) Port error (code: 200) • The value specified in “SPORT_NO” is not within 1 to 127. • A duplicate “SPORT_NO” value is found within the resource. • When the same “SPORT_NO” has been specified in the communication module. (4) Connection number/client port number full (code: 201) • An attempt was made to open more than 56 ports at the same time within a resource. • An attempt was made to open more ports than permitted for a single communication module (varies with the communication module).
Symbol  	
Function M_OPEN is an FB(function block) for setting up the destination module with which messages are transferred. The settings established by this function block are used in M_SEND (Send Message) and M_RECEIVE (Receive Message) function blocks which are described later. No check is made to verify the establishment of connection with the destination module. This function block can establish up to 56 connections. (1) Setting to on the “EN_C” starts open processing. (A single open process cannot end during one scan.) (2) When open processing is completed, the output “VALID” is turned on and a connection number is output in the output “CON_NO.” In this state, the M_SEND and M_RECEIVE function blocks are available. (3) If open processing terminates abnormally, the output “ERROR” stays on during one scan period and an error code is output in the output “STATUS.” (4) Channel close processing is carried out when the “EN_C” is set to off (close processing does not end within one scan period). (5) When close processing ends, the output “VALID” is turned off (close processing will never terminate abnormally). Note: 1) A single M_OPEN function block enables both message send and receive processing. Although messages can be transferred both ways, i.e., sent and received if they are to be transferred via a T-link, P-link, or PE-link communication module, the module cannot send a message while receiving another message or vice versa. 2) When sending messages via a PC card interface, make sure that the destination module is already opened when executing M_OPEN on the active side. Consequently, it is necessary to complete M_OPEN processing for message receive before executing M_OPEN processing for message send. 3) When the M_OPEN process terminates with an error, it is retried in scanning next time when "EN_C" is set to “1.”	

<Communication ports (PRORT_NO, SPORT_NO)>
The communication port identifies the receiving port of the destination module specified in STATION_NO or MODULR_NO, to which a message is sent.

MICREX-SX can be used to assign any of numbers 1-127 to self-port number.

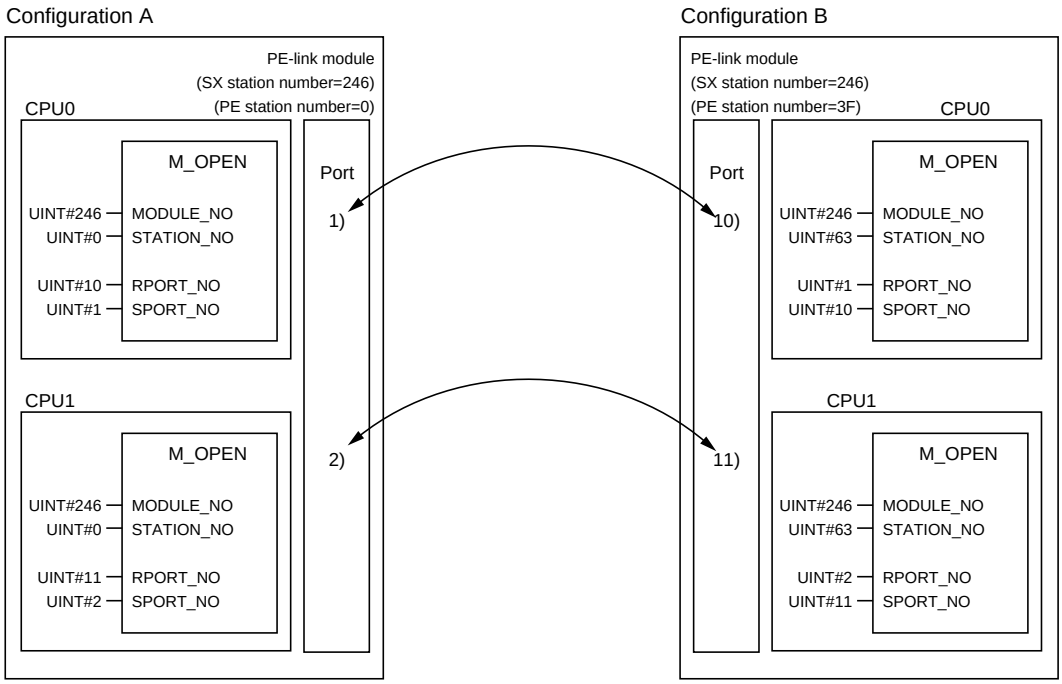
1) Communication ports used in communication within one configuration
In communication within one configuration, the value for SPORT_NO, an N_OPN input is the self-port number. It is the port receiving the message. The value of an input

RPORT_NO is the destination port number in the mate system.

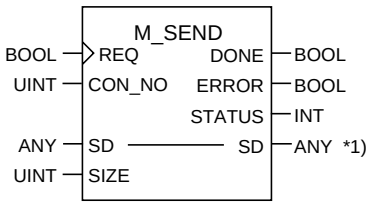


2) Communication port in communication out of the configuration
In communication out of the configuration, the value for SPORT_NO, an M_OPEN input, is the self-port number. It is the port receiving the message in the communication module network. The value for an input RPORT_NO is the destination port number in the mate system (communication module).
Note that the allowable port numbers are restricted,

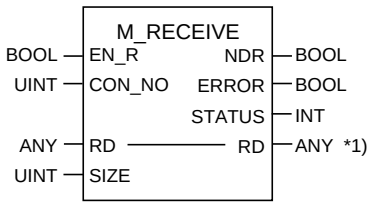
depending on the network type (TCP/IP), and even using the numbers 1-127, ports may not be set. In such a case, the “SPORT_NO” value, to which an offset value is added, is used for the port in the network. Similarly, an offset value is added to the “RPORT_NO” value. The offset value may be defined in the system definition parameter setting for the communication module.



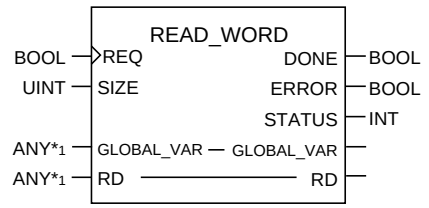
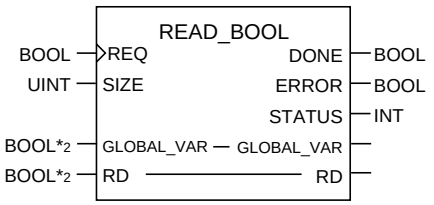
(5) Send message M_SEND

Name, Symbol, Function	Example
Name M_SEND	
Symbol  *1) When SPS is used, be sure to connect same variable to both IN and OUT terminals.	<Terminal description> Input REQ: Send request flag CON_NO: Number of an open connection SIZE: The size in words of the variable for storing the send data (Note) Output DONE: Normal completion flag (set to 1 on completion of the send and held for one scan period.) ERROR: Error flag (set to 1 for one scan period when an error occurs) STATUS: Status Input/output SD: Variable for storing the send data.
Function M_SEND sends messages to a destination module specified in M_OPEN. (1) A message is sent using the settings identified by the connection number specified in the "CON_NO" on the rising edge (0 to 1) of the "REQ." The send processing does not end within one scan period. (2) If the message send is completed, the "DONE" is set and on for one scan period. (3) If the message send is unsuccessful, the "ERROR" is set on for one scan period and an error code is output in the "STATUS." Note: 1) The volume of data that can be sent in a single message send is 2,048 words in the in-configuration communications mode. In the out-of-configuration communications mode, the volume of data varies according to the communication module through which data is to be sent. 2) Available data types of the "SD" are all data types except BOOL. 3) The"REQ" is active on rising edge. However, any low to high transition of "REQ" occurring while a message is being sent (from the time the "REQ" goes high until the time the "DONE" or "ERROR" goes high) is invalid. 4) The state of the input/output signal "SD" must not be changed while M_SEND is processing a message. The integrity of the send data is not guaranteed if the state of this signal is changed. 5) If the number of words specified by the "SIZE" is greater than the size of the variable specified by "SD," the value of the extra words is likely to be unpredictable. Make sure that the value of the "SIZE" matches the size of the variable.	Note: SIZE must be made equal to the size of the SD variable. Garbage data will be appended to the end of the send data if the value of SIZE is greater than that of SD. <Status (STATUS) summary> The status codes related to the M_SEND FB are described below. Refer to page 2-187 for the common status codes for FBs related with message transmission. (1) Send message error (code: 195) • Unable to send a message to the destination communication module. • No response is received from the destination communication module (transmission is completed but no ACK is returned). (2) Channel closed (code: 199) • A closed destination module is found during an out-of-configuration communication. (3) Port error (code: 200) • The destination module is not opened. (4) Buffer overflow (code: 206) • The size of the send data exceeds 4,096 bytes. • The maximum send data size of the communication module is exceeded. (5) Connection number error (code: 207) • An unopened connection number is used. • An attempt was made to send with a connection number that was already in use (this error can occur when two M_SEND FBs are used in parallel). (6) Parameter error (code: 177) • A 0 is supplied to the SIZE terminal. • When a variable for storing sent data is out of the memory area. 6) Program so that 1 is input to an "REQ" when M_OPEN "VARID" is set to "1." 7) M_SEND terminates if the mate system is out of the onfiguration regardless of its M_RECEIVE behavior. If the mate system is within the same configuration, it terminates when M_RECEIVE has been executed normally in the mate system.

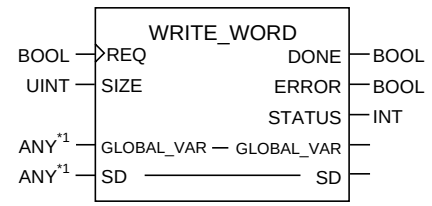
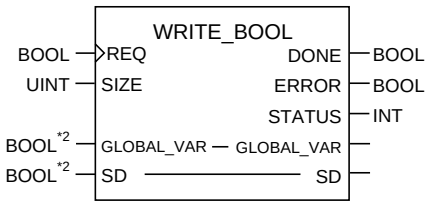
(6) Receive message M_RECEIVE

Name, Symbol, Function	Example
Name M_RECEIVE	<Terminal description> Input EN_R: Receive enable CON_NO: Number of an open connection SIZE: The size of the variable for storing the receive data (in words) Output NDR: Normal completion flag ERROR: Error flag STATUS: Status Input/output RD: Variable for storing the receive data.
Symbol  *1) When SPS is used, be sure to connect same variable to both IN and OUT terminals.	
Function M_RECEIVE receives messages from a source module specified in M_OPEN. (1) A message is received using the settings identified by the connection number specified in the “CON_NO” when the “EN_R” is set to 1. The receive processing does not end within one scan period. (2) If the message receive is completed, the end of receive “NDR” is set and held for one scan period. (3) If the message receive is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The volume of data that can be received in a single message receive is 2,048 words in the in-configuration communications mode. In the out-of-configuration communications mode, the volume of data varies according to the communication module through which data is to be received. 2) Available data types of the “RD” are all data types except BOOL. 3) The “EN_R” must be held at 1 during message receive processing (from the time the “EN_R” goes high until the time the “NDR” or “ERROR” goes high). Setting the “EN_R” to 0 means the suspension of receive processing. 4) Receive processing resumes if the “EN_R” is set high after the suspension of receive processing. Receive processing resumes using the old “CON_NO,” “RD,” and “SIZE” values even if they are altered during the suspension. These changes are not reflected in the subsequent message receive processing. 5) If the “EN_R” remains as 1 in the next scan following the completion of a message receive operation, a new message receive operation will start.	<Status (STATUS) summary> The status codes related to the M_RECEIVE FB are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) Channel closed (code: 199) • A closed source module is found during an out-of-configuration communication. (2) Port error (code: 200) • The source module is not opened. (3) Parameter error (code: 177) • A 0 is supplied to the SIZE terminal. • When a variable for storing sent data is out of the memory area. (4) Buffer overflow (code: 206) • The size of the receive data exceeds the size of the variable for storing the receive data. • In this case, “RD” contains the effective receive data. (5) Connection number error (code: 207) • An unopened connection number is used. • An attempt was made to receive with a connection number that was already being used (this error can occur when two M_RECEIVE FBs are used in parallel). 6) The state of the input/output signal “RD” must be maintained while M_RECEIVE is receiving a message. The integrity of the receive data is not guaranteed if the state of this signal is changed. 7) If the number of words specified by the “SIZE” is greater thanthe size of the variable specified by “RD,” the area for other variables may be overridden. Make sure that the value of the input“SIZE” matches the size of the variable. 8) Program so that 1 is input to an “EN_R” when M_OPEN “VARID” is set to “1.”

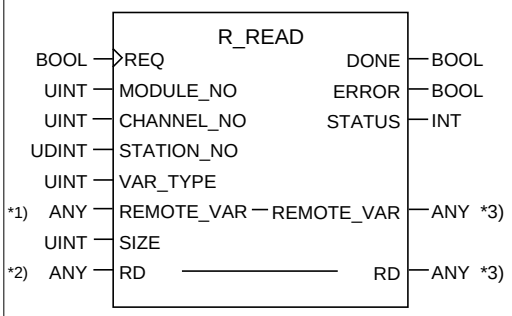
(7) Direct read READ_WORD/READ_BOOL (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name READ_WORD/READ_BOOL	<Terminal description> Input REQ: Read request SIZE: The size (in words or bits) of GLOBAL_VAR and RD variables Input/output GLOBAL_VAR: Global variable assigned to the other resource to be read (Note 7) RD: Variable for storing the read data. Output DONE: Normal completion flag ERROR: Error flag STATUS: Status
Symbol  *1: Any data type except BOOL type Array of any data type except BOOL type  *2: BOOL type or array of BOOL type	<Status (STATUS) summary> The status codes related to the READ_WORD and READ_BOOL FBs are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) Global variable error (code: 176) • The variable declared as GLOBAL_VAR is a variable that is assigned to the local CPU. (2) Parameter error (code: 177) • A 0 is supplied to the SIZE terminal. • When an area for storing READ data is out of the memory area. (3) No free port (code: 201) • When it is attempted to open a port exceeding the limit on the number of port for one communication module. (Direct read/write module also uses a communication port of the module.)
Function READ_WORD and READ_BOOL are used to read variables from different resources (CPU, etc). within the same configuration. Normally, “LD” and “MOVE” are used to read variables from resources on the same processor bus (these FBs may also be used). (1) The variable designated by the “GLOBAL_VAR” is read into the variable area designated by the “RD” on the low to high (0 to 1) transition of the “REQ.” (The read does not end within one scan period). (2) If the read is completed, the “DONE” is set and held for one scan period. (3) If the read is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The “GLOBAL_VAR” and “RD” must be of the same data type. 2) The “SIZE” to be used in READ_WORD must be specified in words and that to be used in “READ_BOOL” in bits. 3) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while data is being read (from the time the “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid.	Note: 4) The state of the input/output signal “RD” must not be changed while read processing is in progress. The integrity of the read data is not guaranteed if the state of this signal is changed. 5) The uniqueness of the read data is guaranteed on a basic data type basis except STRING. The uniqueness of STRING type variables is not guaranteed. Access to an array or structure variable is guaranteed only on an element or member basis. 6) If the number of data items specified by the “SIZE” is greater than the size of the variable specified by “RD,” the area for other “SIZE” matches the size of the variable. 7) The global variable associated with “GLOBAL_VAR” for the resource in the mate system must be specified for both self- and mate-party resources to have the AT statement declared as a global variable.

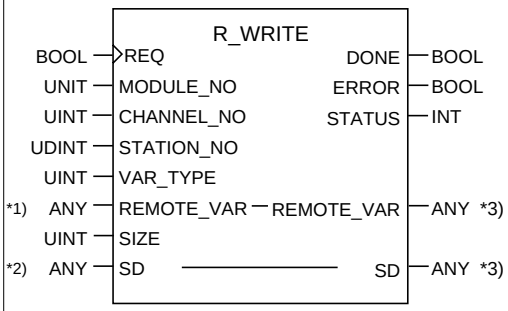
(8) Direct write WRITE_WORD/WRITE_BOOL (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name WRITE_WORD/WRITE_BOOL **Symbol**  *1: Any data type except BOOL type Array of any data type except BOOL type  *2: BOOL type or array of BOOL type **Function** WRITE_WORD and WRITE_BOOL are used to write variables of different resources (CPU, etc.) within the same configuration. These instructions are not required when writing variables into resources on the same processor bus (these FBs may also be used). The WRITE_WORD instruction can be used to write data into the waiting CPU in the warm standby mode in the duplex system. This function equalizes variable data. (1) The variable area designated by the “SD” is written into the global variable area designated by the “GLOBAL_VAR” on the low to high (0 to 1) transition of the “REQ.” (The write does not end within one scan period.) (2) If the write is completed, the “DONE” is set and held for one scan period. (3) If the write is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The “GLOBAL_VAR” and “SD” must be of the same data type. 2) The “SIZE” to be used in WRITE_WORD must be specified in words and that to be used in “WRITE_BOOL” in bits.	<Terminal description> Input REQ: Write request SIZE: The size (in words or bits) of GLOBAL_VAR and SD variables Input/output GLOBAL_VAR: Global variable assigned to the other resource to be written Note 7) When data is written into the waiting CPU in the 1 to 1 duplex system, the variable assigned to self-resource SD: Variable for storing the data to be written Output DONE: Normal completion flag ERROR: Error flag STATUS: Status <Status (STATUS) summary> The status codes related to the WRITE_WORD and WRITE_BOOL FBs are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) Global variable error (code: 176) • When the warm standby mode is not used in the 1 to 1 duplex system and the variable specified in GLOBAL_VAR has been assigned to the self-CPU. • When the warm standby mode is used in the 1 to 1 duplex system and the variable specified for GLOBAL_VAR is assigned to the high-speed memory area of the self-CPU. (2) Parameter error (code: 177) • A 0 is supplied to the SIZE terminal. (3) Transmission interlock error (code: 35) • When the mate-party module is interlocked. Transmission interlock occurs when an instance screen opens by accompanying any operation such as download. In this case, retry it. (4) No free port (code: 201) • When it is attempted to open a port exceeding the limit on the number of port for one communication module. (Direct read/write module also uses a communication port of the module.) 3) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while data is being written (from the time the “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid. 4) The state of the input/output signal “SD” must not be changed while write processing is in progress. The integrity of the write data is not guaranteed if the state of this signal is changed. 5) The uniqueness of the written data is guaranteed on a basic data type basis except STRING. The uniqueness of STRING type variables is not guaranteed. Access to an array or structure variable is guaranteed only on an element or member basis. To equalize data into the waiting CPU, consistency may be assured until a “SIZE” value reaches 240. 6) If the number of data items specified by the “SIZE” is greater than the size of the variable specified by “GLOBAL_VAR,” the area for other variables in the target module may be overridden. Make sure that the value of the “SIZE” matches the size of the variable. 7) The global variable associated with “GLOBAL_VAR” for the resource in the mate system must be specified for both self- and mate-party resources to have the AT statement declared as a global variable. 8) The WRITE_BOOL instruction cannot be used for memory in the P/PE-link module.

(9) Remote data read R_READ

Name, Symbol, Function	Example
Name R_READ	<Terminal description> Input REQ: Read request MODULE_NO: SX bus station number of the communication module CHANNEL_NO: Channel number of the communication module STATION_NO: Network station number of the source station VAR_TYPE: Variable specification method (Refer to page 2-186.) SIZE: The size (in words) of variables storing the data to be read and the read data Output DONE: Normal completion flag ERROR: Error flag STATUS: Status Input/output REMOTE_VAR: Read address (Refer to page 2-186.) RD: Variable for storing the read data.
Symbol  *1: Array of any data type except BOOL type *2: Any data type except BOOL type Array of any data type except BOOL type *3: When SPS is used, be sure to connect same variable to both IN and OUT terminals.	<Status (STATUS) summary> The status codes related to the R_READ FB are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) Channel open error (code: 193) • When an invalid value has been set for a channel number (2) Message send error (code: 195) • When an invalid value has been set for a station number • When an invalid value has been set for a channel number • When any other value than type codes has been set for a memory type (3) No free port (code: 201) • When you make an attempt to open excessive ports in one communication module (4) Transfer size over (code: 206) • When any other value than “0” is specified for an input “VAR_TYPE” and a message data exceeds the size limit for the communication module through which it will pass (5) Parameter error (code: 177) • When “0” is input into “SIZE” • When any other value than specified values is input into “VAR_TYPE” • When a value out of a range of allowed values for an SX bus station number is input into “MODULE_NO” (For example, MODULE_NO is self SX bus station number.) • When a variable for storing READ data is out of the memory area. (6) Internal resource shortage (code: 171) • When a shortage occurs in internal resources for executing R_READ and R_WRITE If more than one R_READ and R_WRITE are started simultaneously, a shortage may occur in internal resources. In this case, wait for a moment and retry it. (7) Memory address specification error (code: 68) • When an invalid address has been specified in “REMOTE_VAR.” (Only when 0 is specified in “REMOTE_VAR” does it occur.) (8) Memory size over (code: 69) • When the address specified in “REMOTE_VAR” + the input “SIZE” value is out of the range of allowed addresses. In this case, an I/O value “RD” is not assured. (9) Mate specification error (code: 160) • When VAR_TYPE = 0 and the CPU number in the mate system specified in REMOTE_VAR is not found
Function R_READ reads data in the direct addressing mode from a device that is connected to a network via a communication module listed below that can be read. • Memory in a CPU of the SPH system via a network (of any type). • Memory in a CPU of the MICREX_F or FLEX-PC via a T-link, P-link, or PE-link. • Device that is connected to an open standard network such as JPCN-1. (1) The data designated by the “MODULE_NO,” “CHANNEL_NO,” “STATION_NO,” and “REMOTE_VAR” is read into the variable area designated by the input “RD” on a low to high (0 to 1) transition of the “REQ.” (The read does not end within one scan period.) (2) If the read is completed, the “DONE” is set and held for one scan period. (3) If the read is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The “VAR_TYPE” must specify the method of specifying the variables to be read. 2) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while data is being read (from the time the input “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid. 3) The state of the input/output signal “RD” must not be changed while read processing is in progress. The integrity of the read data is not guaranteed if the state of this signal is changed.	4) There is no restriction on the “SIZE” when the input “VAR_TYPE” is set to 0. In the other cases, it is subject to restrictions depending on the communication module through which data is to be read. 5) If the number of data items specified by the “SIZE” is greater than the size of the variable specified by “RD,” the area for other variables may be overridden. Make sure that the value of the “SIZE” matches the size of the variable. 6) Under a condition of VAR_TYPE ≠ 0, the entire read data is stored into an I/O “RD” and the operation terminates when the size of data read out from the device is smaller than “SIZE.” 7) Be careful so that no compile error will occur even when BOOL type array variable is used for “RD.”

(10) Remote data write R_WRITE

Name, Symbol, Function	Example
Name R_WRITE	<Terminal description> Input REQ: Write request MODULE_NO: Communication module number CHANNEL_NO: Channel number STATION_NO: Destination network station number VAR_TYPE: Variable specification method (Refer to page 2-186.) SIZE: The size of write data Output DONE: Normal completion flag ERROR: Error flag STATUS: Status Input/output REMOTE_VAR: Variable to be accessed (Refer to page 2-186.) SD: Variable for storing the write data.
Symbol  *1: Array of any data type except BOOL type *2: Any data type except BOOL type Array of any data type except BOOL type *3: When SPS is used, be sure to connect same variable to both IN and OUT terminals.	<Status (STATUS) summary> The status codes related to the R_WRITE FB are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission.
Function R_WRITE writes data in the direct addressing mode into a device that is connected to a network via a communication module listed below. • Memory in a CPU of the SPH system via a network (of any type). • Memory in a CPU of the MICREX_F or FLEX-PC via a T-link, P-link, or PE-link. • Device that is connected to an open standard network such as OPCN-1 and FL-net. (1) The data designated by the “SD” is written into the area designated by the “MODULE_NO,” “CHANNEL_NO,” “STATION_NO,” and “REMOTE_VAR” on a low to high (0 to 1) transition of the “REQ.” (The write does not end within one scan period.) (2) If the write is completed, the “DONE” is set and held for one scan period. (3) If the write is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The “VAR_TYPE” must specify the method of specifying the variables to be written. 2) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while data is being write (from the time the “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid. 3) If the size of data specified by the “SIZE” is greater than the size of the variable specified by “SD,” the extra data is likely to be unpredictable. Make sure that the value of the “SIZE” matches the size of the variable.	(1) Channel open error (code: 193) • When an invalid value has been set for a channel number. (2) No free port (code: 201) • An attempt was made to open more ports than permitted in a single communication module. (3) Transfer size overflow (code: 206) • The size limit defined for the communication module to be used is exceeded when the input “VAR_TYPE” is set to a nonzero value. (4) Parameter error (code: 177) • A 0 is supplied to the SIZE terminal. • An invalid value is supplied to the “VAR_TYPE” terminal. • A value greater than the maximum permissible SX bus station number is supplied to the “MODULE_NO” terminal. • When a variable for storing written data is out of the memory area. • When MODULE_NO is the self-SX bus station number. (5) Internal resource shortage (code: 171) • When a shortage occurs in internal resources for executing R_READ and R_WRITE • If more than one R_READ and R_WRITE are started simultaneously, a shortage may occur in internal resources. In this case, wait for a moment and retry it. (6) Memory address specification error (code: 68) • When an invalid address has been specified in “REMOTE_VAR.” (Only when 0 is specified in “REMOTE_VAR” does it occur.) (7) Memory size over (code: 69) • When the address specified in “REMOTE_VAR” + input “SIZE” is out of a range of allowed address for the mate module. In this case, data might have been written into the mate module. (8) Mate specification error (code: 160) • When VAR_TYPE = 0 and the CPU number in the mate system specified in REMOTE_VAR is not found. (9) Transmission interlock error (code: 35) • When the mate-party module is interlocked Transmission interlock occurs when an instance screen opens by accompanying any operation such as download. In this case, retry it. 4) Be careful so that no compile error will occur even when BOOL type array variable is used for “SD.”

<Variable specification method>

The legitimate values of the variable specification method to be specified in the Remote Data Read (R_READ) and Remote Data Write (R_WRITE) instructions ("VAR_TYPE" and "REMOTE_VAR") are defined individually for each target of access in the destination station.

VAR_TYPE	Variable Specification Method	Target of Access
0	SPH address	CPU memory in the SPH system via a network (of any type)
1	MICREX_F or FLEX-PC address	CPU memory in the MICREX or FLEX-PC via a T-link, P-link, or PE-link
2	Sequence of numerals	Device connected to an open standard network such as OPCN1
3	Character string	Device connected to an open standard network such as OPCN1

<VAR_TYPE formats>

- **When VAR_TYPE = 0**
  - REMOTE_VAR

F 0

CPU No.
Memory type *
Address, lower-order
Address, upper-order

- **When VAR_TYPE = 1**
  - REMOTE_VAR

F 0

File No.
Word address within file

- **When VAR_TYPE = 2**
  - REMOTE_VAR

F 0

Effective size n	
-	Address 1
-	-
-	-
-	Address n

In this case, the effective data is placed in the lower-order 8 bits of a 16-bit array. This is because an SPH system cannot handle 8-bit data.

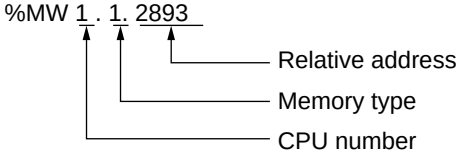
- **When VAR_TYPE = 3**
  - REMOTE_VAR

F 0

ASCII character string The end of a string is represented by a null code.

For the "REMOTE_VAR" format, refer to the manuals for the communication module through which data is to be read or written and the device to be read from or written to.

* The memory type is the code for identifying standard memory, retain memory, etc., in user memory. It is the second column value in addressing (AT statement) any other CPU memory.



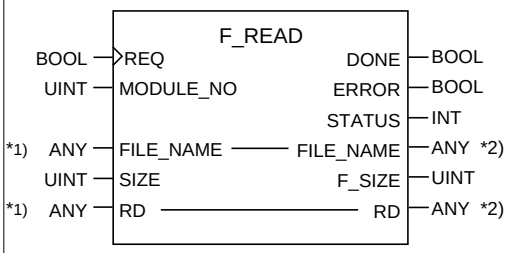
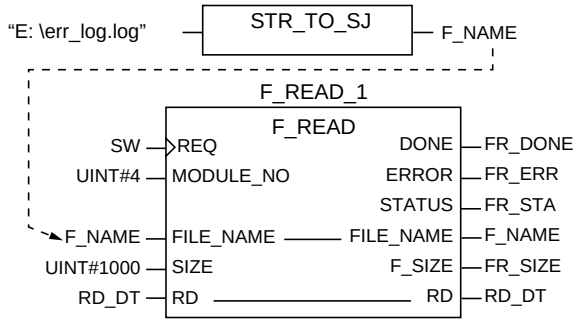
Name	Memory type code
Standard memory	1
Retain memory	3
User FB memory	5
System FB memory	9
System memory	10

Note) Do not specify 1, 3, 5, 9, or 10 for the memory type code.

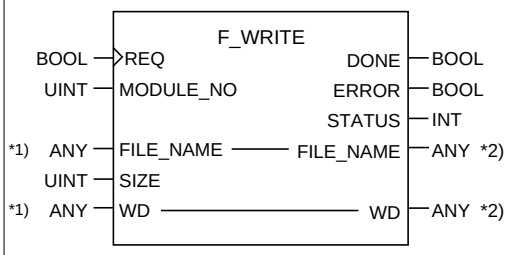
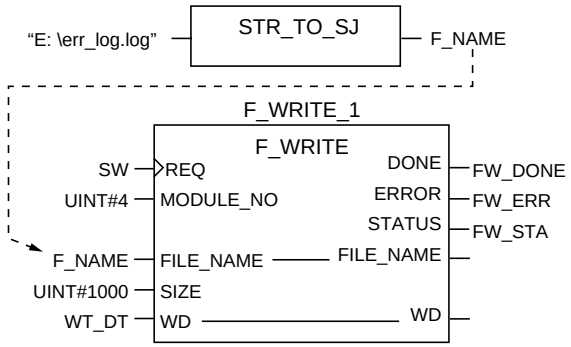
<Common status codes>

Error Code	Name	Cause	Countermeasure
66(42h)	Memory access error	Attempted to access uninstalled P/PE-link memory or FL-net common memory.	• Check configuration of the module.
162(A2h)	No response to command	No answer command was received even when specified time had elapsed.	• Check the condition and wiring of the remote equipment.
164(A4h)	Message send error	The mate system is missing or the specified SX station number has no module.	• Check the input terminal for defining the destination.
165(A5h)	Message receive busy	Can't send a message over an SX bus because the destination module is busy.	• Verify that the mate-party module supports message communication. • Re-execute the FB at a later time. • If this error occurs frequently, it implies that the message transmission load of the destination module is too high. Lower the message load.
170(AAh)	Message send busy	Can't send a message because the CPU resource for sending a message is busy.	• Re-execute the FB at a later time. • If this error occurs frequently, it implies that the message transmission load of the local CPU is too high. Lower the message load.
197(C5h)	Network send busy	Can't perform an inter-communication module transmission over a network because the destination communication module is busy.	• Re-execute the FB at a later time. • If this error occurs frequently, it implies that the message transmission load of the local CPU is too high. Lower the message load.

(11) File data read F_READ

Name, Symbol, Function	Example
Name F_READ	<Terminal description> Input REQ: Read request MODULE_NO: SX bus station number of the module containing a memory card SIZE: The size (in words) of variables connected to RD Output DONE: Normal completion flag ERROR: Error flag STATUS: Status F_SIZE: The size (in words) of the read file Input/output FILE_NAME: File name (Character code is Shift-JIS.) RD: Variable for storing the read data.
Symbol  *1: Array of any data type except BOOL type *2: When SPS is used, be sure to connect same variable to both IN and OUT terminals.	<Status (STATUS) summary> The status codes related to the F_READ FB are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) File name error (code: 65) • The file with the specified file name is not found. (2) File access error (code: 66) • A check sum error occurs while reading a file. (3) Parameter error (code: 177) • A value greater than the maximum permissible SX bus station number is supplied to the “MODULE_NO” terminal. • When a variable for storing read data is out of the memory area • When MODULE_NO is the self-SX bus station number (4) No free port (code: 201) • When it is attempted to open a port exceeding the limit on the number of port for one communication module. (5) Buffer overflow (code: 206) • When received data exceeds the size of the variable associated with RD (6) Transmission interlock error (code: 35) • When the module with a memory card inserted is accessed using the file access instruction, the system enters the transmission interlock state. If another file access memory makes an attempt to enter the module, a transmission interlock error occurs. Retry it. (7) Channel open error (code: 193) (only for SPS) • Internal error
Function F_READ reads a file from a memory module in the configuration or from a PC card interface module containing a memory card. (1) The file designated by the “FILE_NAME” is read from the memory module or PC card interface module designated by the “MODULE_NO” into the variable area designated by the “RD” on a low to high (0 to 1) transition of the “REQ.” (The read does not end within one scan period.) (2) If the read is completed, the “DONE” is set and held for one scan period. (3) If the read is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the "STATUS." Note: 1) The “FILE_NAME” must specify a file name in Shift-JIS code. 2) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while the file is being read (from the time the “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid. 3) The state of the “RD” must not be changed while a file is being read. The integrity of the read data is not guaranteed if the state of this signal is changed. 4) If the number of data items specified by the “SIZE” is greater than the size of the variable specified by “RD,” the area for other variables may be overridden. Make sure that the value of the “SIZE” is equal to or smaller than the size of the variable. 5) Do not use the memory card if the module for storing an application program for the n to 1 duplex mode for file memory for file data read/write access is from a CPU application program. Prepare another memory card if the module is for file read/write.	<Programming example> The sample program shown below reads a file with a file name of “E:\err_log.log” into the 1000-word variable “RD_DT.” It is assumed that the SX bus station number of the module containing the memory card is “4.”  If the memory card of module is shared, a contention occurs in accessing, and switching cannot be performed between the operating and waiting CPUs in the duplex mode. 6) Be careful so that no compile error will occur even when BOOL type array variable is used for “RD.”

(12) File data write F_WRITE

Name, Symbol, Function	Example
Name F_WRITE **Symbol**  *1: Array of any data type except BOOL type *2: When SPS is used, be sure to connect same variable to both IN and OUT terminals. **Function** F_WRITE writes a file to a memory module in the configuration or to a PC card interface module containing a memory card. (1) The data designated by the “WD” is written from the memory module or PC card interface module designated by the “MODULE_NO” into the file designated by the “FILE_NAME” on a low to high (0 to 1) transition of the “REQ.” (The write does not end within one scan period.) (2) If the write is completed, the “DONE” is set and held for one scan period. (3) If the write is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.” Note: 1) The “FILE_NAME” must specify a file name in Shift-JIS code. 2) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring while thefile is being written (from the time the “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid. 3) The state of the “WD” must not be changed while a file is being written. The integrity of the write data is not guaranteed if the state of this signal is changed. 4) An existing file that has the same name is overwritten. 5) If the number of data items specified by the “SIZE” is greater than the size of the variable specified by “WD,” the extra data in the unpredictable state is written into the file. Make sure that the value of the “SIZE” is equal to or smaller than the size of the variable. 6) Do not use the memory card if the module for storing an application program for the n to 1 duplex mode for file memory for file data read/write access is from a CPU application program.	<Terminal description> Input REQ: Write request MODULE_NO: SX bus station number of the module containing a memory card SIZE: The size (in words) of write data Output DONE: Normal completion flag ERROR: Error flag STATUS: Status Input/output FILE_NAME: File name (Character code is Shift-JIS.) WD: Variable for storing the write data. <Status (STATUS) summary> The status codes related to the F_WRITE FB are described below. Refer to page 2-187 for the common status codes for FBs related with the message transmission. (1) File name error (code: 65) • The directory with the specified file name is not found. If the directory exists but the file does not exist, a new file is created. (2) File access error (code: 66) • An error occurred while writing to a memory card. (3) No free area (code: 69) • Unable to write because there is no free space in the memory card. (4) Parameter error (code: 177) • A value greater than the maximum permissible SX bus station number is supplied to the “MODULE_NO” terminal. • When a variable for storing written data is out of the memory area • When MODULE_NO is the self-SX bus station number (5) No free port (code: 201) • When it is attempted to open a port exceeding the limit on the number of port for one communication module. (6) Transmission interlock error (code: 35) • When the module with a memory card inserted is accessed using the file access instruction, the system enters the transmission interlock state. If another file access memory makes an attempt to enter the module, a transmission interlock error occurs. Retry it. (7) Channel open error (code: 193) (only for SPS) • Internal error <Programming example> The sample program shown below writes the contents of “WT_DT” into a file with a file name of “E:\err_log.log.” It is assumed that the SX bus station number of the module containing the memory card is “4.”  Prepare another memory card if the module is for file read/write. If the memory card of module is shared, a contention occurs in accessing, and switching cannot be performed between the operating and waiting CPUs in the duplex mode. 7) Be careful so that no compile error will occur even when BOOL type array variable is used for “WD.”

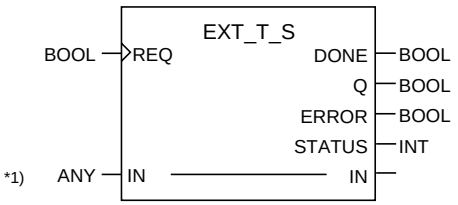
(13) Extension test & set EXT_T_S (These functions are not supported in SPS.)

Name, Symbol, Function

Name

EXT_T_S

Symbol



*1: BOOL type

Function

EXT_T_S acquires a semaphore. It is used when the bit variable to be used to implement a semaphore is assigned to the memory of a module on a different processor bus.

(1) EXT_T_S tests and sets the global bit variable area designated by the “IN” on a low to high (0 to 1) transition of the “REQ.” (This processing does not end within one scan period.)
(2) If the test & set is completed, the “DONE” is set and held for one scan period and the result is output in the “Q.”
(3) If the test & set is unsuccessful, the “ERROR” is set and held for one scan period and an error code is output in the “STATUS.”

Note: 1) The WRITE_BOOL instruction must be used to reset set data.
2) The “REQ” is active on rising edge. However, any low to high transition of “REQ” occurring during test & set processing (from the time the input “REQ” goes high until the time the “DONE” or “ERROR” goes high) is invalid.
3) Use the T_S instruction if the bit variable to be used to implement a semaphore is connected to the same processor bus or the local CPU.
4) Do not use this instruction for memory in the P/PE-link module. The execution terminates abnormally with an invalid FB result.
5) In the duplex system, to pass data to the bit variables used for semaphores at changeover between the operating and waiting CPUs:
• Assign the bit variables for semaphores to the standard memory area to avoid equalization. (They are reset to 0 at changeover.)
• When changeover occurs, make an attempt to get semaphores again.

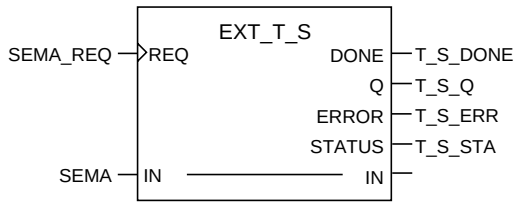
Example

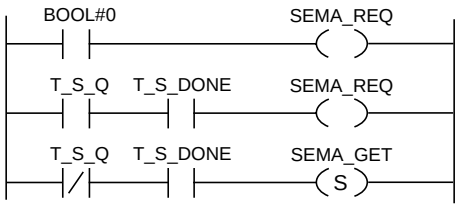
<Terminal description>
Input
REQ: Test & set request
Input/output
IN: Global bit variable assigned to another resource
Output
DONE: Normal completion flag
Q: Test & set result
0: Set complete
1: Set disabled (already set)
ERROR: Error flag
STATUS: Status

<Status (STATUS) summary>
(1) Global bit variable specification error (code: 170)
The variable designated by “IN” is assigned to the local CPU.
(2) Unsupported instruction detection (code: 32)
When this instruction is used for memory in the P/PE-link module.

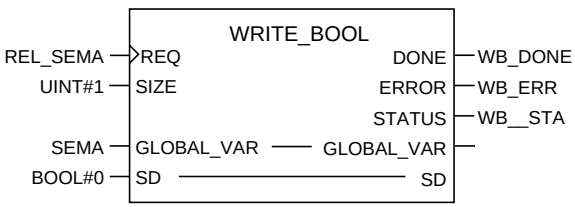
<Programming example>
The sample program shown below acquires a semaphore when the state of “SEMA_REQ” is changed from 0 to 1. “SEMA_GET” is set to 1 when the processing is completed. The instruction retries if the processing is unsuccessful. The EXT_T_S releases the semaphore when the state of “REL_SEMA” is switched from 0 to 1. “SEMA_GET” is set to 0 when the release processing is completed.

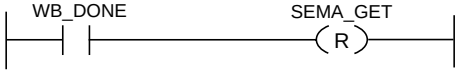
<Getting a semaphore>



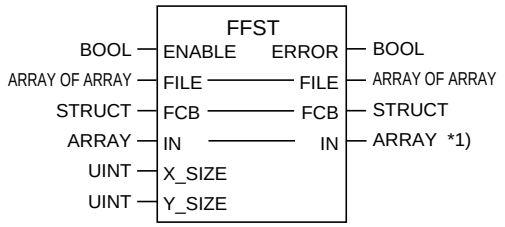
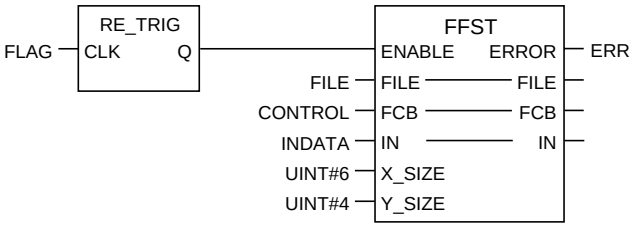
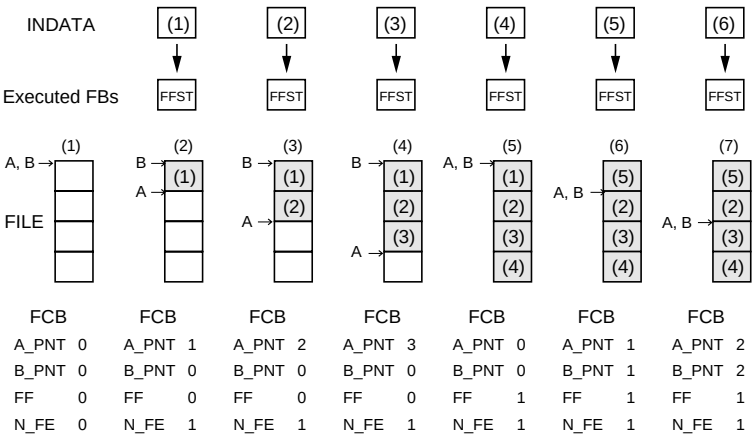


<Releasing a semaphore>

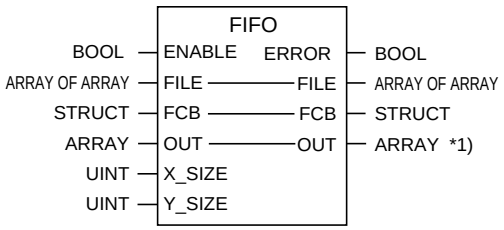
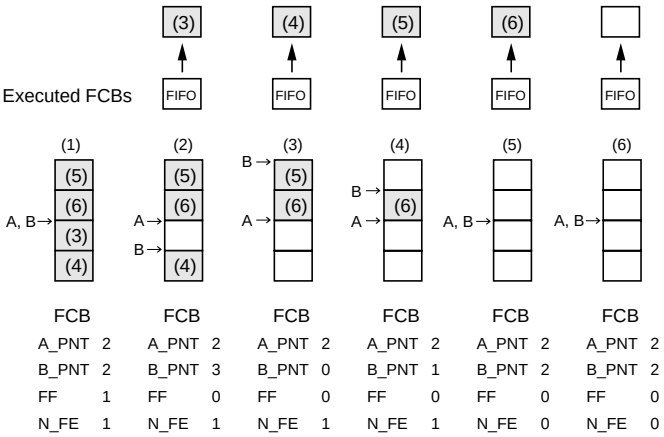




(14) Sequential file store FFST

Name, Symbol, Function	Example
Name FFST **Symbol**  *1: When SPS is used, be sure to connect same variable to both IN and OUT terminals. **Function** (1) FFST writes the data block designated by the “IN” into the sequential file area designated by the “FILE” when the “ENABLE” is set to on. (2) The “FCB” specifies the file control block which is used for controlling write processing. (3) Specify the number of elements of the array connected to “IN” for a “X_SIZE” (one-dimensional size of an array). Specify the two-dimensional size of the array for a “Y_SIZE.” (4) If the file area is already full, the old data is discarded and the data designated by the “IN” is written. (5) If the write is unsuccessful, the “ERROR” is set and held for one scan period. (6) Available data type of “IN” is an array of any data type except BOOL type. Normal operation is not guaranteed if an array of BOOL type is specified on “FILE.” Note: If the result of “X_SIZE” and “Y_SIZE” multiplication is different from (larger than) the size of the array of arrays connected to “FILE,” any other variable area may be overwritten.	<Programming example>  <Sample data type definition> TYPE ELEMENT1 : ARRAY[1..6] OF INT; (*X_SIZE*) FILE1 : ARRAY[1..4] OF ELEMENT1; (*Y_SIZE*) FCB1 : STRUCT A_POINTER : UINT; B_POINTER : UINT; N_FE : BOOL; FF : BOOL; END_STRUCT; END_TYPE File control block definition The file control block should be defined as shown to the left. A_POINTER: FFST Write pointer FILO read pointer B_POINTER: FIFO read pointer N_FE: A “0” indicates that the file is empty (Not File Empty). FF: A “1” indicates that the file is full (File Full). (Examples of variable declarations) VAR FILE : FILE1; INDATA : ELEMENT1; ← Note) Use the same data type for “IN” as that of “FILE” (*X_SIZE*). CONTROL : FCB1; END_VAR <Operation> When a FLAG is set, “INDATA” data is stored in “FILE.” 

(15) Sequential file load first FIFO

Name, Symbol, Function	Example
Name FIFO **Symbol**  *1: When SPS is used, be sure to connect same variable to both IN and OUT terminals. **Function** (1) Setting to on the “ENABLE” causes the oldest data (at the B_POINTER position in the FCB) to be read from the specified sequential file area into the area designated by the “OUT” and the B_POINTER to be incremented by 1. At this moment, if the FF flag (File Full) is on, it is set to off. (2) The N_FE bit is set to off (to notify that the file has become empty) when B_POINTER catches up with A_POINTER. (3) The “FCB” specifies the file control block which is used for controlling read processing. (4) Specify the number of elements of the array connected to “OUT” for a “X_SIZE” (one-dimensional size of an array). Specify the two-dimensional size of the array for a “Y_SIZE.” (5) Processing is flagged as invalid if an attempt is made to read an empty file. (6) If the read is unsuccessful, the “ERROR” is set and held for one scan period. (7) Available data type of “OUT” is an array of any data type except BOOL type. Normal operation is not guaranteed if an array of BOOL type is specified on “FILE.” (8) “OUT” is overwritten when “ENABLE” is set to “ON” and the file is not empty. Note) If the result of “X_SIZE” and “Y_SIZE” multiplication is different from (larger than) the size of the array of arrays connected to “FILE,” any other variable area may be overwritten.	<Operation> 

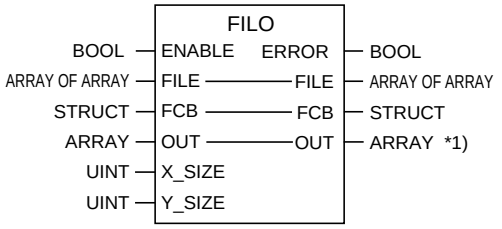
(16) Sequential file load last FILO

Name, Symbol, Function	Example

Name

FILO

Symbol



*1: When SPS is used, be sure to connect same variable to both IN and OUT terminals.

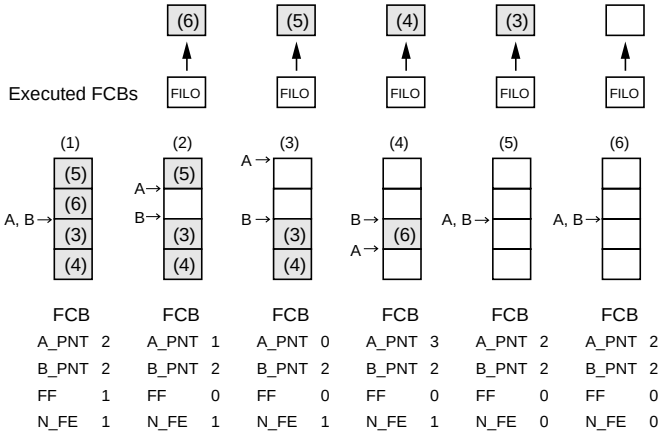
Function

(1) Setting to on the “ENABLE” causes the latest data (at the B_POINTER position in the FCB) to be read from the specified sequential file area into the area designated by the “OUT” and the A_POINTER to be decremented by 1. At this moment, if the FF flag (File Full) is on, it is set to off.
(2) The N_FE bit is set to off (to notify that the file gets empty) when A_POINTER catches up with B_POINTER.
(3) The “FCB” specifies the file control block which is used for controlling read processing.
(4) Specify the number of elements of the array connected to “OUT” for a “X_SIZE” (one-dimensional size of an array). Specify the two-dimensional size of the array for a “Y_SIZE.”
(5) Processing is flagged as invalid if an attempt is made to read an empty file.
(6) If the read is unsuccessful, the “ERROR” is set and held for one scan period.
(7) Available data type of “OUT” is an array of any data type except BOOL type. Normal operation is not guaranteed if an array of BOOL type is specified on “FILE.”
(8) “OUT” is overwritten when “ENABLE” is set to “ON” and the file is not empty.

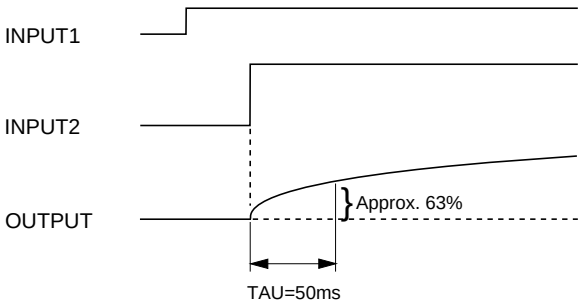
Note) If the result of “X_SIZE” and “Y_SIZE” multiplication is different from (larger than) the size of the array of arrays connected to “FILE,” any other variable area may be overwritten.

Example

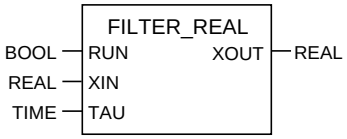
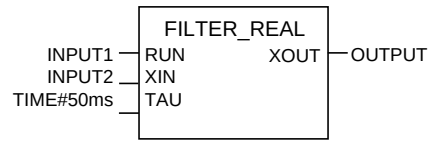
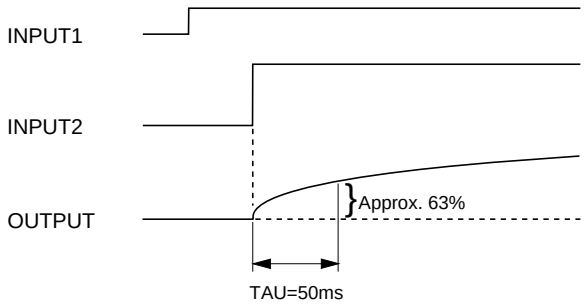
<Operation>



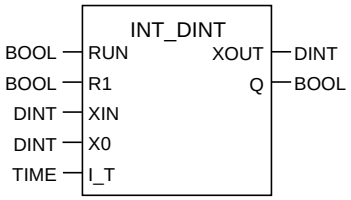
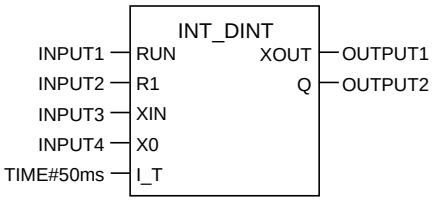
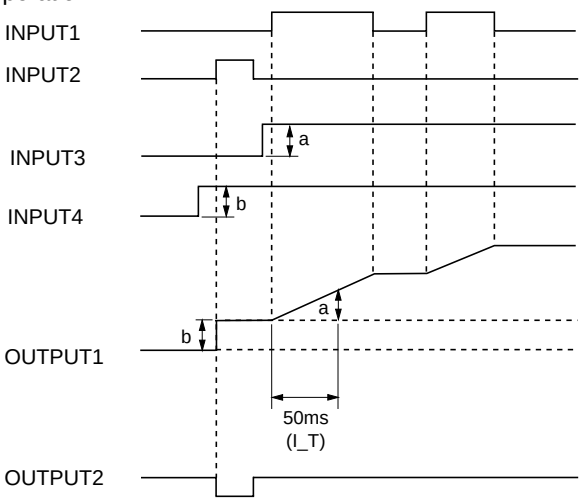
(17) Filter FILTER_DINT

Name, Symbol, Function		Example
Name	FILTER_DINT	<Programming example> 
Symbol		<Operation> 
Function	(1) FILTER_DINT filters the “XIN” and outputs the result on “XOUT” when the “RUN” is on. (2) The data from the “XIN” is transferred to “XOUT” without filtering if the “RUN” is off. (3) Available data type of “XIN” and “XOUT” is DINT type. (4) The “TAU” specifies the time constant of the filter. The larger the time constant value is, the gentler the slope of the “XOUT” curve is. The “XOUT” involves a large error if the time constant value is too small. (5) The “XOUT” carries the preceding value when the “RUN” is set on. (6) When the “TAU” is set to 0, the “XOUT” carries the value from the “XIN” as is. (7) The maximum allowable value of the filter time constant “TAU” is 24 hours (86400000 ms). (8) Set the filter time constant “TAU” to a value that is greater than the execution interval of this FB. (9) Enter the value that does not cause an overflow during operation or in the result. If an overflow occurs, the “XOUT” values may be unexpected. Note that for powerful CPUs, no overflow occurs if conditions 7 and 8 are met.	<Formulas> $XOUT = \frac{(XIN - XOUT') \Delta T + WORK'}{TAU} + XOUT'$ where WORK is the remainder of $\frac{(XIN - XOUT') \Delta T + WORK'}{TAU}$ XOUT’ and WORK’ denote the preceding value of XOUT and XWORK, respectively. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error becomes greater as the period (ΔT) becomes shorter. For standard CPUs, set a 2ms or greater interval for ΔT.

(18) Filter FILTER_REAL

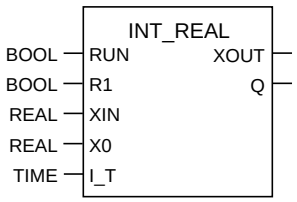
Name, Symbol, Function	Example
Name FILTER_REAL	<Programming example>
Symbol	
Function (1) FILTER_REAL filters the “XIN” and outputs the result in “XOUT” when the “RUN” is on. (2) The data from the “XIN” is transferred to “XOUT” without filtering if the “RUN” is off. (3) Available data type of “XIN” and “XOUT” is REAL. (4) The “TAU” specifies the time constant of the filter. The larger the time constant value is, the gentler the slope of the “XOUT” curve is. The “XOUT” involves a large error if the time constant value is too small. (5) The “XOUT” carries the preceding value when the “RUN” is set on. (6) When the input “TAU” is set to 0, the “XOUT” carries the value from the “XIN” as is. (7) The number of significant digits is 6. (8) The maximum allowable value of the filter time constant “TAU” is 24 hours (86400000 ms). (9) Set the filter time constant “TAU” to a value that is greater than the execution interval of this FB. (10) Enter the value that does not cause an overflow during operation or in the result. If an overflow occurs, the “XOUT” values may be unexpected.	 <Operation>  <Formulas> $XOUT = \frac{(XIN - XOUT') \Delta T}{TAU} + XOUT'$ XOUT’ denotes the preceding value of XOUT. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error gets larger as the period becomes shorter.

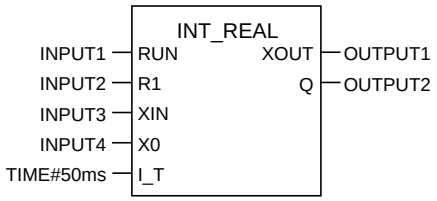
(19) Integrate INT_DINT

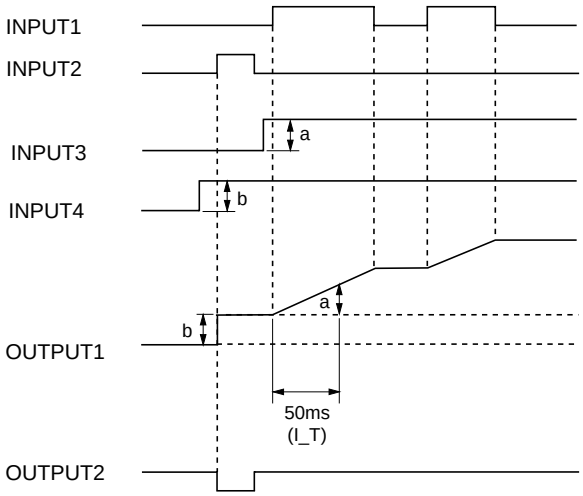
Name, Symbol, Function	Example
Name INT_DINT	<Programming example>
Symbol 	
Function (1) INT_DINT integrates the “XIN” and outputs the result in “XOUT” when the “RUN” is on. (2) The value of the “XOUT” is retained if the “RUN” is off. (3) The output is set to the initial value “X0” when the “R1” turns on. (4) The “Q” carries the NOT of the “R1.” (5) The “X0” must carry the initial value. (6) The “XOUT” carries the linear output such that it is equal to the value indicated by the “XIN” when the integration time designated by the “I_T” elapses, plus the initial value “X0.” (7) The “XOUT” carries the preceding value when the “RUN” is set on. (8) When the “I_T” is set to 0, the “XOUT” carries a value of 0. (9) The maximum allowable value of the integration time “I_TU” is 24 hours (86400000 ms). (10) Set the integration time “I_T” to a value that is greater than the execution interval of this FB. (11) Enter the value that does not cause an overflow during operation or in the result. If an overflow occurs, the “XOUT” values may be unexpected. Note that for powerful CPUs, no overflow occurs if conditions 9 and 10 are met.	<Operation>  <Formulas> $XOUT = \frac{(XIN\Delta T + WORK')}{I\_T} + XOUT'$ where WORK is the remainder of $\frac{XIN\Delta T + WORK'}{I\_T}$ XOUT’ and WORK’ denote the preceding value of XOUT and WORK, respectively. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error gets larger as the period (ΔT) becomes shorter. For standard CPUs, set a 2ms or greater interval for ΔT.

(20) Integrate INT_REAL

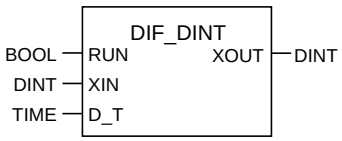
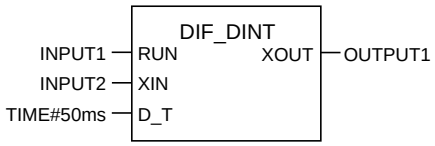
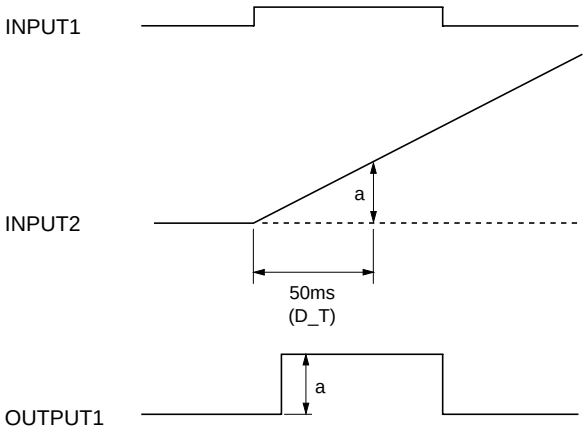
Name, Symbol, Function	Example
Name INT_REAL	<Programming example>
Symbol	
Function (1) INT_REAL integrates the “XIN” and outputs the result in “XOUT” when the “RUN” is on. (2) The value of the “XOUT” is retained if the “RUN” is off. (3) The output is set to the initial value “X0” when the “R1” turns on. (4) The “Q” carries the NOT of the “R1.” (5) The “X0” must carry the initial value. (6) The “XOUT” carries the linear output such that it is equal to the value indicated by the “XIN” when the integration time designated by the “I_T” elapses, plus the initial value “X0.” (7) The “XOUT” carries the preceding value when the “RUN” is set on. (8) When the “I_T” is set to 0, the “XOUT” carries a value of 0. (9) The number of significant digits is 6. (10) The maximum allowable value of the integration “I_TU” is 24 hours (86400000 ms). (11) Set the integration time “I_T” to a value that is greater than the execution interval of this FB. (12) Enter the value that does not cause an overflow during operation or in the result. If an overflow occurs, the “XOUT” values may be unexpected.	<Operation> <Formulas> $XOUT = \frac{XIN\ \Delta T}{I\_T} + XOUT'$ XOUT’ denotes the preceding value of XOUT. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error gets larger as the period becomes shorter.





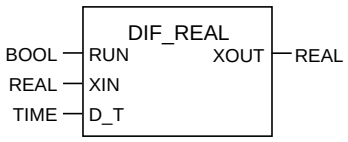


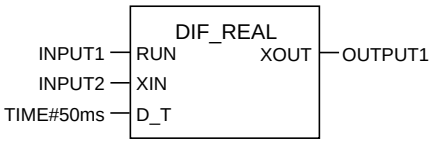
(21) Differentiate DIF_DINT

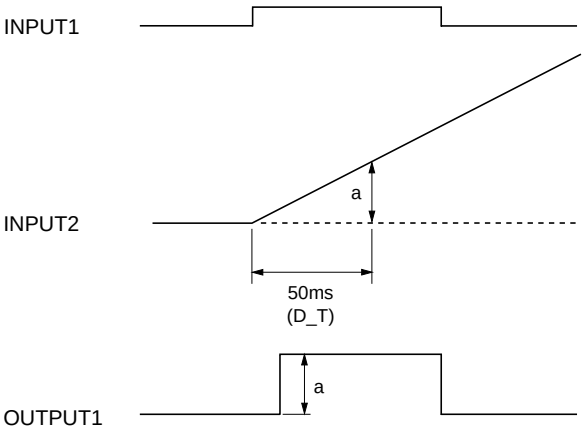
Name, Symbol, Function		Example
Name	DIF_DINT	<Programming example>
Symbol		
Function	(1) DIF_DINT computes a differential from the "XIN" and the input from the output generated during the preceding execution and outputs the result in "XOUT" when the "RUN" is on. (2) The "XOUT" carries a "0" when the "RUN" is off. (3) The "D_T" must specify the derivative time. (4) The DIF_DINT computes the variation of the "XIN" and outputs on "XOUT" the expected variation when the derivative time elapses. (5) The "XOUT" carries the output that is based on the variation that has been established since the time "RUN" is turned on. Consequently, "XOUT" is set to 0 when "RUN" is turned on. (6) When the "I_T" is set to 0, the "XOUT" carries a value of 0. (7) The maximum allowable value of the derivative time "D_T" is 24 hours (86400000 ms). (8) The valid number of output digits is six. (9) If the difference between current XIN and previous XIN is larger than six digits, an error may occur in a XOUT. (10) Enter a value that does not cause an overflow in D_T during operation (XIN-XM). If an overflow occurs, the "XOUT" values may be unexpected.	<Operation>  <Formulas> $XOUT = \frac{(XIN-XM)D\_T}{\Delta T}$ where XM is the value of XIN established in the preceding execution. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error gets larger as the period (ΔT) becomes shorter. For standard CPUs, set a 2ms or greater interval forΔT.

(22) Differentiate DIF_REAL

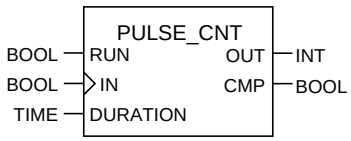
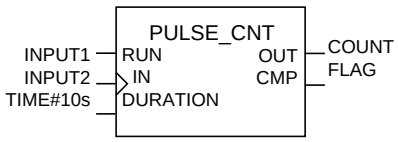
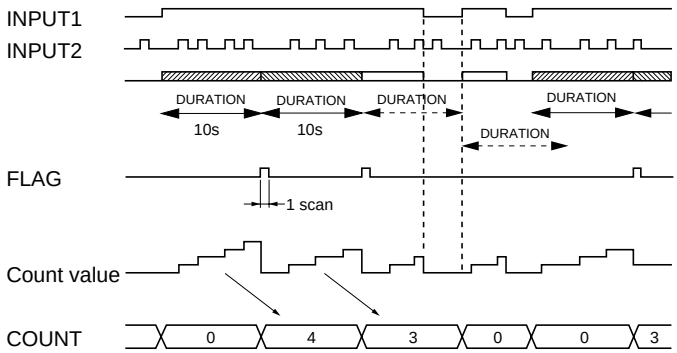
Name, Symbol, Function	Example
Name DIF_REAL	<Programming example>
Symbol	<Operation>
Function (1) DIF_REAL computes a differential from the "XIN" and the input from the output generated during the preceding execution and outputs the result in "XOUT" when the "RUN" is on. (2) The "XOUT" carries a "0" when the "RUN" is off. (3) The "D_T" must specify the derivative time. (4) The DIF_REAL computes the variation of the "XIN" and outputs, on "XOUT," the expected variation when the derivative time elapses. (5) The "XOUT" carries the output that is based on the variation that has been established since the time "RUN" is turned on. Consequently, "XOUT" is set to 0 when "RUN" is turned on. (6) When the "I_T" is set to 0, the "XOUT" carries a value of 0. (7) The number of significant digits is 6. (8) The maximum allowable value of the derivative time "D_T" is 24 hours (86400000 ms). (9) Enter a value that does not cause an overflow in D_T during operation (XIN-XM). If an overflow occurs, the "XOUT" values may be unexpected.	<Formulas> $XOUT = \frac{(XIN-XM)D\_T}{\Delta T}$ where XM is the value of XIN established in the preceding execution. ΔT is the execution period for this FB. Note: This FB is generally used by periodic tasks. The error gets larger as the period becomes shorter.



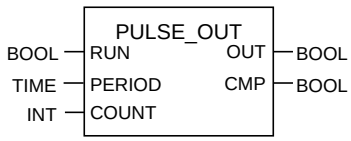
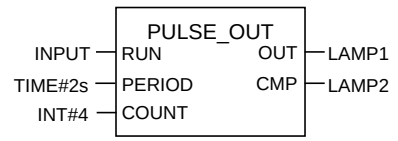




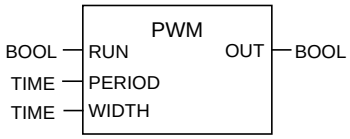
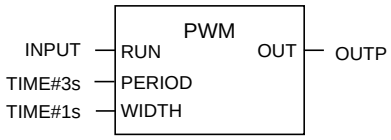
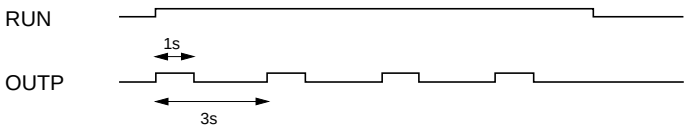
(23) Pulse count PULSE_CNT

Name, Symbol, Function	Example
Name PULSE_CNT **Symbol**  **Function** (1) When a “RUN” is set to “ON,” the pulses input at “IN,” the time specified in a “DURATION,” are counted, and INT type results are output to “OUT.” If the result exceeds an INT type of boundary value, the INT type of boundary value is output. When the time specified in “DURATION” has passed, a “CMP” is set to “ON” for a period of one scan. If a “RUN” is set to “ON,” counting is resumed. (2) The “OUT” retains the current value when the “RUN” is set to off. (3) The “OUT” is cleared to zero on the rising edge of the “RUN.” (4) If “DURATION” is set to 0 ms, “CMP” holds on while “RUN” is on.	<Programming example>  <Operation>  Note:1) Since “DURATION” processes are clocking in the same manner as that for the timer instruction, an error +0 - +2 in scan time occurs. 2) The pulses input to “IN” require a pulse width of two times or more the period for instruction execution.

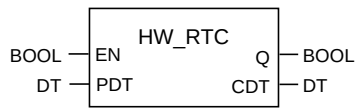
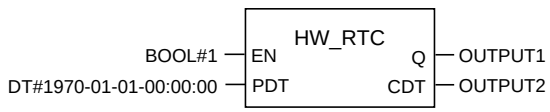
(24) Pulse output PULSE_OUT

Name, Symbol, Function	Example
Name PULSE_OUT **Symbol**  **Function** (1) When the “RUN” is set to on, PULSE_OUT outputs as many pulses of “PERIOD” period as specified by “COUNT.” The “CMP” turns on when the pulse is completed. When the “RUN” is set to off, the “CMP” is turned off. (2) PULSE_OUT stops the generation of pulses when the “RUN” is set to OFF. (3) When “COUNT” ≤ 0, pulses continue to be output while a “RUN” is set to “ON.” (4) The duty factor of the “OUT” is 1 to 1.	<Programming example>  <Operation>  Note:1) The period of “OUT” has an error +0 to +4 in scan time compared with “PERIOD.” 2) Specify sufficiently longer time for the preset period “PERIOD” compared with the period for instruction execution at two times or more).

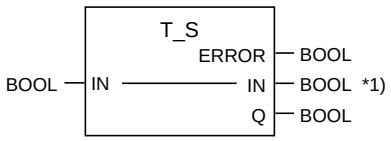
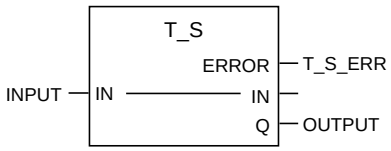
(25) Pulse PWM

Name, Symbol, Function	Example
Name: PWM Symbol:  Function: (1) PWM outputs pulses of the duration "WIDTH" at the period "PERIOD" when the "RUN" is set to on. The output is stopped when the "RUN" is set to OFF. (2) Do not set "PERIOD" ≤ "WIDTH;" otherwise, no "OUT" is output. Similarly, when "WIDTH" = 0, no "OUT" is output.	<Programming example>  <Operation>  Note :1) Since it processes clocking in the same manner as that for the timer instruction, an error +0 to +2 in scan time occurs. 2) Specify the values that satisfy the following condition for "PERIOD" and "WIDTH." "PERIOD" - "WIDTH" > (a period for one instruction execution)

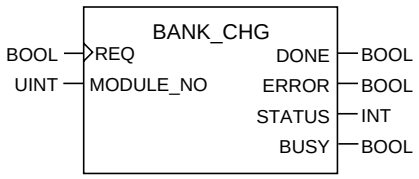
(26) Hardware RTC (Real-time Clock) HW_RTC (These functions are not supported in SPS.)

Name, Symbol, Function	Example
Name: HW_RTC Symbol:  Function: (1) HW_RTC writes the set value "PDT" into the hardware RTC (clock in the CPU module) on a low to high (0 to 1) transition of the "EN." If the set value is DT#1970-01-01-00:00:00, the current value of the hardware RTC is output in "CDT" as the current value. (2) The "Q" is set to 1 after the "PDT" value is written into the hardware RTC in the CPU module if the "EN" is on. If the set value is DT#1970-01-01-00:00:00, the "Q" is set to 1 immediately when "EN" is set high. (3) The value of the hardware RTC is output on the "CDT" while the "Q" is in the 1 state. (4) When an input "EN" is set to "ON," an "CDT" is not updated. Q is set to 0. Note: The valid preset value range is from DT#1970-01-01-00:00:01 to DT#2069-12-31-23:59:59.	<Programming example>  <Operation> The current calendar value is always output at "OUTPUT2" if the program is set up as shown in the above figure (preset value = DT#1970-01-01-00:00:00).

(27) Test & set T_S

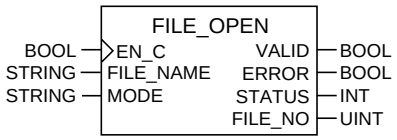
Name, Symbol, Function	Example
Name T_S	<Programming example>
Symbol  *1: When SPS is used, be sure to connect same variable to both IN and OUT terminals.	 <Operation> This FB is used when a resource is to be shared by two or more tasks. Since “INPUT” serves as a semaphore bit for the resource, it needs to be assigned to a global variable. The flag indicating whether the resource is obtained successfully is output on “OUTPUT.”
Function (1) T_S sets the variable “IN” (of BOOL type) to 1 and outputs the result on the “Q.” <Output “Q” state> 0: Setting completed 1: Setting unsuccessful (already set) (2) The set data need to be reset using the ST (Store) instruction. (3) If the test & set is unsuccessful, the output “ERROR” is set and held for one scan period.	Note: In the duplex system, to pass data to the bit variables used for semaphores at changeover between the operating and waiting CPUs: • Assign the bit variables for semaphores to the standard memory area to avoid equalization. (They are reset to 0 at changeover.) • When changeover occurs, make an attempt to get semaphores again.

(28) Change bank BANK_CHG (These functions are not supported in SPS.)

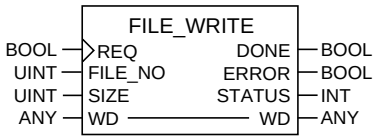
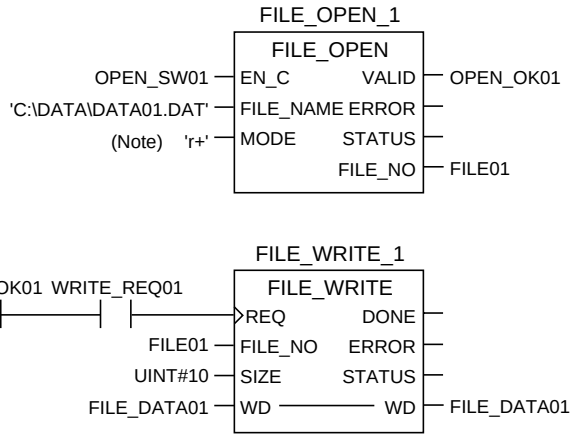
Name, Symbol, Function	Example
Name BANK_CHG **Symbol**  **Function** BANK_CHG is used to establish synchronization between the data in the broadcast communications area which is used by the P/PE link modules. (1) A request to switch the bank is issued to the P/PE link module designated by the P/PE link module SX station number “MODULE_NO” on the rising edge of the bank change request signal “REQ.” (2) “BUSY” turns on when a bank change request is issued. (3) When bank change processing is completed, the output “DONE” or “ERROR” is set and held for one scan period and “BUSY” is set off. (4) When accessing the broadcast communications area in the application program, it is possible to establish data synchronization by performing a read or write after “DONE” is set on. Note:1) Two or more bank change FBs may be programmed for one CPU. The sequence of processing from the issuance of a change request to the verification of the completion of bank change must be carried out within a single FB. 2) Install the CPU module and P/PE-link module, between which the bank is to be changed, on the same processor bus (on the same base). When they are not installed on the same processor bus, an error occurs during application program execution if this FB is executed.	<Terminal description> Input REQ: Bank change request flag MODULE_NO: SX bus station number of the P/PE link module for which the bank is to be changed (246 or 245). Output DONE: Normal completion flag (held for one scan period on completion) ERROR: Error flag (held for one scan period on an error) STATUS: Status BUSY: Held on during processing. <Status (STATUS) summary> (1) SX bus station number error (code: 64) An SX bus station number which is not for P/PE-link module is specified. (2) Duplicate bank change request (code: 65) Two or more bank change requests are issued by one CPU. (3) Bank change processing processor bus error (code: 66) An error occurred while accessing the processor bus during bank change processing. <Programming example> Refer to the user's manual for the P/PE link modules (FEH203) for programming examples.

2-5-12 Original FBs dedicated to SPS

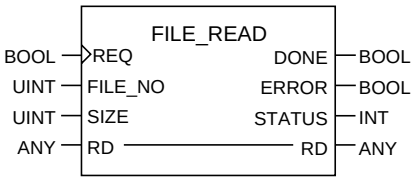
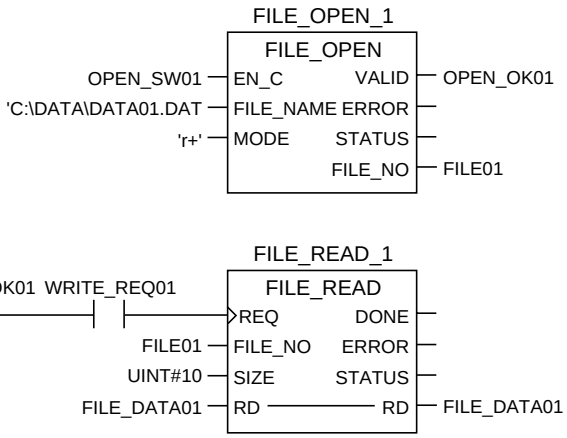
(1) File open FILE_OPEN

Name, Symbol, Function	Example
Name FILE_OREN	<Terminal description> Input EN_C: Request to open file (Request to open at the leading edge of signal; request to close at the trailing edge of signal. FILE_NAME: Specify the name of the file to be opened, including drive name and path name. MODE: Specify file access mode.
Symbol  	<Mode list> 'r': Open a file in read mode. If specified file does not exist or cannot be found, "file open error" occurs. 'w': Open a new file in write mode. If specified file already exists, its content is destroyed. 'a': This mode is used to write new data by adding it following existing data. 'r+': This mode is used to read and write an existing file. 'w+': This mode is used to read and write a new file. If specified file already exists, its content is destroyed. 'a+': This mode is used to read and write by adding to existing data.
Function This FB is used to open data files stored in the personal computer. This FB is used in combination with FILE_READ, FILE_WRITE and FILE_SEEK that are explained in the following paragraphs. Maximum 32 files can be opened at the same time. (1) When EN_C is set ON, file open operation is started. (File open operation does not complete within one scan cycle.) (2) When file open operation is completed successfully, VALID is set ON and file numbers (1 to 32) are output to FILE_NO. In this condition, FILE_READ, FILE_WRITE and FILE_SEEK can be used. (3) If file open operation ended unsuccessfully, ERROR is turned ON for one scan cycle, and an error code is output to STATUS. (4) When EN_C is set OFF, file close operation is started. (File close operation does not complete within one scan cycle.) When file close operation is completed, VALID is set OFF. Note: One FILE_OPEN function block is used for one file. One FILE_OPEN function block cannot be used for multiple files.	Output VALID: File open operation is completed successfully (set ON when successful) ERROR: File open operation error (Set ON for one scan cycle when unsuccessful) STATUS: Status error FILE_NO: Opened file management number (1 to 32) <Status (STATUS) summary> (1) File open error (code: 101) MODE: File open was executed by "r" or "r+," but specified file could not be found. (2) Maximum file open (code: 201) It was attempted to open more than 32 files.

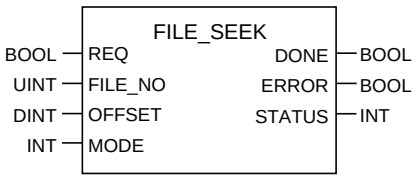
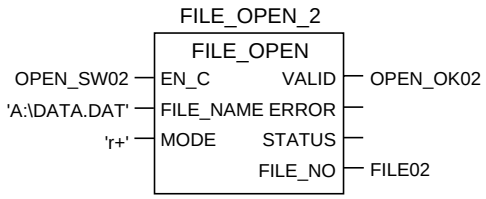
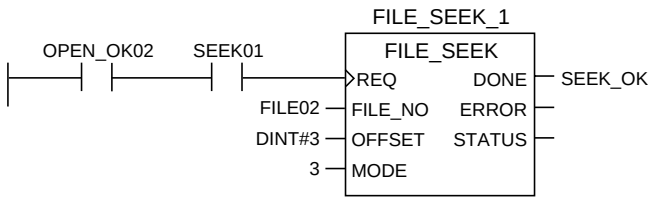
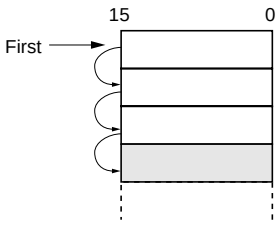
(2) File data write FILE_WRITE

Name, Symbol, Function	Example
Name FILE_WRITE **Symbol**  **Function** This function block is used to write data in a file that is opened by FILE_OPEN. (1) When REQ is set ON, the content of WD is written in the file that is specified by FILE_OPEN at the leading edge of the signal. This operation does not complete within one scan cycle. (2) When data writing operation is completed successfully, DONE is set ON for one scan cycle. (3) If data writing operation is completed unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS. Notes: 1) Maximum 4096 words of data can be written at a time by FILE_WRITE. 2) REQ is the input that takes effect at the leading edge of the signal, but if REQ is set ON and then set ON again before DONE or ERROR is set ON (before one writing operation is completed), the request is ignored.	<Terminal description> Input REQ: Request to write file data FILE_NO: File No. opened by FILE_OPEN SIZE: Write data size of WD (in units of word) Output DONE: Writing is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) I/O WD: Write data storage variable <Status (STATUS) summary> (1) File access error (code: 102) • The file existed when FILE_OPEN was executed but cannot be found when this FB is executed. • The write data cannot be written due to the shortage in capacity of the file. • Writing was attempted on the file that is opened by MODE = 'r.' (2) File number error (code: 207) • A file number is used that is not opened by FILE_OPEN. <Programming example> 10 words of the content of FILE_DATA01 are written in the "\DATA\DATA01.DAT" file in the C drive of the personal computer.  Note: When both reading and writing operations are to be executed on one file, use 'r+' mode.

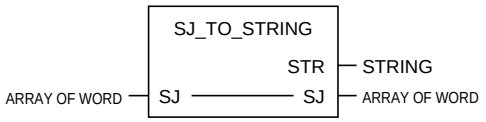
(3) File data read FILE_READ

Name, Symbol, Function	Example
Name FILE_READ **Symbol**  **Function** This function block is used to read data from a file that is opened by FILE_OPEN. (1) When REQ is set ON, the content of the file that is specified by FILE_OPEN starts to be read out at the leading edge of the signal. This operation does not complete within one scan cycle. (2) When file data reading is completed successfully, DONE is set ON for one scan cycle. (3) If file data reading ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS. Notes: 1) Maximum 4096 words of data can be read out at a time by FILE_READ. 2) REQ is the input that takes effect at the leading edge of the signal, but if REQ is set ON and then set ON again before DONE or ERROR is set ON (before one reading operation is completed), the request is ignored.	<Terminal description> Input REQ: Request to read file data FILE_NO: File No. opened by FILE_OPEN SIZE: Read data size of RD (in units of word) Output DONE: Reading is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) I/O RD: Read data storage variable <Status (STATUS) summary> (1) File access error (code: 102) • The file existed when FILE_OPEN was executed but cannot be found when this FB is executed. (For example, floppy disk was changed.) • Failed to read data from specified file. • Reading was attempted on the file that is opened by MODE = 'w' or 'a.' (2) File number error (code: 207) • A file number is used that is not opened by FILE_OPEN. <Programming example> 10 words of data are read into FILE_DATA01 from the "\DATA\DATA01.DAT" file in the C drive of the personal computer. 

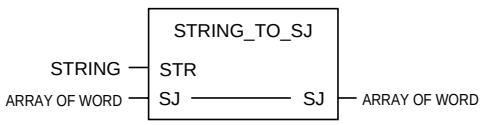
(4) File pointer seek FILE_SEEK

Name, Symbol, Function	Example
Name FILE_SEEK **Symbol**  **Function** This function block is used to move the pointer for object file. In general, file reading and writing operations are executed from the top of the file, but with this function block, you can access arbitrary position in the file. (1) When REQ is set ON, the pointer for the file that is specified by FILE_OPEN moves to a position that is specified by MODE and OFFSET. File pointer moving operation does not complete within one scan cycle. (2) When file pointer moving operation is completed successfully, DONE is set ON for one scan cycle. (3) If file pointer moving operation ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS. Note: REQ is the input that takes effect at the leading edge of the signal, but if REQ is set ON and then set ON again before DONE or ERROR is set ON (before one changing operation is completed), the request is ignored.	<Terminal description> Input REQ: Request to change file pointer FILE_NO: File No. opened by FILE_OPEN OFFSET: Moving length (words) MODE: Initial position of file pointer <Mode list> 1: Current position of file pointer 2: End of file 3: Top of file Output DONE: File pointer change is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) STATUS: Status <Status (STATUS) summary> (1) File access error (code: 102) • The file existed when FILE_OPEN was executed but cannot be found when this FB is executed. (For example, floppy disk was changed.) (2) File number error (code: 207) • A file number is used that is not opened by FILE_OPEN. (3) Parameter error (code: 177) • A value other than the constants for MODE was input.   (Operation) File pointer is moved from the top to 4th. 

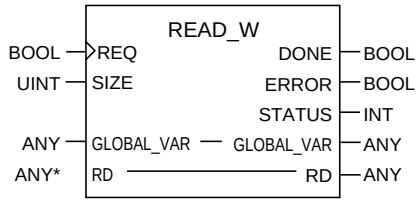
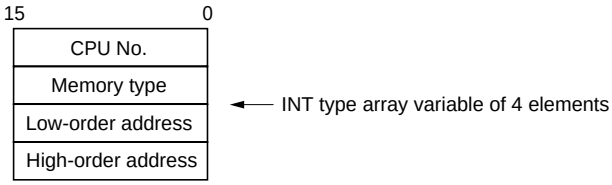
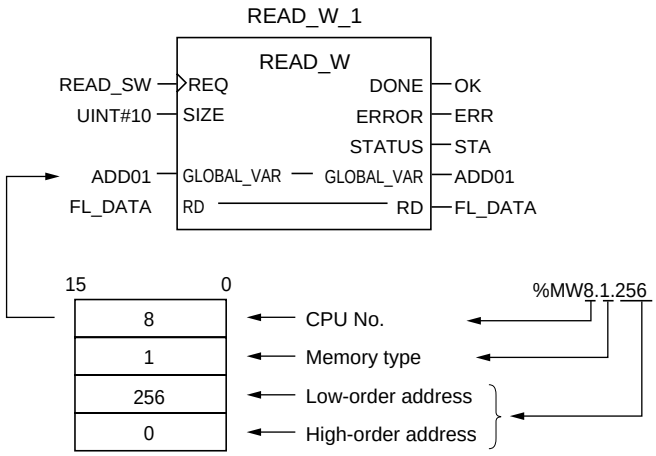
(5) Shifted JIS code to character string conversion SJ_TO_STRING

Name, Symbol, Function	Example
Name SJ_TO_STRING **Symbol**  **Function** (1) Convert the shifted JIS code that is defined as an array of WORD type into the data of STRING type. (2) NULL code (“00” or “00 00”) is necessary at the end of shifted JIS code. (3) When a shifted JIS code that exceeds 80 single-byte characters or 40 double-byte characters is input to SJ, maximum 80 single-byte characters (40 double-byte characters) of data are output to STR.	<Operating example> ARRAY OF WORD A0 82 A2 82 A4 82 42 41 60 82 00 00 Convert STRING ’あいうABA’ Note: If a code that is not included in shifted JIS code is input to SJ, NULL code is output for the part.

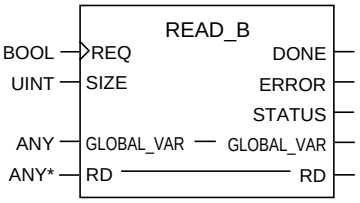
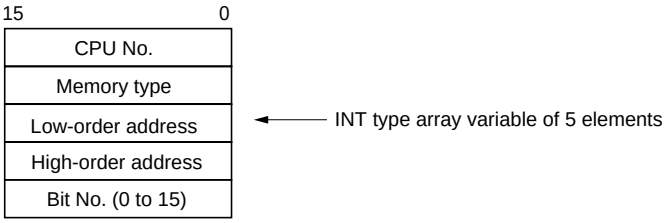
(6) Character string to shifted JIS code conversion STRING_TO_SJ

Name, Symbol, Function	Example
Name STRING_TO_SJ **Symbol**  **Function** (1) Convert STRING type data that is input to STR into the shifted JIS code that is defined as an array of WORD type data. NULL code is added at the end. (2) When output array is sufficiently greater in capacity than input, indefinite data may be added following NULL code (00). (3) When output array is smaller in capacity than input, other areas are rewritten.	<Operating example> STRING ’あいうABA’ Convert ARRAY OF WORD A0 82 A2 82 A4 82 42 41 60 82 XX 00 XX XX XX XX Note: Because NULL code is added at the end, the size of output array must be one word greater than the number of input characters.

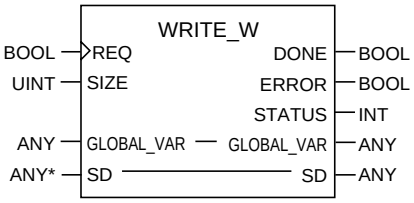
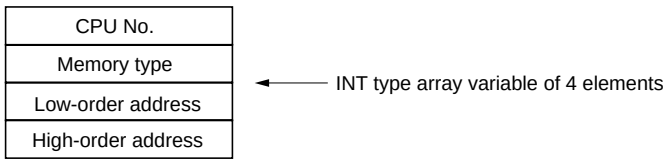
(7) Direct read READ_W

Name, Symbol, Function	Example
Name READ_W	<Terminal description> Input REQ: Start to read SIZE: Size of read data (in units of word) Output DONE: Reading is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) STATUS: Status I/O GLOBAL_VAR: Specify the memory to be accessed in the following format:
Symbol  * Data types other than BOOL type, BOOL type array and STRING type	
Function This function block is used to read the memory of a resource (CPU, FL-net, etc.) in same configuration (1) At the leading edge (0 to 1) of REQ, the data of the memory that is specified by GLOBAL_VAR is read out to RD. (The reading operation does not complete within one scan cycle.) (2) When data reading operation is completed successfully, DONE is set ON for one scan cycle. (3) If data reading operation ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS. Note: If the number of words to be read that is specified by SIZE is greater than the size of the variable that is connected to RD, the content of other variable is rewritten. Be careful so that the number of words to be read which is specified by SIZE does not exceed the size of the variable connected to RD.	<Status (STATUS) summary> (1) Parameter error (code: 177) • When 0 (zero) is input to SIZE. <Programming example> 10 words of data are read from the common memory %MW8.1.256 in the FL-net module of CPU No. 8.  * FL_DATA must be a variable of array data type that is greater than 10 words.

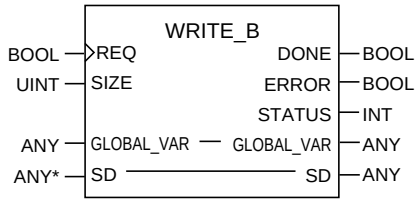
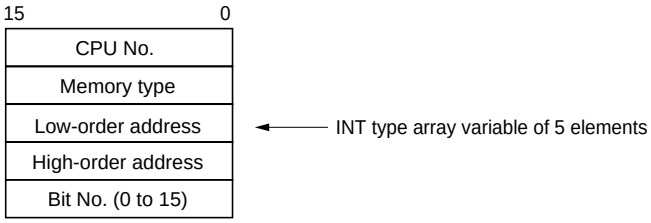
(8) Direct read READ_B

Name, Symbol, Function	Example
Name READ_B	<Terminal description> Input REQ: Start to read SIZE: Size of read data (in units of word) Output DONE: Reading is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) STATUS: Status I/O GLOBAL_VAR: Specify the memory (top address) to be accessed in the following format:
Symbol  * BOOL type, array data type of BOOL type	 RD: Stores the read data.
Function This function block is used to read the memory of a resource (CPU, FL-net, etc.) in same configuration. (1) At the leading edge (0 to 1) of REQ, the data of the memory that is specified by GLOBAL_VAR is read out to RD. (The reading operation does not complete within one scan cycle.) (2) When data reading operation is completed successfully, DONE is set ON for one scan cycle. (3) If data reading operation ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS. Note: If the quantity of data to be read that is specified by SIZE is greater than the size of the variable that is connected to RD, the content of other variable is rewritten. Be careful so that the quantity of data to be read which is specified by SIZE does not exceed the size of the variable connected to RD.	<Status (STATUS) summary> (1) Parameter error (code: 177) • When 0 (zero) is input to SIZE. • When a value other than 0 to 15 is input for bit No.

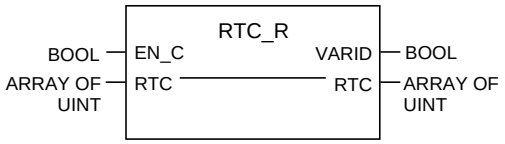
(9) Direct write WRITE_W

Name, Symbol, Function	Example
Name WRITE_W	<Terminal description> Input REQ: Start to write SIZE: Size of write data (in units of word) Output DONE: Writing is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) STATUS: Status I/O GLOBAL_VAR: Specify the memory (top address) to be accessed in the following format:
Symbol  * Data types other than BOOL type, BOOL type array and STRING type	 SD: Stores the data to be written.
Function This function block is used to write data in the memory of a resource (CPU, FL-net, etc.) in same configuration. (1) At the leading edge (0 to 1) of REQ, data is written in the memory that is specified by GLOBAL_VAR. The data to be written is prepared in SD. (2) When data writing operation is completed successfully, DONE is set ON for one scan cycle. (3) If data writing operation ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS.	<Status (STATUS) summary> (1) Parameter error (code: 177) • When 0 (zero) is input to SIZE.

(10) Direct write WRITE_B

Name, Symbol, Function	Example
Name WRITE_B	<Terminal description> Input REQ: Start to write SIZE: Size of write data (in units of word) Output DONE: Writing is completed successfully (set ON for one scan cycle when completed successfully) ERROR: Error flag (Set ON for one scan cycle when ended unsuccessfully) STATUS: Status I/O GLOBAL_VAR: Specify the memory (top address) to be accessed in the following format:
Symbol  * BOOL type, array data type of BOOL type	 SD: Stores the data to be written. <Status (STATUS) summary> (1) Parameter error (code: 177) • When 0 (zero) is input to SIZE. • When a value other than 0 to 15 is input for bit No.
Function This function block is used to write data in the memory of a resource (CPU, FL-net, etc.) in same configuration. (1) At the leading edge (0 to 1) of REQ, data is written in the memory that is specified by GLOBAL_VAR. The data to be written is prepared in SD. (2) When data writing operation is completed successfully, DONE is set ON for one scan cycle. (3) If data writing operation ended unsuccessfully, ERROR is set ON for one scan cycle, and an error code is output to STATUS.	

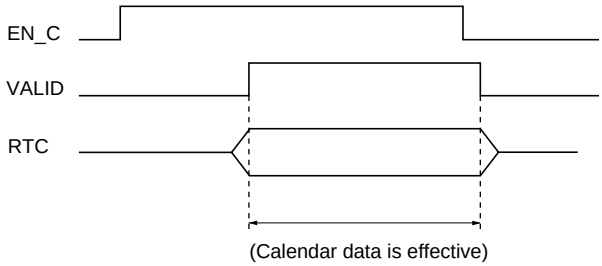
(11) Calendar read RTC_R

Name, Symbol, Function	Example
Name RTC_R	<Terminal description> Input EN_C: Request to read calendar data Output VARID: Reading completed I/O RTC: Read data storage area The data type of the variable that is connected to RTC must be a 7-ement array of UNIT type.
Symbol  	
Function (1) When EN_C is set ON, the calendar data stored in the personal computer is read out to be stored in RTC. When the data storage is completed, VARID is set ON. (2) To change the calendar data that is stored in the personal computer, do on Windows.	

Year
Month (1 to 12)
Day (1 to 31)
Hours (0 to 24)
Minutes (0 to 59)
Seconds (0 to 59)
Day of the week (0 to 6)

0: Sunday
1: Monday
2: Tuesday
3: Wednesday
4: Thursday
5: Friday
6: Saturday

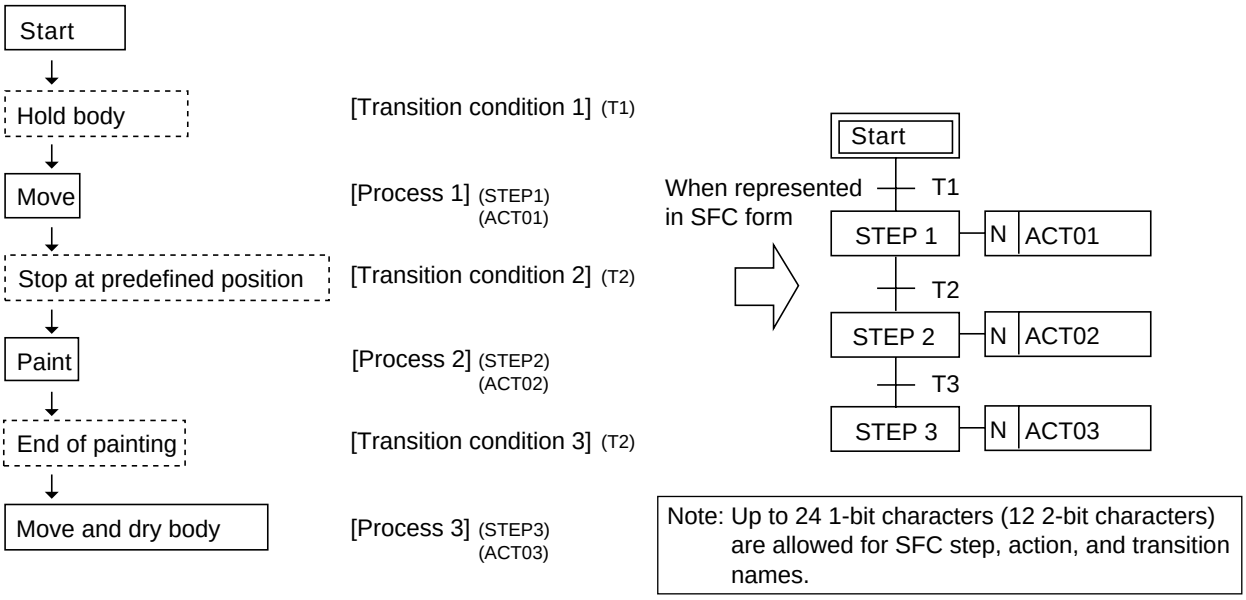
<Operation>



2-6 SFC Elements

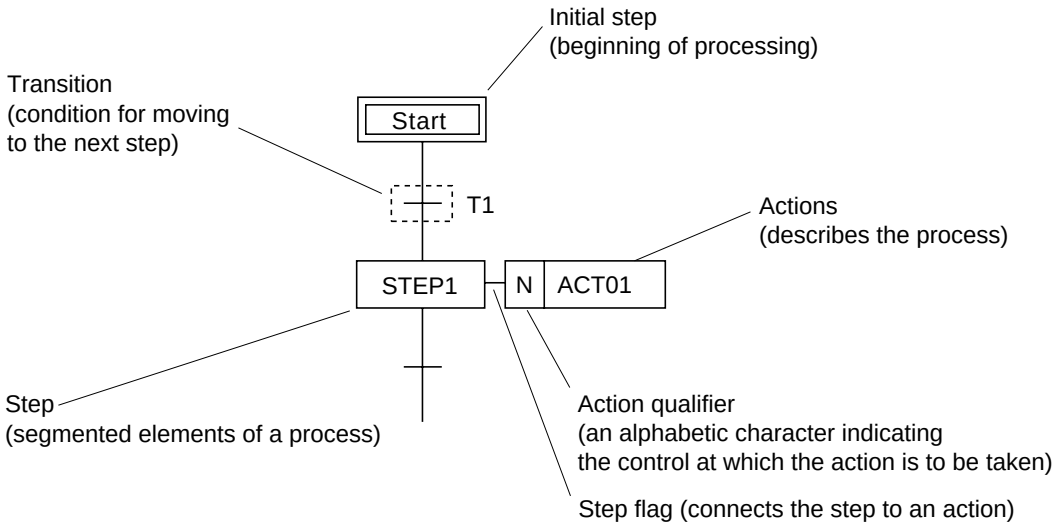
An SFC is a program that specifies the operation of a plant or machine in flowchart form. Since it allows the flow of processes to be programmed in a natural manner, it facilitates the programmer to develop programs efficiently and to grasp the flow of a program as the flow of processes.

For example, the process of painting automobiles can be depicted as shown below in an SFC form.



Note: Up to 24 1-bit characters (12 2-bit characters) are allowed for SFC step, action, and transition names.

The elements of an SFC have the following meanings:



Each action or transition must be assigned to a BOOL variable or programmed using the IL, ST, LD, or FB language.

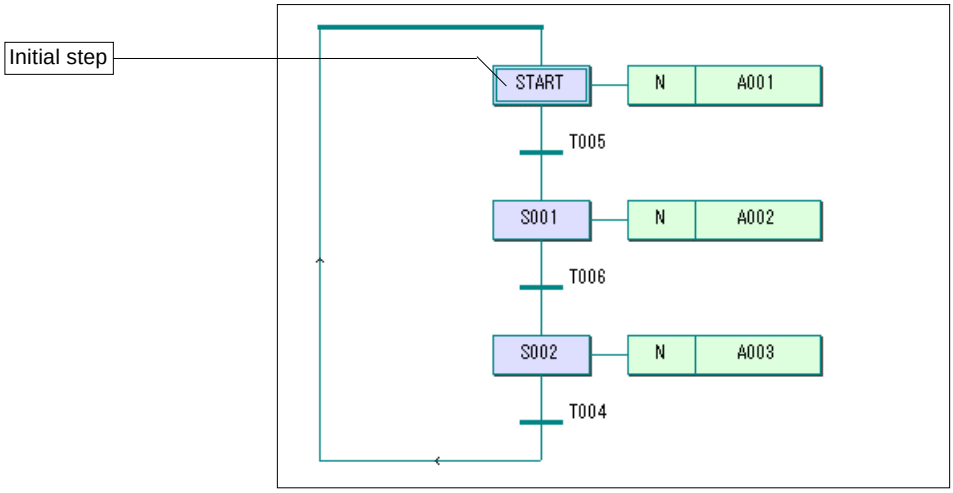
<Number of elements/POU>

No. of steps	110
No. of transitions	110
No. of actions	165

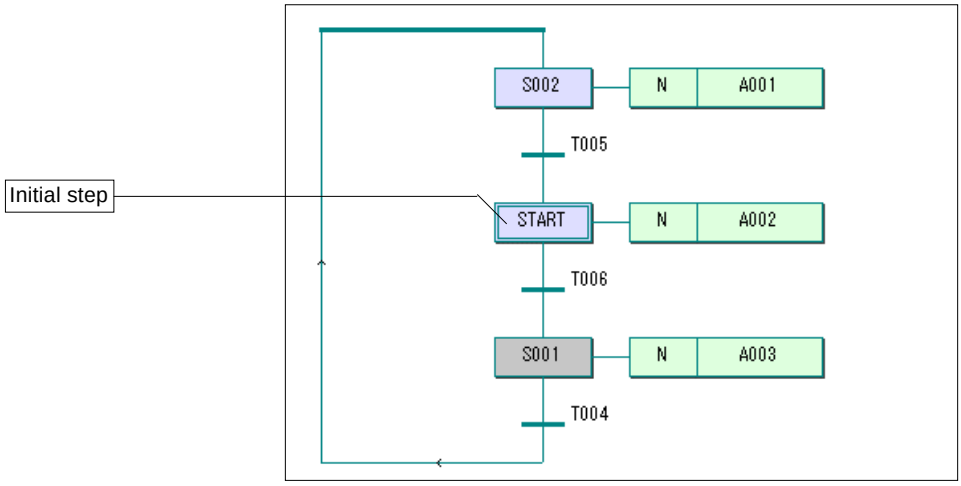
2-6-1 SFC elements

(1) Initial step

This step activates the SFC network at the beginning of program execution. Only one initial step is required for an SFC network.

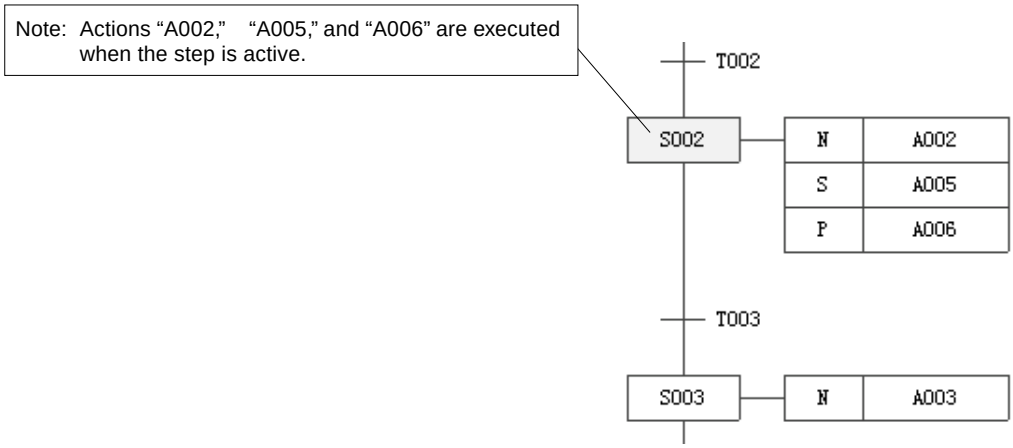


The initial step can be programmed in the middle of the SFC network.



(2) Normal step

A normal step represents one process in the SFC program. A normal step may be in the active or inactive state. When a normal step is in the active state, it executes actions according to the control of the action qualifiers.

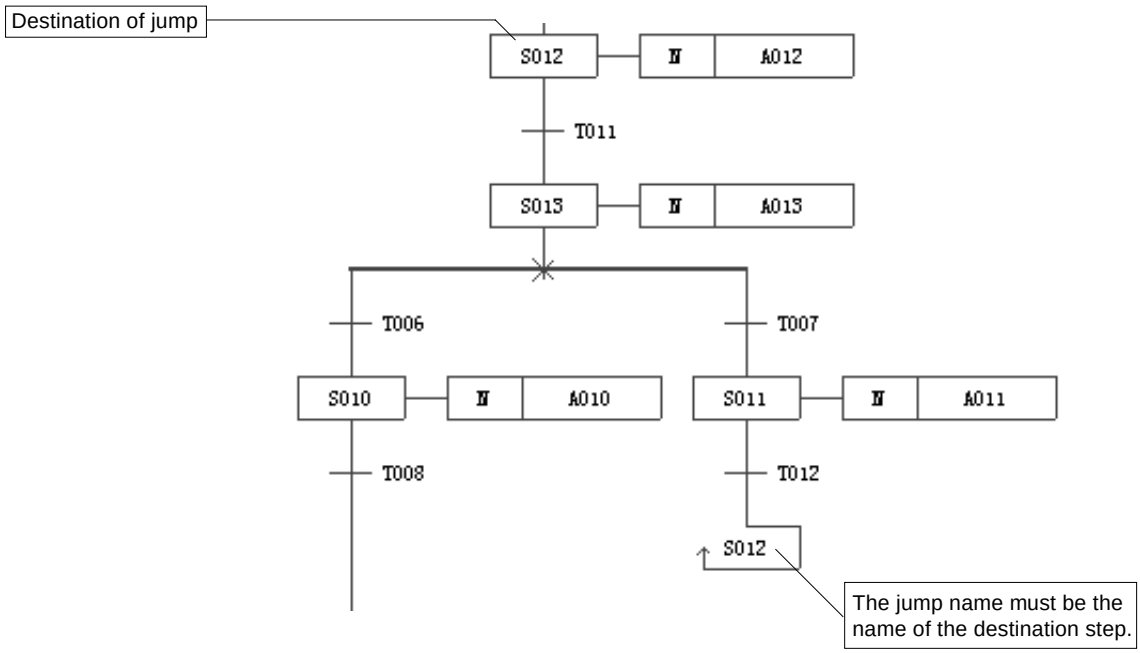


As shown in the above figure, one step may be assigned two or more actions. There is no limit to the number of actions that can be assigned to one step.

Note: The state of a step is represented by its status. For example, the flag of “S002” in the above figure is described as “S002.X.”

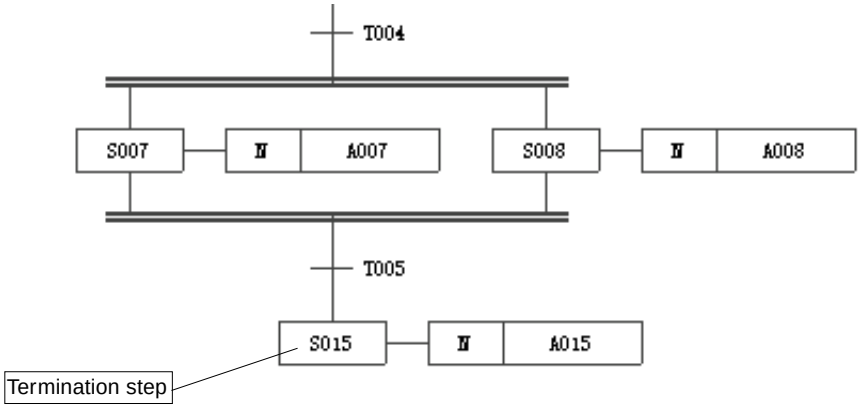
(3) Jump

A jump is a step whose next sequential step is the destination of the jump within the same worksheet or on another worksheet in the same POU. Execution never jumps into another POU. No action can be assigned to a jump.



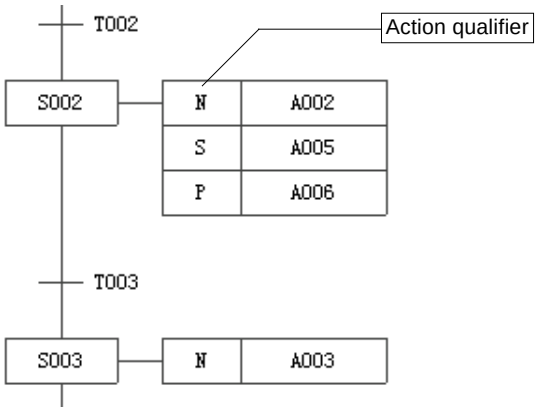
(4) Termination step

A termination step is one that has no following transition. It is used to mark the end of an SFC program. A termination step may be assigned an action which can be used for the termination processing of the application.



(5) Action/action qualifier

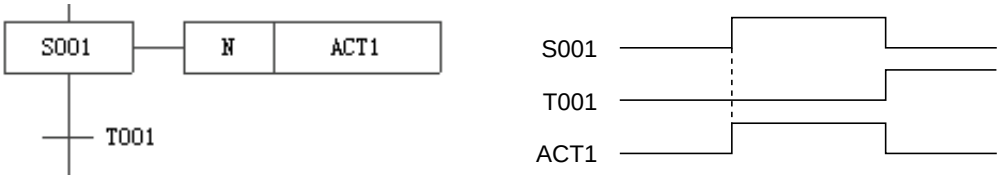
An action is a real program which is assigned to the initial, normal, and termination steps and which is executed on the object to be controlled. In an action, the programmer can assign a BOOL variable, or a program in IL, ST, LD, or FBD language.



An action qualifier specifies the timing of execution or suppression of an action according to the active or inactive state of the step. There are nine types of action qualifiers.

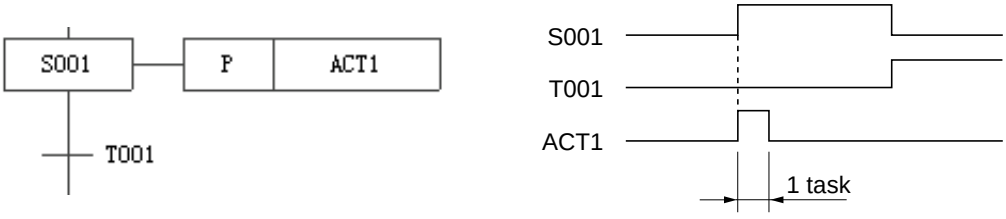
1) N (non-stored)

The action is repeated while the step is active. (Note)

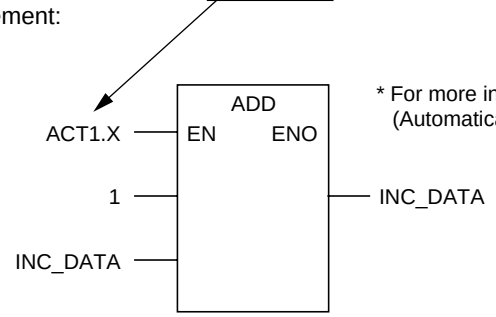


2) P (Pulse)

The action is executed only once when the step is activated.



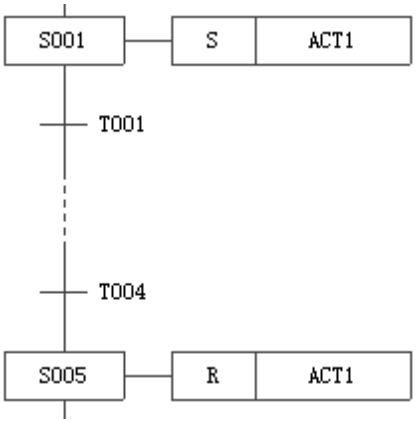
Note: IEC standard stipulates, “Action instruction statement or network must finally be executed at the trailing edge of action qualifier.” Therefore, the action of P qualifier is executed twice when the step becomes active.
To make it executed only once, use “action flag” as an interlock.
Example of increment:

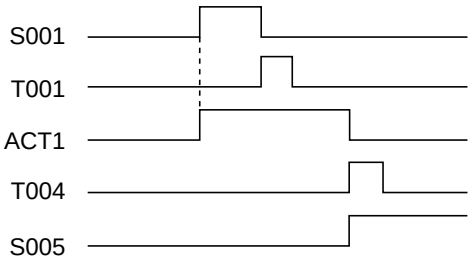


* For more information about action flag, refer to 2-6-3 (Automatically generated SFC variables).

3) S (Set)

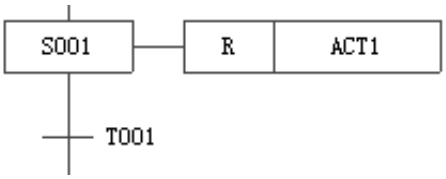
The execution of the action is started when the step is activated, but the action is executed repeatedly when the step is deactivated. The execution of the step is stopped using the R qualifier.

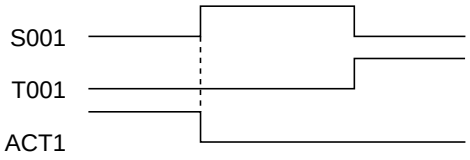




4) R (Overriding reset)

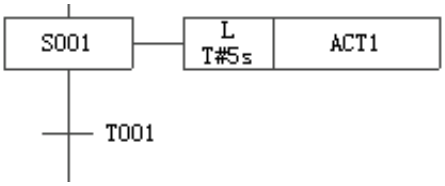
This qualifier is used in combination with the S, SD, DS, or SL qualifier. The action is stopped if a specified action has been executed in advance by an S, SD, DS, or SL qualifier when this step is activated. If the action is a BOOL variable, the variable is reset to "0." No action is taken (equivalent to being assigned no action) if the specified action is not in execution.

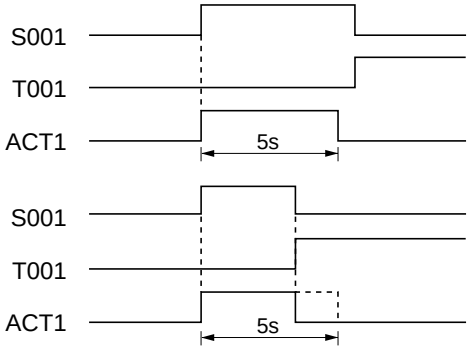




5) L (Time limited)

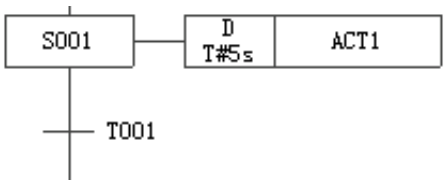
The action is started and repeated until a specified time has elapsed once the step is activated. The execution of the action is stopped after the elapse of the specified time or when the step is deactivated within the specified time.

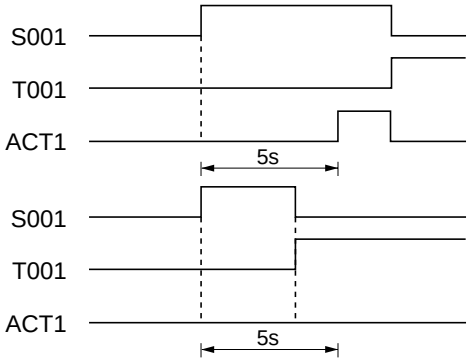




6) D (Time delayed)

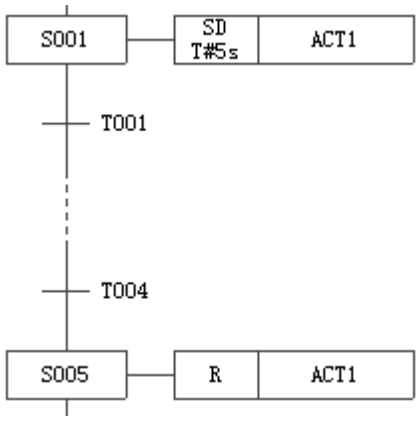
The action is started after a preset time after the step is activated. The action is executed repeatedly until the step is deactivated. The action is not executed if the step is deactivated before the preset time is reached.

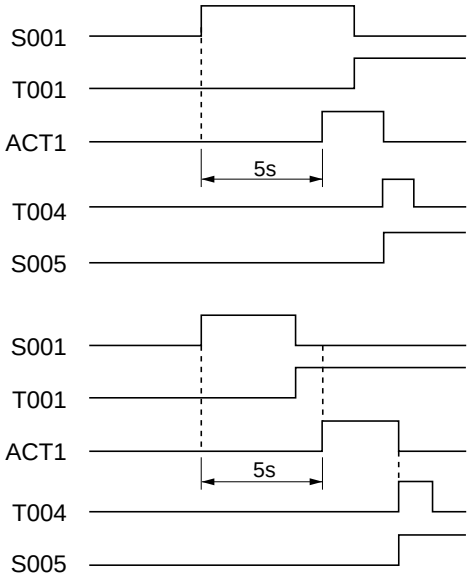




7) SD (Stored and time delayed)

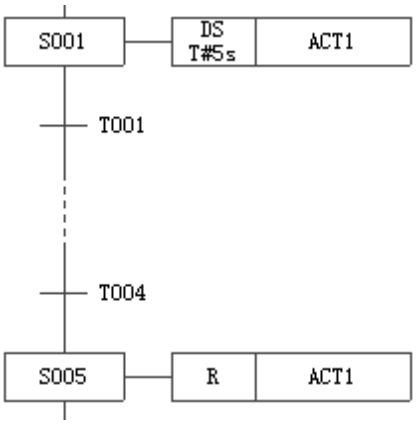
The action is started after a preset time after the step is activated. The action is started after the preset time has elapsed even when the step is deactivated within the preset time. The action repeatedly continues execution until it is reset by the R qualifier.

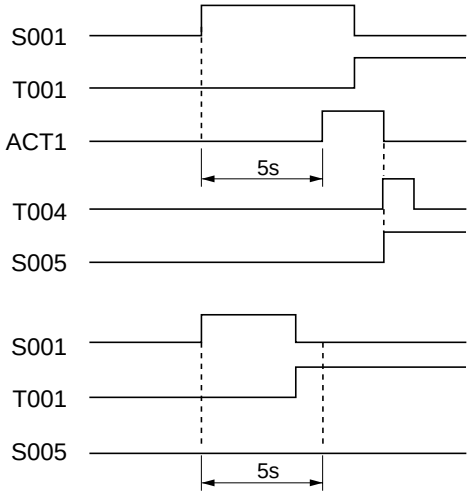




8) DS (Delayed and stored)

The action is started only when the step is held active for longer than a preset time. The action repeatedly continues execution until it is reset by the R qualifier. The action is not executed if the step is deactivated before the preset time is reached.

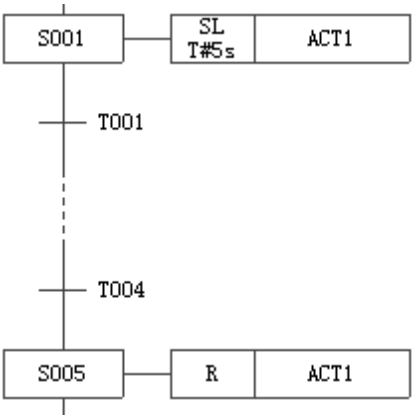


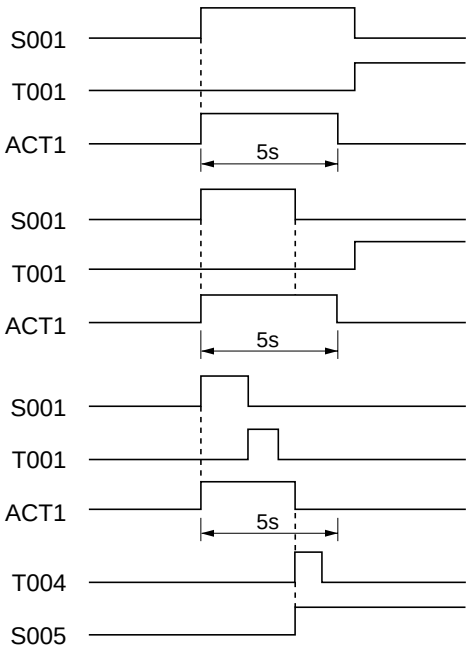


9) SL (Stored and time limited)

When the step is activated, the action is started and its execution is repeated until a preset time is reached. The action is stopped when the preset time is reached. This control is not affected even when the step is deactivated (differs from the L qualifier in this respect).

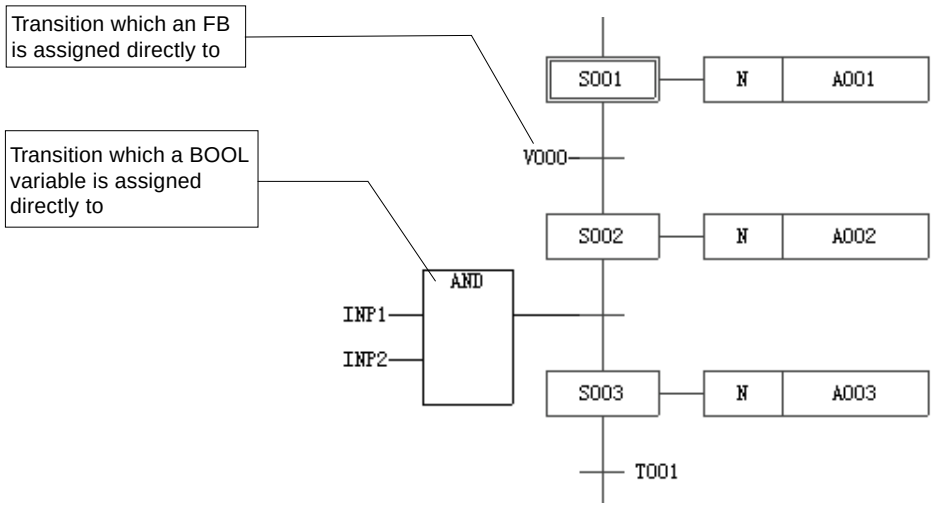
The R qualifier is used to stop the execution of the action within the preset time.





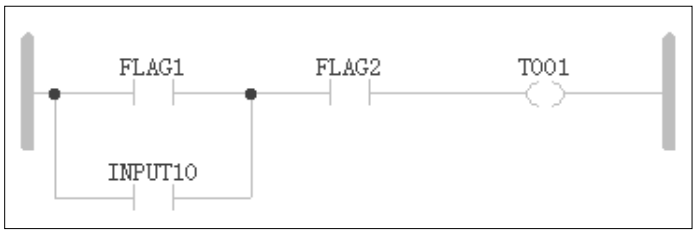
(6) Transition

Only one transition is programmed between two adjacent steps and specifies the conditions for making a transition from the preceding step to the following step. A transition condition is defined by assigning a BOOL variable or programming by either IL, ST, LD, or FBD language.



The user can assign a BOOL variable or FB to a transition as shown in the above figure. Programming as shown below when programming by an IL, ST, LD, or FBD language for a transition.

<Sample program of transition condition to transition“T001”>

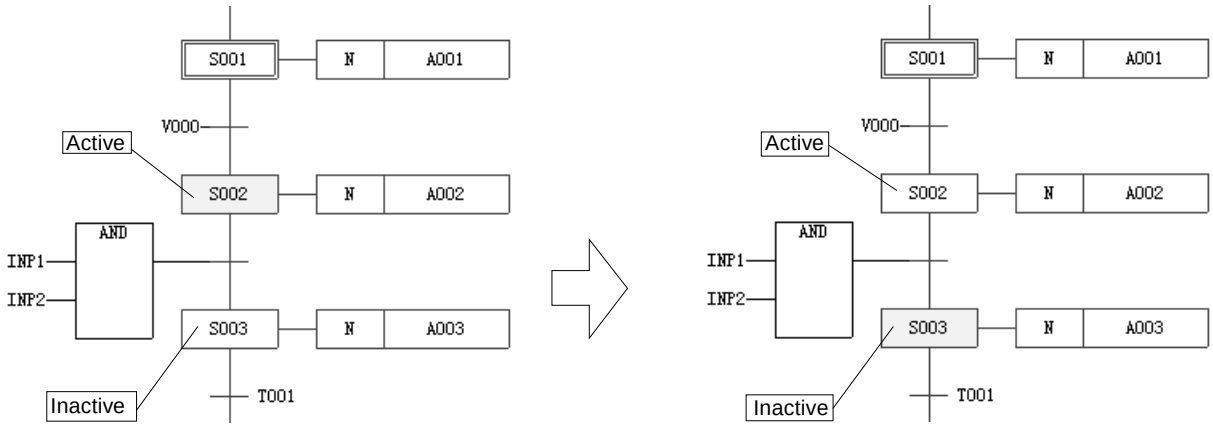


2-6-2 Step transition

The process of transferring the active state from one step to another is called “transition.” When the current step is active and the transition condition for the next step is turned on, the current step is deactivated and the next step is activated.

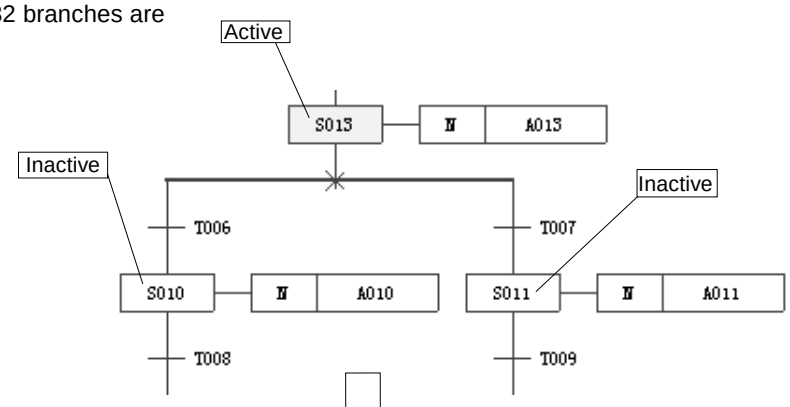
(1) Single-flow transition

In the example shown below, a transition to S003 occurs when S002 is active and both inputs of transition condition INP1 and INP2 are set to 1.

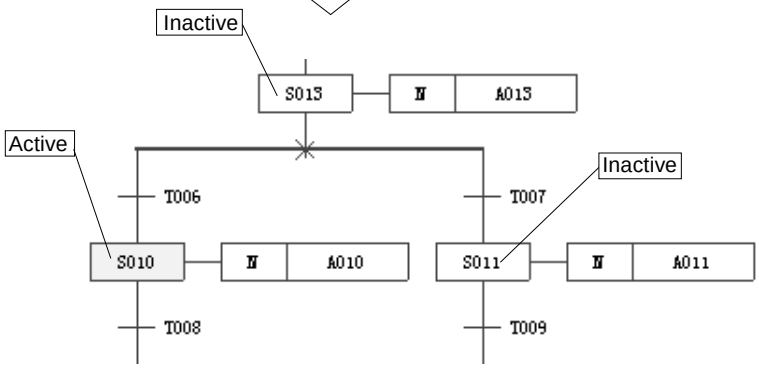


(2) Divergence of sequence selection

A single step has two or more step branches, one of which is selected for transition. A maximum of 32 branches are allowed.



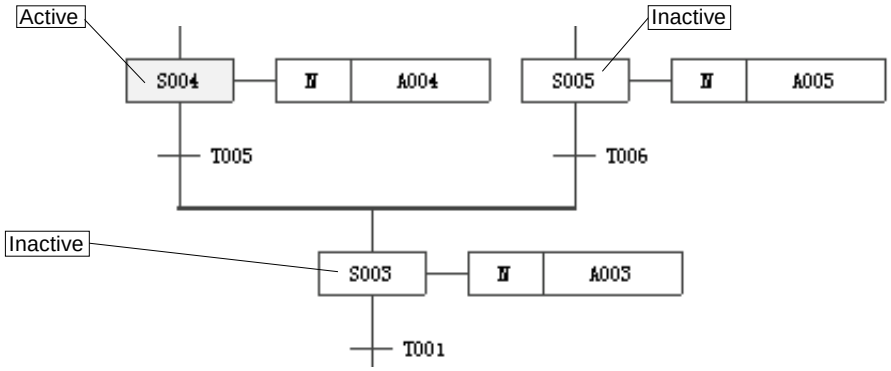
A branch to “S010” occurs when “S013” is active and “T006” is set to “1.”



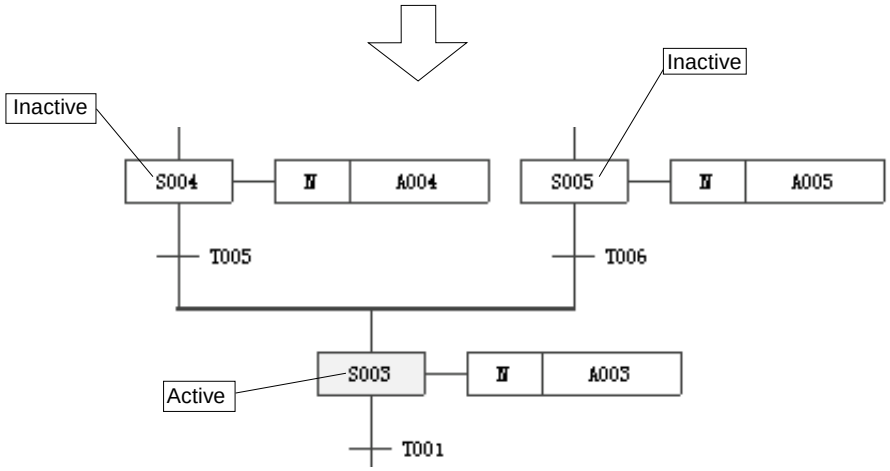
Note: The left-hand side step “S010” is selected when both “T006” and “T007” are set to 1 at the same time.

(3) Convergence of sequence selection

Two or more control flows which are implemented by the divergence of sequence selection converge into one.

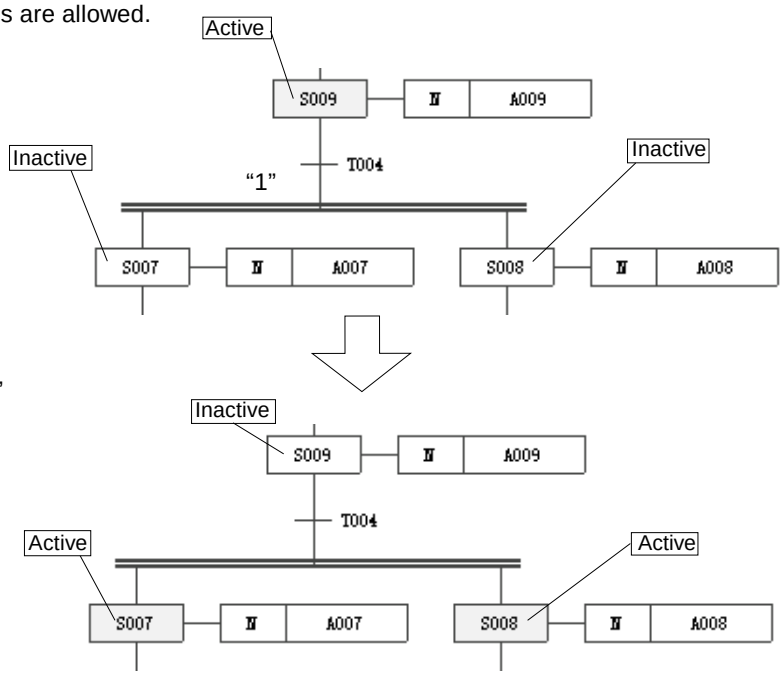


Control is transferred to “S003” when either “S004” or “S005” is active and the subsequent transition is set to “1.”



(4) Simultaneous sequences-divergence

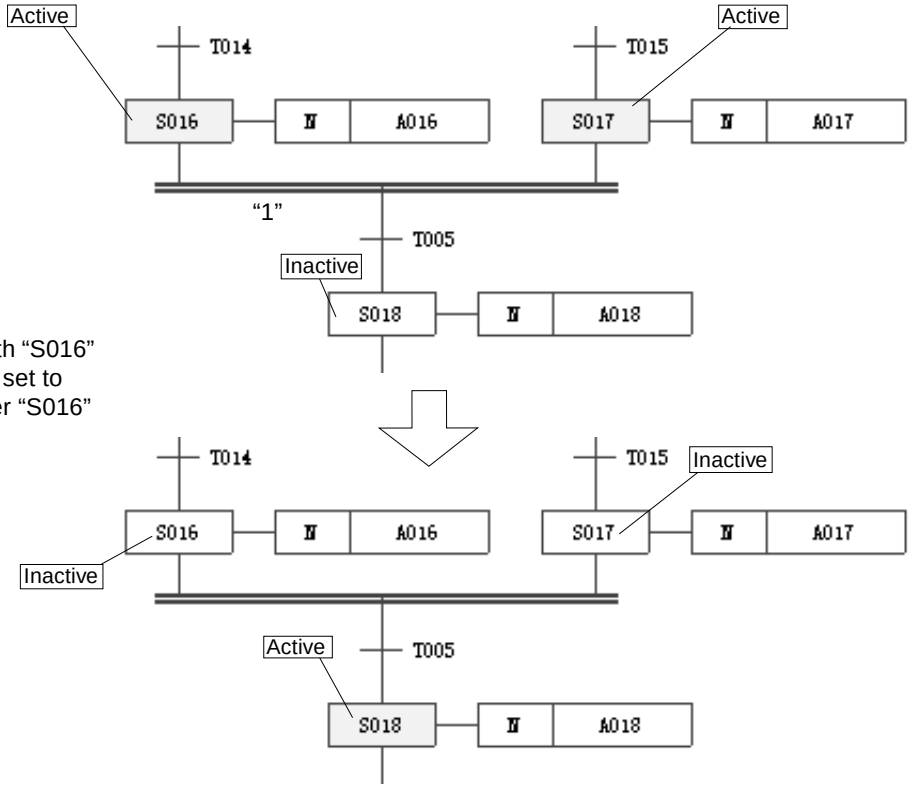
Control is transferred to two or more steps from one step at the same time. A maximum of 32 branches are allowed.



Control is transferred to "S007" and "S008" simultaneously when "S009" is active and "T004" is set to "1."

(5) Simultaneous sequences-convergence

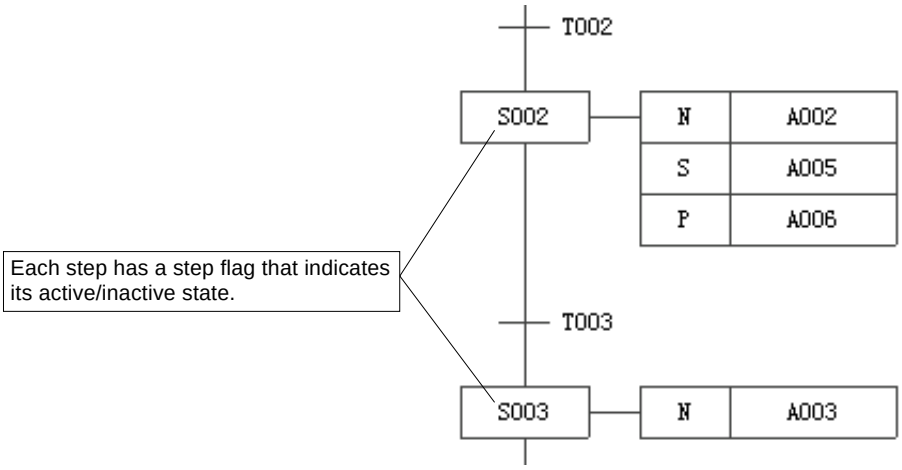
Two or more control flows which are implemented by the simultaneous sequences-divergence converge into one.



Control transfers to "S018" when both "S016" and "S017" are active and "T005" is set to "1." No transition occurs when either "S016" or "S017" is inactive.

2-6-3 Automatically generated SFC variables

Step flags and action flags variables are automatically generated in SFC. There is no need for the user to declare step flag and action fiag variables.



<Variable representation>

Period

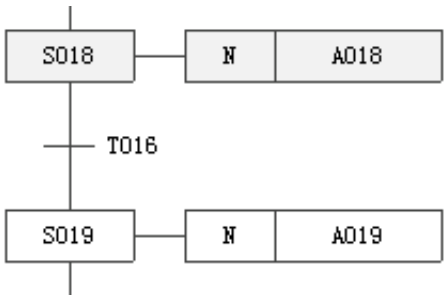
Step flag: step name.x (data type: BOOL; “1” when active, “0” when inactive)

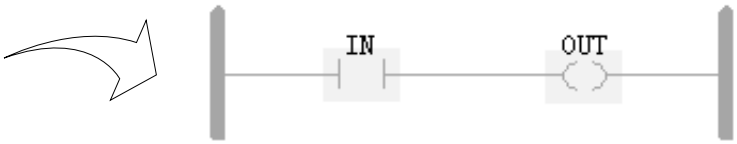
Action flag: Action name.x (data type: BOOL; “1” when active, “0” when inactive)

2-6-4 SFC programming precautions

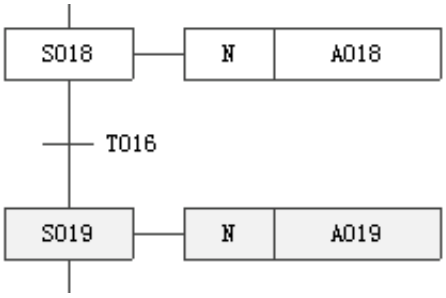
<Step reset processing>

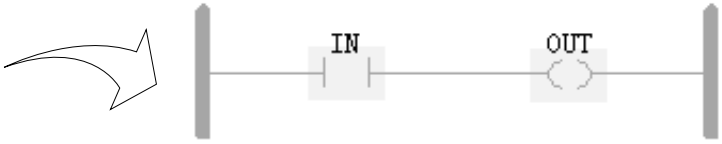
The system does not reset the action program that is connected to a step when its state is switched from the active to inactive state. If necessary, such action programs must be reset by the application.





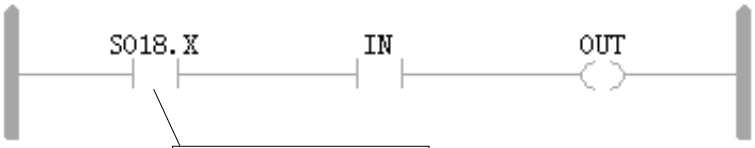
"OUT" remains on regardless of the state of "IN" when a step transition occurs with "OUT" being in the "ON" state.





<Example of resetting an action program>

In the above example, "OUT" remains on when the step active state switches from "S018" to "S019." To turn off "OUT" on state transition, create the following program in the action:



Add the S018 of step flag as an interlock.

<Programming a jump in an action/transition>

When programming a jump in an action or transition, make sure that the jump destination falls within the program.

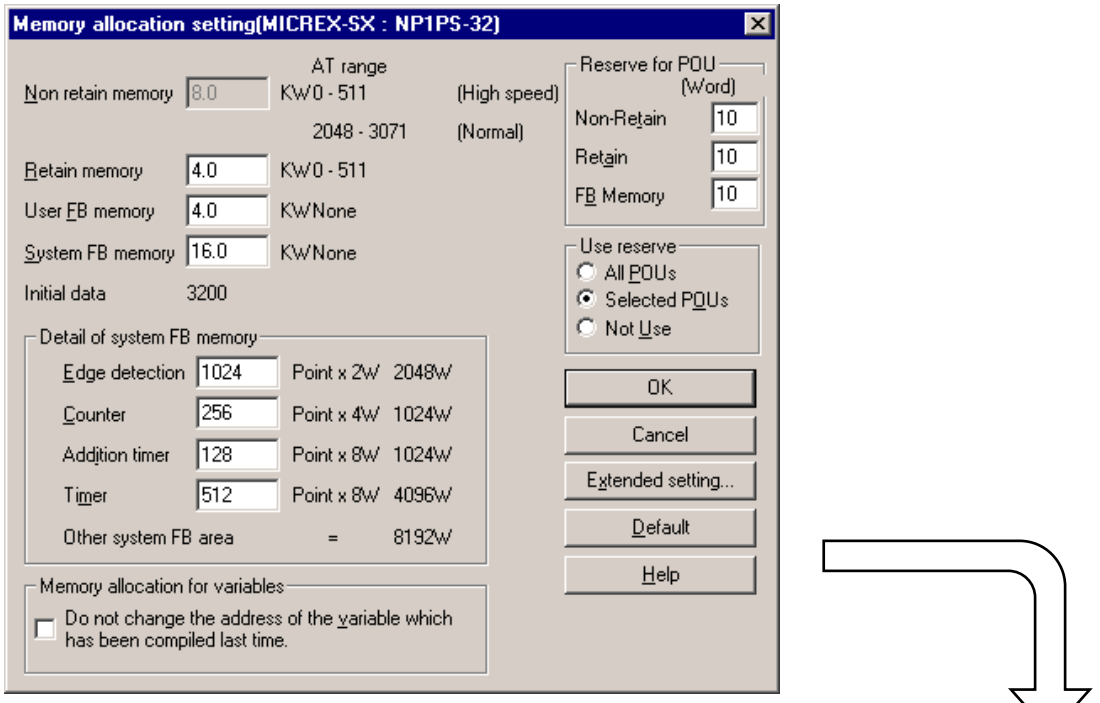
A jump specified in a program must not go out of the program.

2-6-5 Continuous operation of SFC

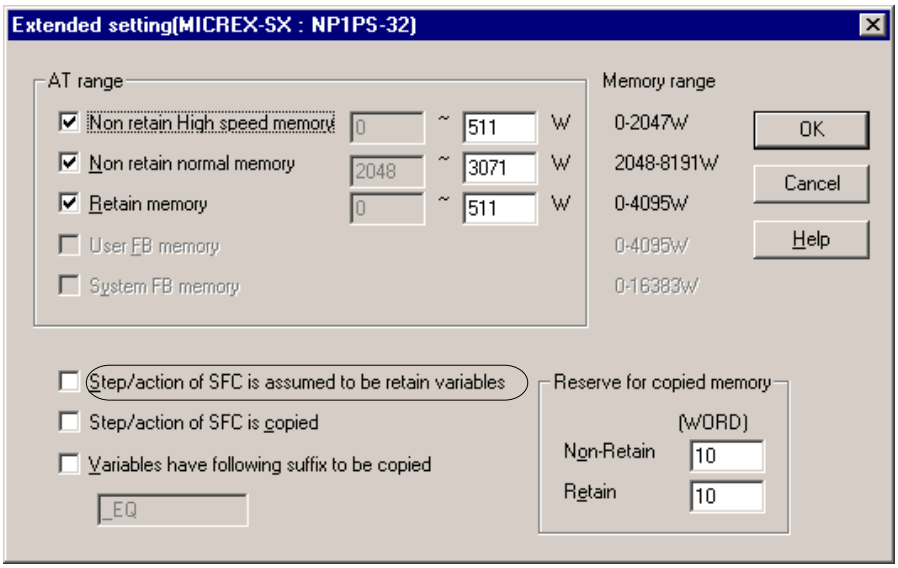
When compilation is performed using SFC step or action flag as retain variable with D300Win (V2.2 or newer), SFC step flag or action flag is assigned to the retain memory.
This operation enables SFC program to continuously run at warm start of the CPU.

<Operation of D300Win>

1) When the [Extended setting(A)...] button in the [Memory allocation setting] dialog is left-clicked, the [Extended setting] dialog is displayed.



[Extended setting] dialog



2) For warm start, if you want to start from the step at which the system was turned off, check [Step/action of SFC is assumed to be retain variable].

Section 3 System Definitions

Page

Section 3 System Definitions
3-1 System Definition Summary

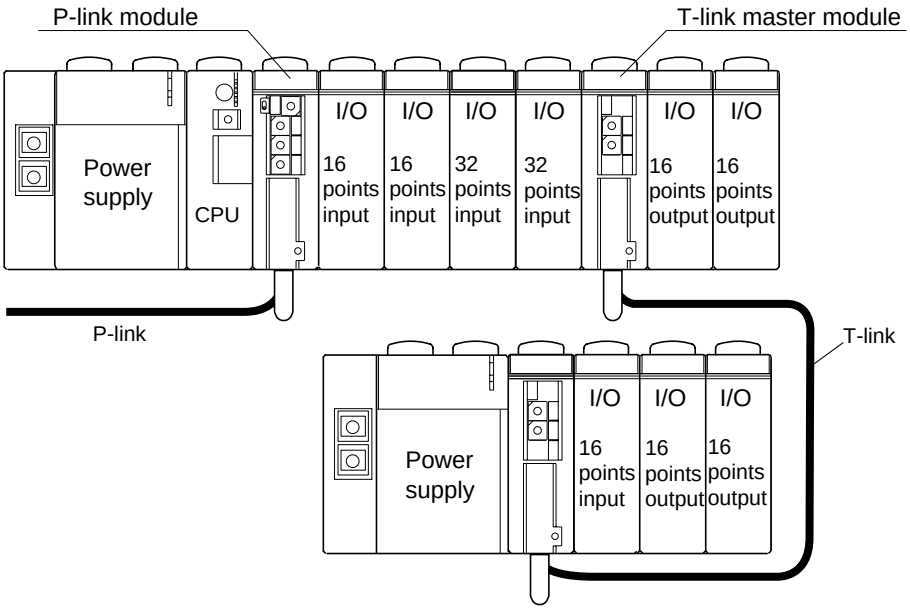
Name	Definition	Reference	Recognition Timing	
			High-performance CPU	Standard CPU
System configuration definition	Module registration and SX bus station number in system configuration	3-2	Configuration reset Download	Configuration reset
System operation definition	SX bus Takt period, configuration check waiting time, initialization method	3-3-1	Configuration reset	Configuration reset
Duplex mode definition	Duplex mode enabled/disabled, 1 to 1 duplex mode, N to 1 duplex mode	3-3-2	Configuration reset	Not supported
System fail-soft enabling definition	The fail-soft start-up start SX bus station number	3-3-3	Configuration reset	Not supported
System output definition	System output module	3-4	Configuration reset	Configuration reset
CPU operation definition	Watchdog timer, battery-less operation, operation at power-on	3-5-1	Configuration reset	Configuration reset
CPU memory size definition	Data memory size, AT specification range, reserve memory	3-5-2	Configuration reset	Configuration reset Download
I/O group	I/O group registration	3-5-3	Configuration reset	Configuration reset
Fail-soft setting	Fail-soft enabled/disabled for modules (I/O module, etc) other than common modules	3-5-4	Download	Configuration reset
Input filtering time	Input filtering time for digital input modules (DC input devices)	3-6-1	Download	Configuration reset
Output hold definition	Hold/reset registration for output modules	3-6-2	Download	Configuration reset
T-link master module parameter	Individual output hold station definition	(Note 2)	Configuration reset	Configuration reset
OPCN-1 master module parameter	Response time definition	(Note 2)	Configuration reset	Configuration reset
DeviceNet master module parameter	Individual operation definition, Individual output hold station definition	(Note 2)	Configuration reset	Configuration reset
P/PE link parameter	Operation definition, configuration registration definition, area setting, bank-switched CPU operation definition	(Note 2)	Configuration reset	Configuration reset
FL-net parameter	Operation definition, configuration registration definition, area setting, bank-switched CPU definition	(Note 2)	Configuration reset	Not supported

Note: 1) Reset includes all resets (configuration resets), individual reset (resource reset), and power-on.
2) Refer to the user's manual for each individual module.

Manual Name	Manual No.
MICREX_SX Series SPH User's Manual for the T-link Master Module	FEH204
MICREX_SX Series SPH User's Manual for the OPCN-1 Master Module	FEH238
MICREX_SX Series SPH User's Manual for the DeviceNet Master Module	FEH232
MICREX_SX Series SPH User's Manual for the P/PE-link Module	FEH203
MICREX_SX Series SPH User's Manual for the FL-net Master Module	FEH234

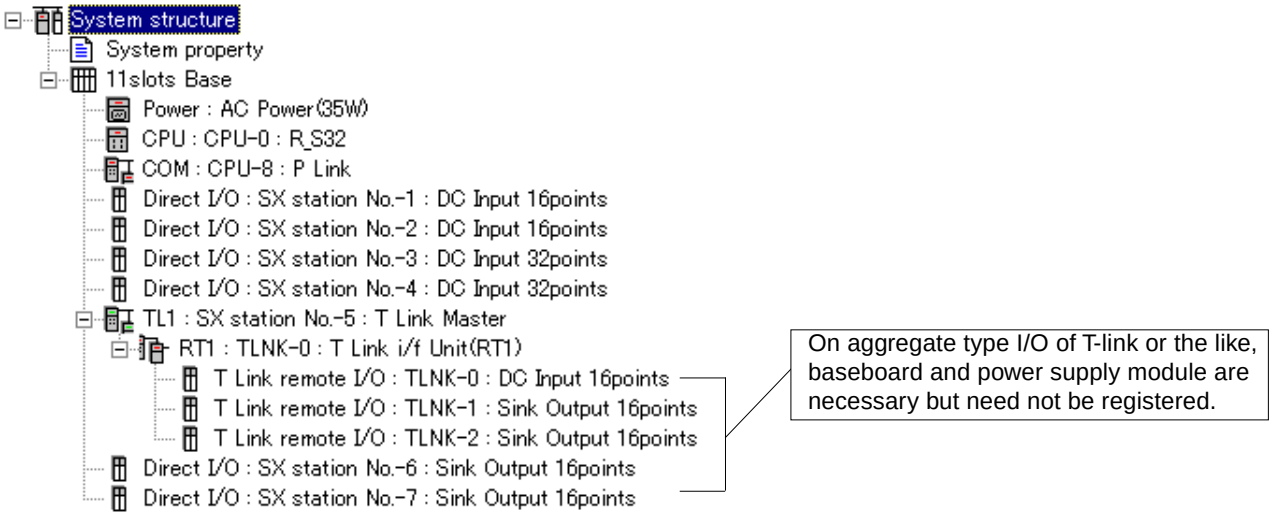
It is necessary to register the modules to be used to the MICREX-SX series system as system configuration definitions.

<System configuration example>



<System configuration definition tree window>

The system configuration definition tree for the above system configuration is as shown below.



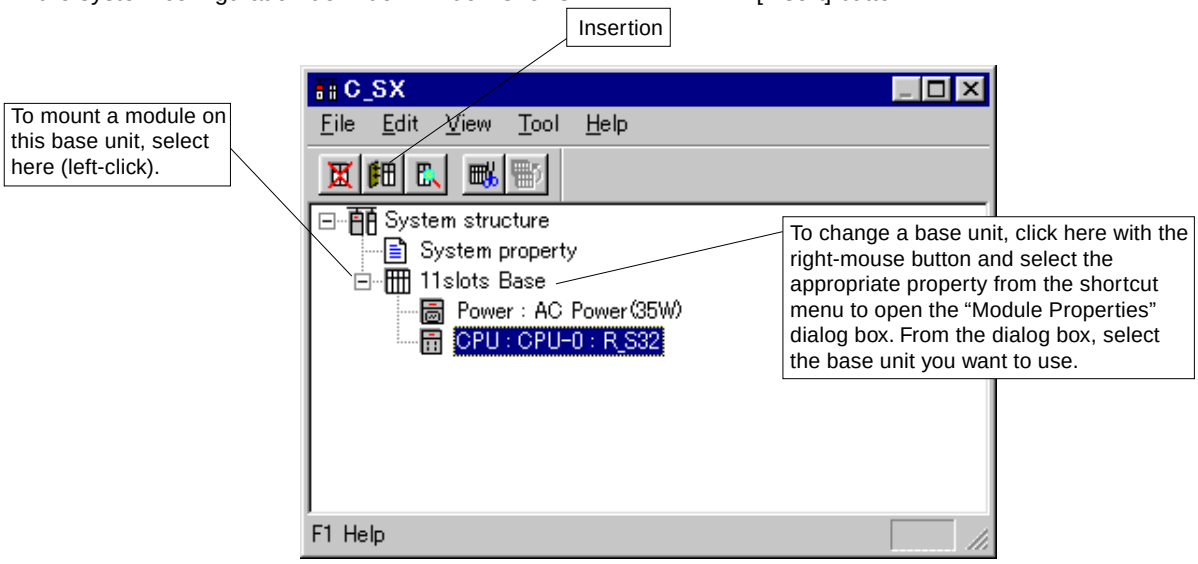
Note: In the multi-CPU system, use the same system configuration definition for every CPU module.

<Types of icons>

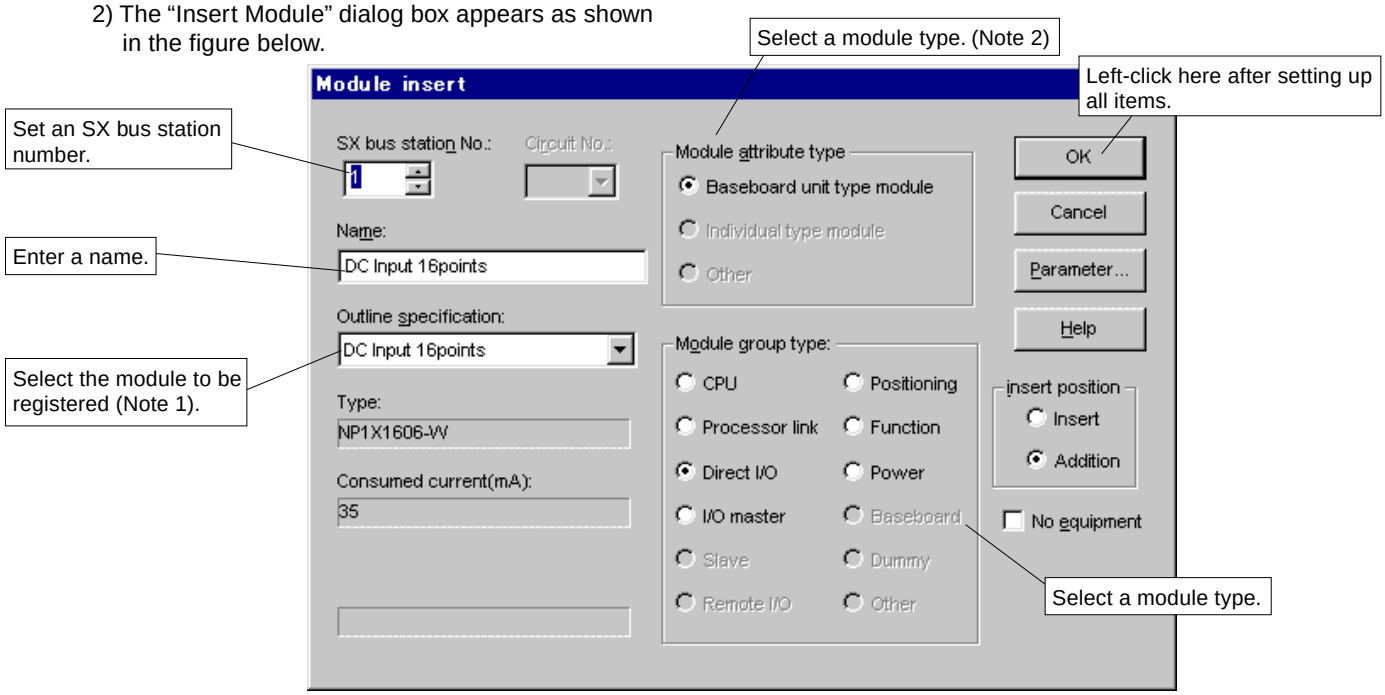
Icon	Description
	Defines the system properties such as SX bus Takt time.
	Denotes a base board.
	Denotes a power supply module.
	Denotes a CPU module.
	Denotes an input/output module.
Red	Denotes a processor link module.
Green	Denotes a remote I/O master module.
Red	Denotes a remote I/O interface module.
Black	Denotes a positioning control module.
Green	Denotes a function module.

<Setup procedure>

1) Double-click the [System_Definition] icon in the project tree with the left mouse button to open the system configuration definition window. Initially, the system configuration definition window shows a power supply module and a CPU module under an 11-slot base board as shown in the figure below. Select the CPU module and left-click the [Insert] button.

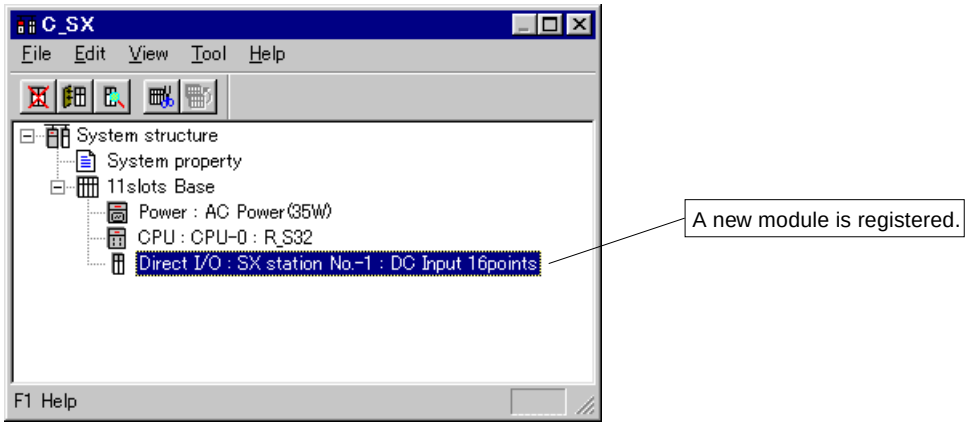


2) The “Insert Module” dialog box appears as shown in the figure below.



Note: 1) To select a module, first select the type of the module to register.
2) To select the unit (an individual module) directly connected to the SX bus, click the “Insert” button using the left-mouse button with Base selected in the “System Configuration Registration” dialog box.

3) The module is registered to the system as shown below. Follow the same procedure to register subsequent modules.



<Defining an SX bus station number>

It is necessary to assign an SX bus station number to each module (except the power supply module) that is to be connected to the SX bus. To assign an SX bus station number, use the SX bus station number field in the “Insert Module” dialog box which is shown in the figure above. (Assign a CPU number when defining a CPU or P/PE link module.)

PIO Number	SX Bus Station Number
1	1
2	2
.	.
.	.
.	.
238	238

CPU Number (Note)	SX Bus Station Number
0	254
1	253
.	.
.	.
.	.
15	239

Note: Make sure that the CPU number set by the D300win matches the CPU number switch settings on the module body.

3-3-1 System operation definitions

(1) SX bus Takt time

The SX bus Takt time is defined as the period at which data is exchanged between the modules (such as input/output modules) that are connected to the SX bus. Takt time can be set in the range from 0.5ms to 10.0ms, in 0.5ms steps (when CPU firmware version is V50 or newer). For older CPU firmware versions, Takt time can be set to 0.5ms, 1.0ms, 2.0ms ..... (in 1ms steps from 1.0ms). Default is 1.0ms.

Note: 1) A 0.5ms Takt period may be executed in such conditions as having a single CPU in a powerful CPU, 256 or less directly-connected I/Os, and no remote I/O and communication module.
2) For the products whose CPU firmware version is V34 or older or V3A to V3Z, Takt time can be set to max. 20ms, in 1ms steps.
3) V2.2 or newer versions of D300win support the feature of setting Takt time in 0.5ms steps.

(2) Configuration check waiting time

At power-on, the CPU module initializes itself. When initialization has been done, a configuration check starts for the modules in one configuration. Set the time for configuration check for “configuration check wait time.”

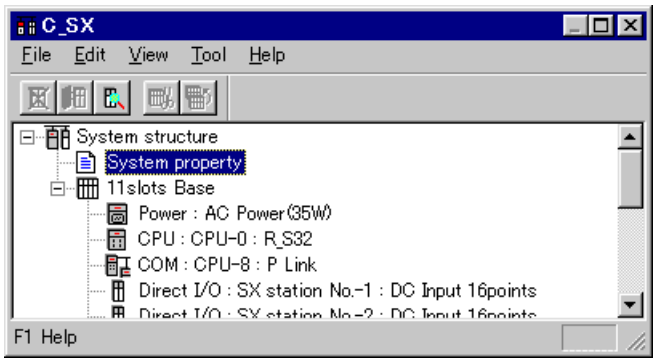
(3) Initialization method

The CPU module initializes itself when the system power is turned on. The user can specify whether the CPU is to perform diagnostics on the internal memory in the CPU.
The CPU initialization time is approximately 4.5s when memory diagnostics are performed and approximately 2.5s when memory diagnostics are disabled.

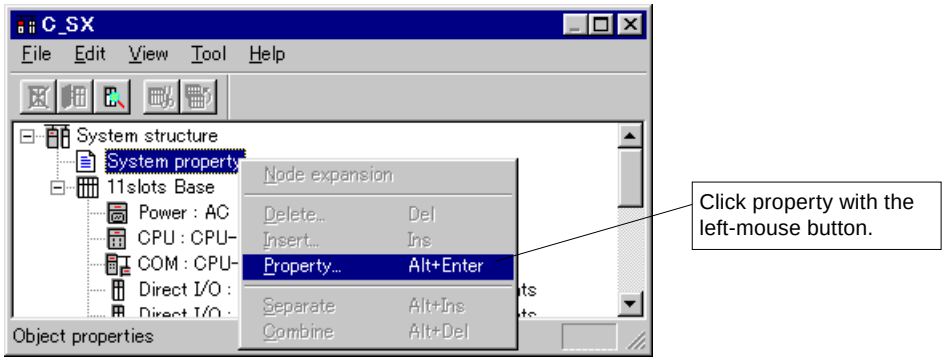
Note: 1) Memory diagnosis performs device read/write check.
Note: 2) The standard CPU does not support this item. (Device read/write check is not performed.)

<Setup procedure>

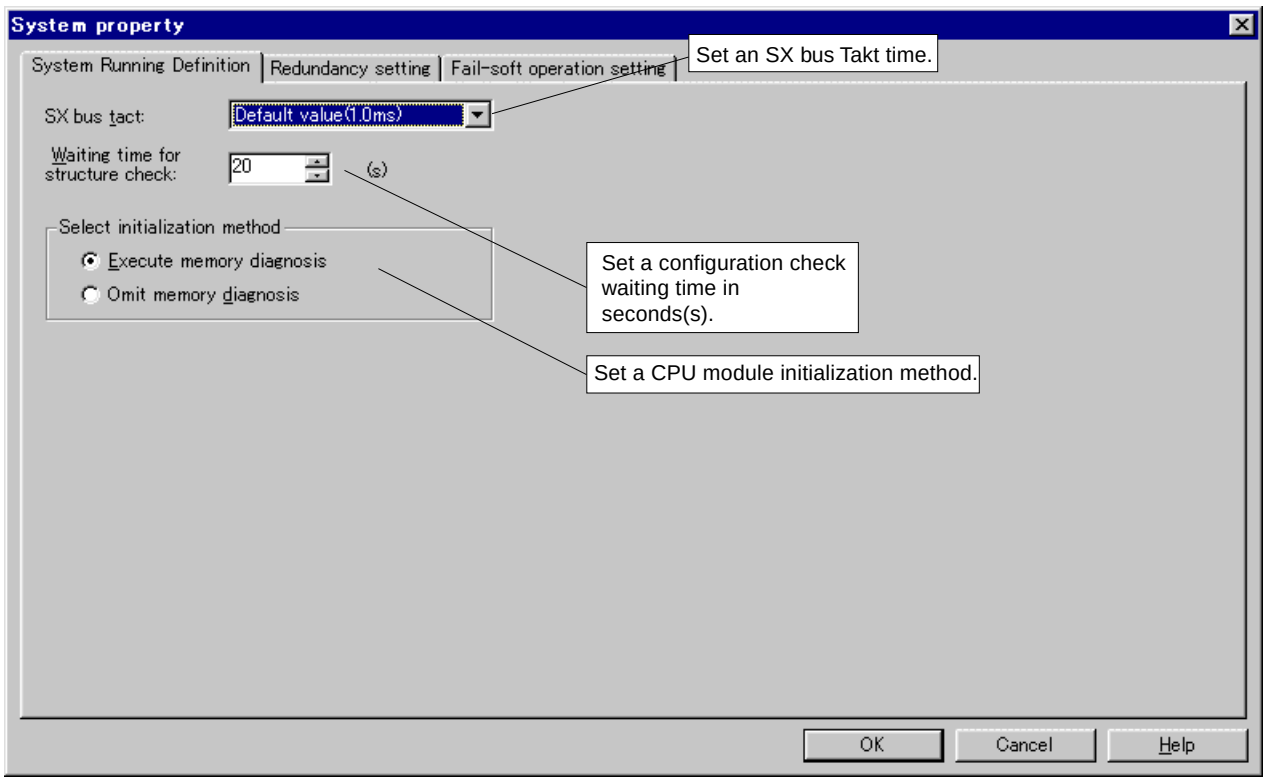
1) Double-click the [System_Definition] icon in the project tree with the left mouse button to open the system configuration definition window.



2) Click system property with the right-mouse button.



2) “System Properties” dialog box appears.



3) When filling the required fields, click the left-mouse button.

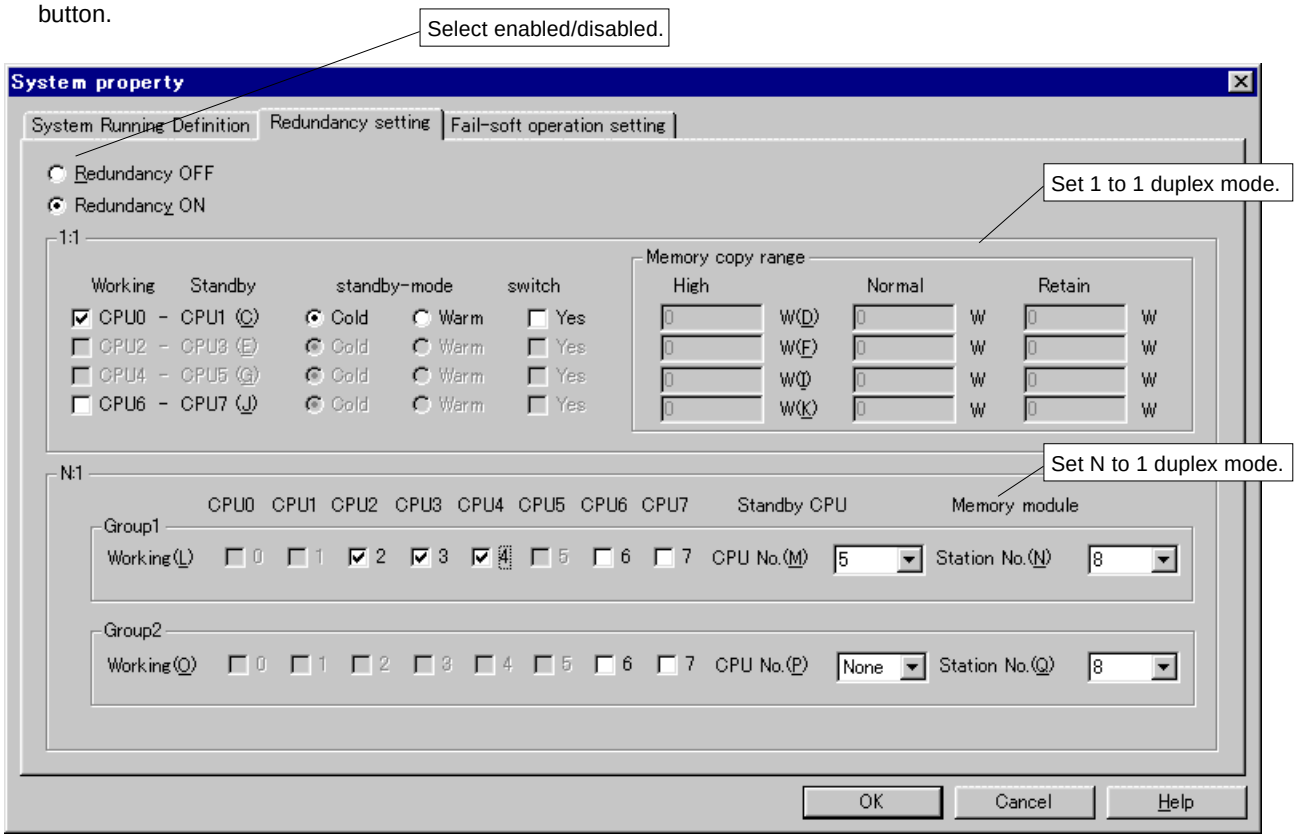
3-3-2 System duplex mode definition

(These functions are not supported in Standard CPU and SPS.)

The mode, in which dual CPUs are used for controlling the system to improve system safety and reliability, is called the CPU duplex system. (Refer to “Chapter 4 Duplex System” for information on the principle of duplex mode operation.)

<Setting the duplex mode>

1) Call up the “System Properties” dialog box and click the “System Duplex Mode Definition” tab with the left-mouse button.



2) When filling out the required fields, click the “OK” button.

<Fields for setting 1 to 1 duplex mode>

1) Setting a pair of CPUs ..... Set a pair of CPUs in the duplex mode.
2) Setting the stand-by method ..... Set to specify whether the internal data of the operating CPU is to be inherited by the waiting CPU at start-up (warm/cold) when changeover occurs between CPUs.
3) Equalization range specification.....Set the range of data to be inherited in case of warm stand-by.
4) Setting relay switching ..... Set a relay-switched pair in the system with more than one 1 to 1 duplex pairs. Check the boxes for all the pairs to be relay-switched.

<Fields for setting N to 1 duplex mode>

1) Setting the operating CPUs ..... Set the operating CPUs for each group. The number of operating CPUs is any of a range 2-7.
2) Setting the waiting CPU ..... Set only one waiting CPU for each group. The CPU already registered as an operating CPU cannot be specified.
3) Setting the memory module(s) ..... The N to 1 duplex system requires at least one memory module for storing programs of N CPUs. Set the SX station number for the memory card interface module to be used.

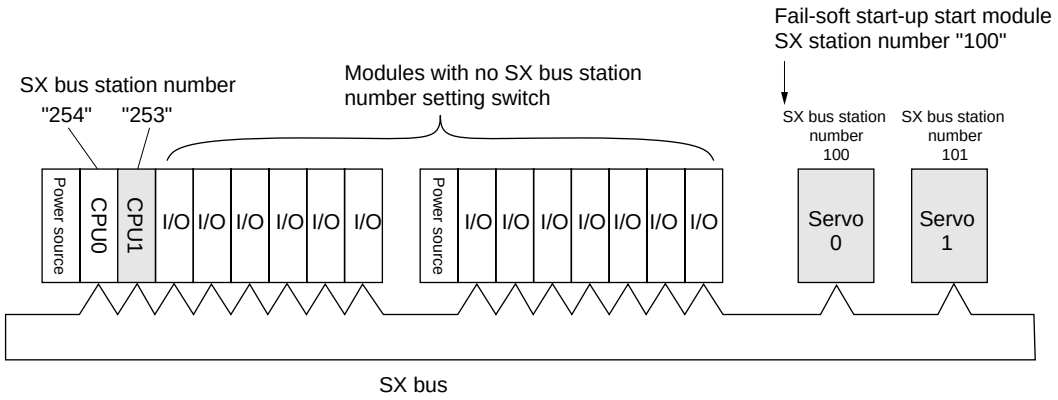
Note: To use the N to 1 duplex mode, the ** 30 or later version of memory card interface module (30 or later version of firmware) must have been installed. For the powerful CPU modules, the 1030 version or later must have been installed.

3-3-3 System fail-soft start-up

(1) Fail-soft disabled
If all the modules registered in the system configuration definition do not start within the system configuration check waiting time (a default: 20 sec.), a system error occurs (the CPU ALM goes on).

(2) Partial fail-soft start-up of modules with SX bus station numbers
At MICREX-SX system start-up, if some modules on which power is not turned on are detected, the system is re-started excluding these modules after the system configuration check waiting time has passed.
The system runs with a non-fatal fault (the CPU module RUN is on and the ALM is on).

<Example of system configuration at fail-soft start-up>



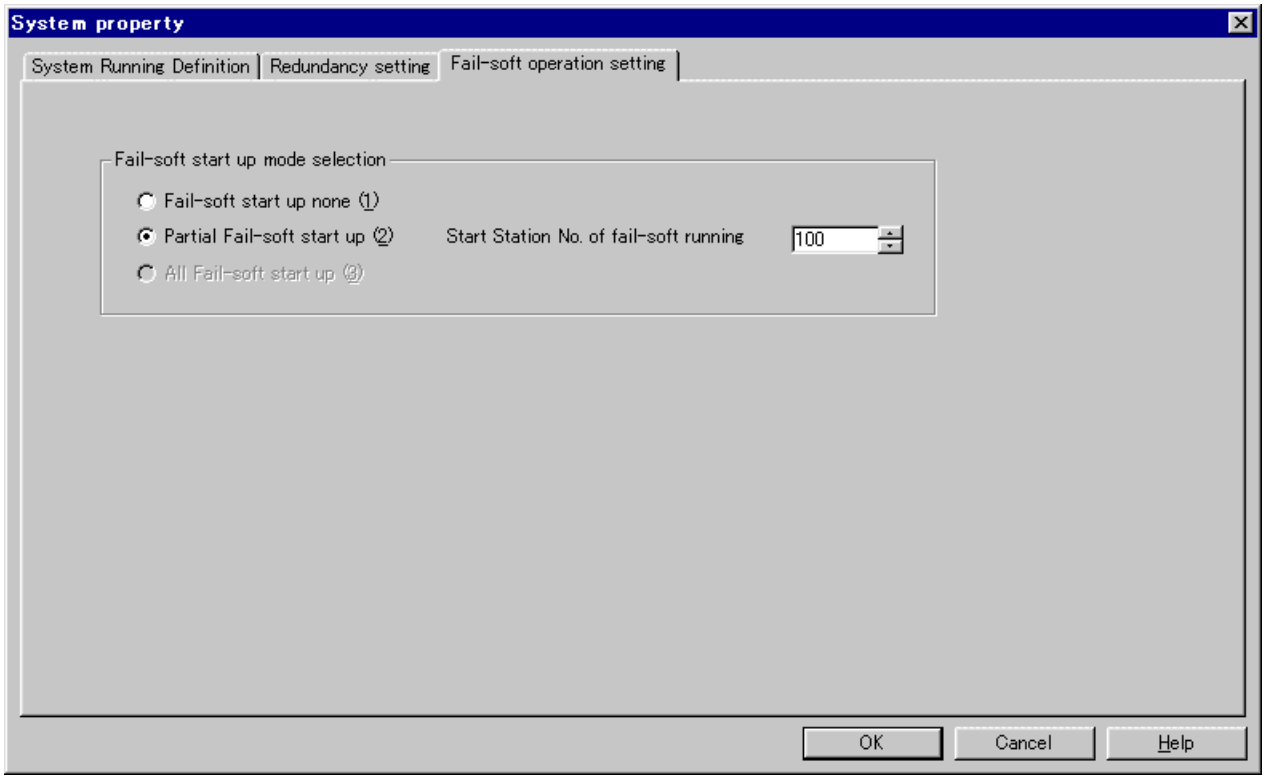
In the above example, the modules to be partially fail-soft started are "servo 0," "servo 1," and "CPU 1" in ☐.

1) The fail-soft start-up start module is any of a range from the SX bus station number, which has been set as the first one to be fail-soft started to 253. Note 1) Set the numbers excluding the start number for the SX bus station numbers for the module to be fail-soft started.
2) Be sure to assign the SX station numbers after one is declared as the start number only to the modules with an SX bus station number setting switch. (You do not need to assign consecutive numbers to them.) Note 2)
3) When the module to be fail-soft started is turned on after fail-soft start-up (in the non-fatal fault state), the module is activated and the system runs normally (The RUN is on and the ALM is off).

Note: 1) The CPU module, PE-link module, and P-link module are fail-soft started as well. Note that CPU0 is excluded.
2) If any module with no SX bus station number setting switch is detected, a system fatal fault (at initial start-up) or non-fatal fault occurs (at successive start-up).
3) Be sure to use the system fail-soft start-up mode only in the fail-soft compatible modules (version 20** or later). If any module incompatible with fail-soft is detected at initial start-up or at successive start-up, a system fatal fault occurs.

<Setting system fail-soft start-up>

1) Call up the “System Properties” dialog box and click the “System Fail-soft Setup Definition” tab with the left-mouse button.

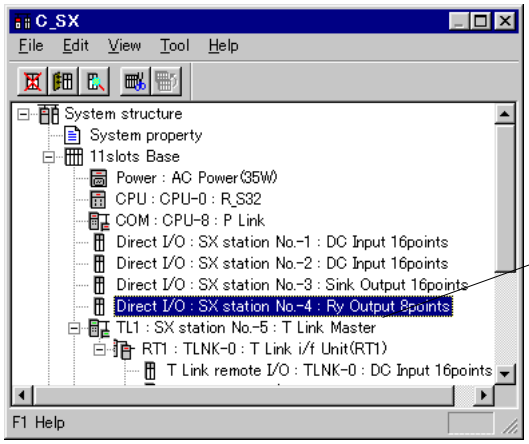


2) Select Partial fail-soft start-up enabled and enter the fail-soft start number.
3) When filling the field out, click the “OK” button with the left-mouse button.

The user can define a bit, for each configuration, that indicates the system operation state regardless of the application. This bit is set to ON when the entire system is running normally and set to OFF if the system has an error. The bit which can be set is bit 0 of the output module.

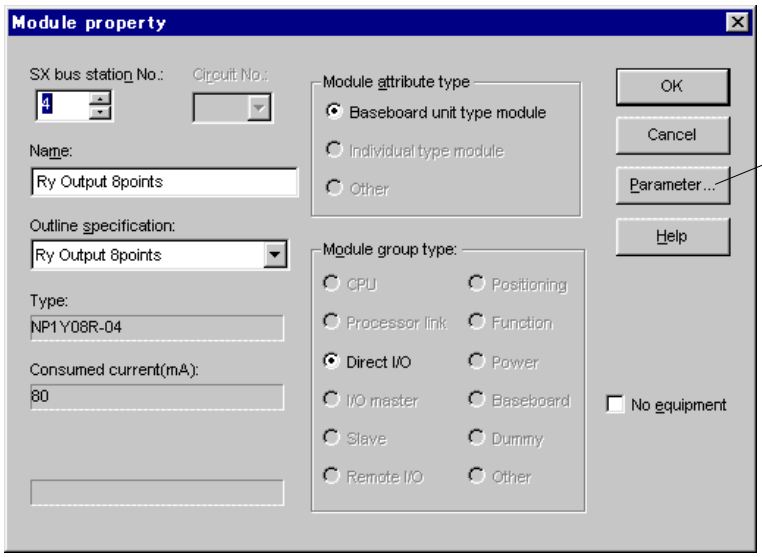
<Setup procedure>

1) Select the digital output module for which system output is to be set.



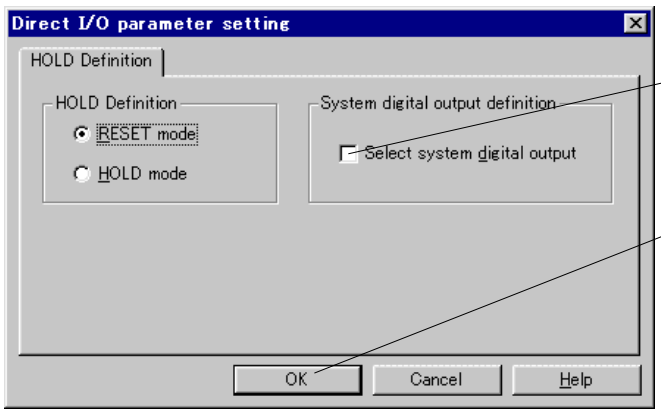
Left-click the [Property] button, or alternatively right-click to display the shortcut menu and left-click the [Property...] command in the menu.

2) The “Module Property” dialog box appears.



Click here with the left mouse button.

3) The “Direct I/O Parameter Setting” dialog box appears.



Place a check mark here to turn it on.

Click here with the left mouse button after setting up the necessary items.

Note: 1) Bit 0 of the module to be specified as system output cannot be registered in the I/O group definition.
2) A module to be specified as system output cannot be registered in the I/O group definition for any CPU other than CPU0.
3) Neither HOLD mode nor fail-soft can be set for a module that is specified as system output.

3-5-1 CPU operation definitions

The CPU operation definitions include the watchdog timer, power-on time operation, and battery less operation.

(1) Watchdog timer

The user can set a watchdog timer preset value from 1 ms to 4,095 ms. The default value is 4,095 ms.

(2) Power-on time operation

This parameter specifies the operation that the CPU module is to perform when system power is turned on when the key switch on the CPU module front panel is set to RUN or TERM. The table given below shows the relationship between the CPU module key switch positions and the CPU module operations. A default value is “RUN = Running/ TERM = Running.”

<CPU operations and key switch positions>

System Definition Setting	Operation	
	RUN	TERM
RUN=run/TERM=run	run	run
RUN=run/TERM=preceding state	run	preceding state (Note)
Run=stop/TERM=stop	stop	stop

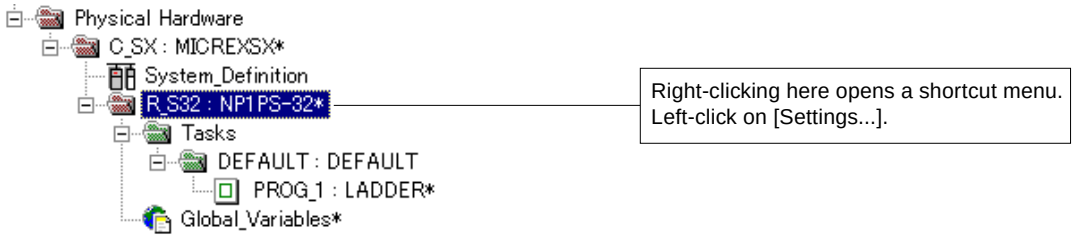
Note: The preceding state is the state of the CPU established before system power is shut off. The state is set to run if the CPU was running and to stop if it was stopped.

(3) Battery less operation

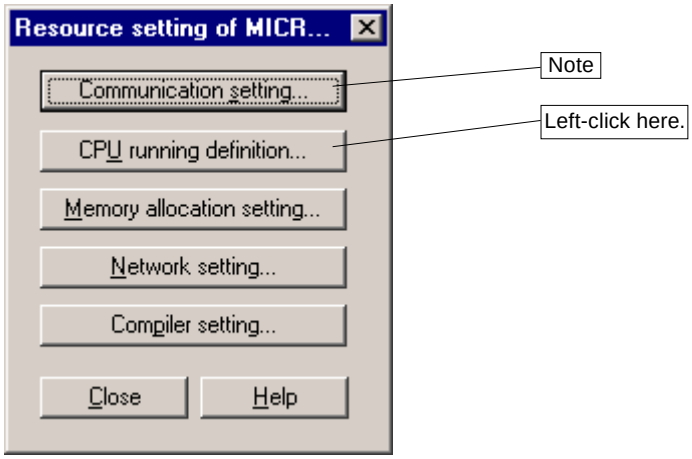
In case of battery-less running, the CPU module, when activated, initializes its memory to start operation. No data backup error is detected. The default value is battery-less running disabled. For the standard CPUs, battery-less running is not applicable if they have no user ROM card (MP8PMF-16) inserted.

<Setup procedure>

1) Select the target resource icon under the project tree.

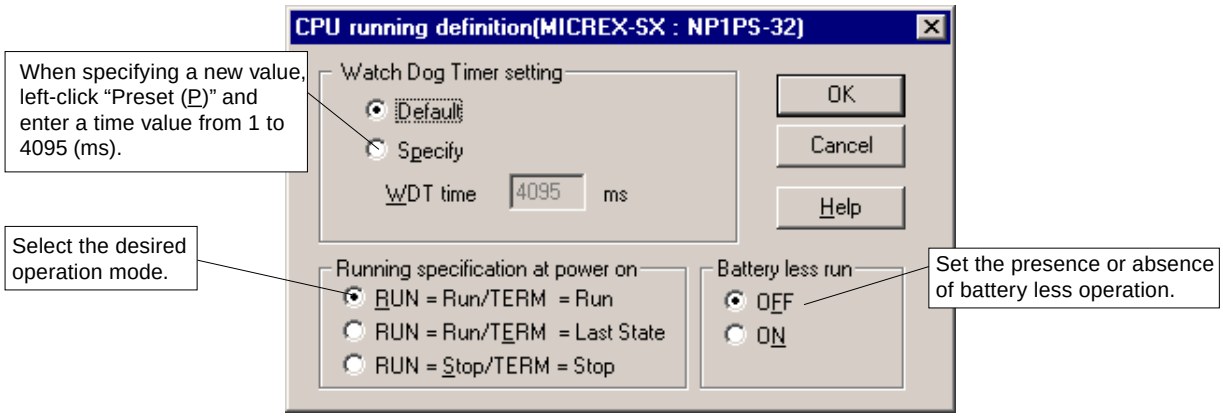


2) The “MICREX-SX Resource Setup” dialog box appears.



Note: The communications settings are used to connect the CPU module and the D300win. Refer to “User’s Manual D300win <Reference Book> (FEH251) / D300winV2 <Reference Book> (FEH254)” for information on communication setup.

3) The “CPU Operation Definition” dialog box appears.



4) After setting up the necessary items, click the [OK] button with the left mouse button.

3-5-2 CPU memory size definition

The user can set the size of the data memory in the CPU module. Although the size of the user memory area is 32K words and each area is defined by default size, the user can change the size of each area as required. The size of each area may be changed in 0.5K word increments.

<Allowable memory area size ranges>

The size of the user memory area may be changed in the range shown below. Note that the total number of words for user memory is fixed. For example, to set 32K words for the standard memory of High-performance CPU (NP1PS-32), specify all 0s for other memory areas.

High-performance CPU (NP1PS-32)

Memory Type		Minimum Value	Initial Value	Maximum Value
Standard memory (non-retained memory)	(%MW1)	2K words	8K words	32K words
Retain memory	(%MW3)	0K words	4K words	30K words
User FB instance memory	(%MW5)	0K words	4K words	14K words
System FB instance memory	(%MW9)	0K words	16K words	30K words

High-performance CPU (NP1PS-74)

Memory Type		Minimum Value	Initial Value	Maximum Value
Standard memory (non-retained memory)	(%MW1)	2K words	32K words	128K words
Retain memory	(%MW3)	0K words	16K words	126K words
User FB instance memory	(%MW5)	0K words	16K words	113K words
System FB instance memory	(%MW9)	0K words	64K words	126K words

High-performance CPU (NP1PS-117R)

Memory Type		Minimum Value	Initial Value	Maximum Value
Standard memory (non-retained memory)	(%MW1)	2K words	128K words	256K words
Retain memory	(%MW3)	0K words	32K words	254K words
User FB instance memory	(%MW5)	0K words	32K words	113K words
System FB instance memory	(%MW9)	0K words	64K words	254K words

Standard CPU (NP1PH-16)

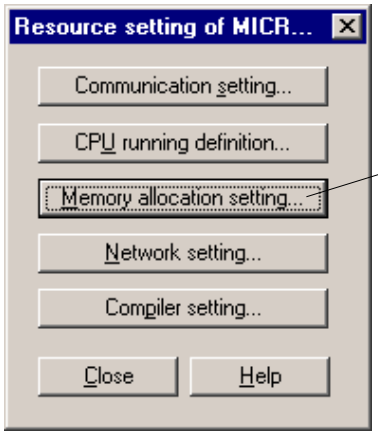
Memory Type		Minimum Value	Initial Value	Maximum Value
Standard memory (non-retained memory)	(%MW1)	0K words	8K words	31K words
Retain memory	(%MW3)	0K words	4K words	31K words
User FB instance memory	(%MW5)	0K words	4K words	14928 words
System FB instance memory	(%MW9)	0K words	8K words	31K words
Initial value setup area		0K words	7K words	31K words

Standard CPU (NP1PH-08)

Memory Type		Minimum Value	Initial Value	Maximum Value
Standard memory (non-retained memory)	(%MW1)	0K words	4K words	15K words
Retain memory	(%MW3)	0K words	2K words	15K words
User FB instance memory	(%MW5)	0K words	4K words	7216 words
System FB instance memory	(%MW9)	0K words	2K words	15K words
Initial value setup area		0K words	3K words	15K words

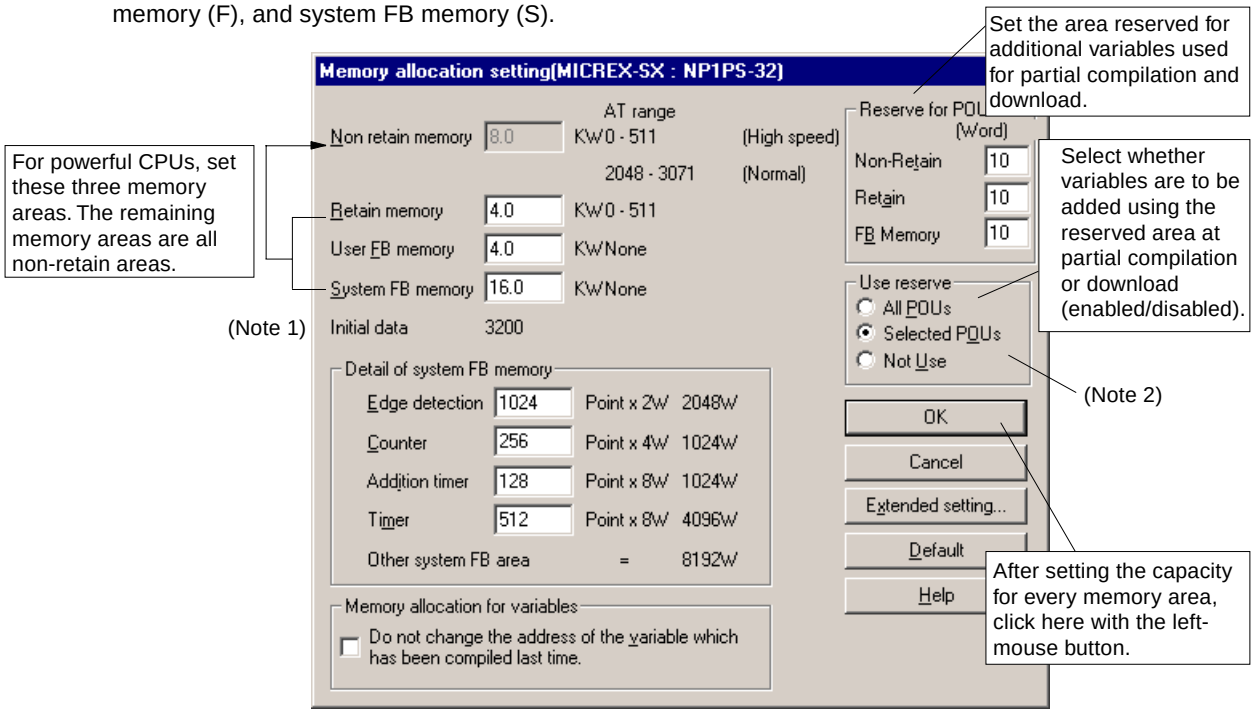
<Setup procedure>

1) Click the [CPU Memory Size Definition (M) ...] button in the “Select Resource” dialog box with the left mouse button.



2) The “CPU Memory Size Definition” dialog box appears.
Set the sizes of retained memory (R), user FB memory (F), and system FB memory (S).

The size of remaining free memory is assigned to non-retained memory (%MW). The memory size can be set in 0.5K word increments.



Note: 1) For standard CPUs, the number of initial values varies with settings for non-display, retain, user FB, and system memories. (The initial value area for the user FB has been automatically calculated.)

<Extended setting>
Extended setting for memory allocation includes “AT range setting,” “retain/equalization setting for automatically generated SFC variables,” “assignment of variable equalization by specifying suffix,” “setting equalization reserve size for each POU.”

(1) AT range
AT range is the memory area for the user to specify the memory allocation for variables. When an area is specified as AT range, no variable is automatically assigned to it.

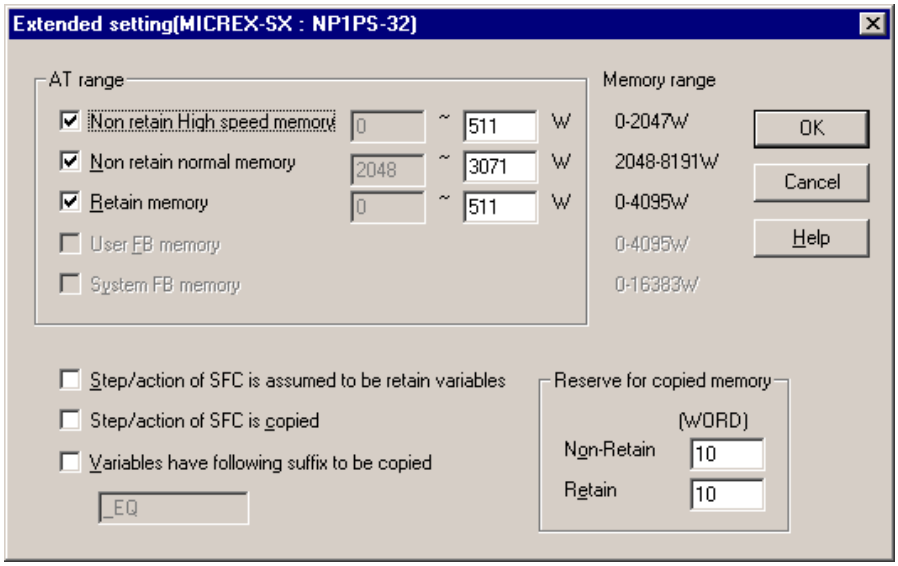
(2) Retain/equalization setting for automatically generated SFC variables
Step flag (step_name.X) and action flag (action_name.X) that are automatically generated by SFC are not declared as variable and therefore cannot be assigned to the equalization area of duplex system. However, when this box is checked, step flag and action flag can be made retain variables and assigned to equalization area of duples system.

(3) Assignment of variable equalization by specifying suffix
This function assigns specified suffix characters (3 to 8 single-byte characters) to equalization area.

(4) Equalization reserve size for each POU
In performing on-line change (POU change) on a duplex system, this area is used when SFC step flag or action flag is to be added or when a variable is to be added that corresponds to [Variables have following suffix to be copied].

<Setting procedure>

(1) When the [Extended setting(A)...] button in the [Memory allocation setting] dialog is left-clicked, the [Extended setting] dialog is displayed.



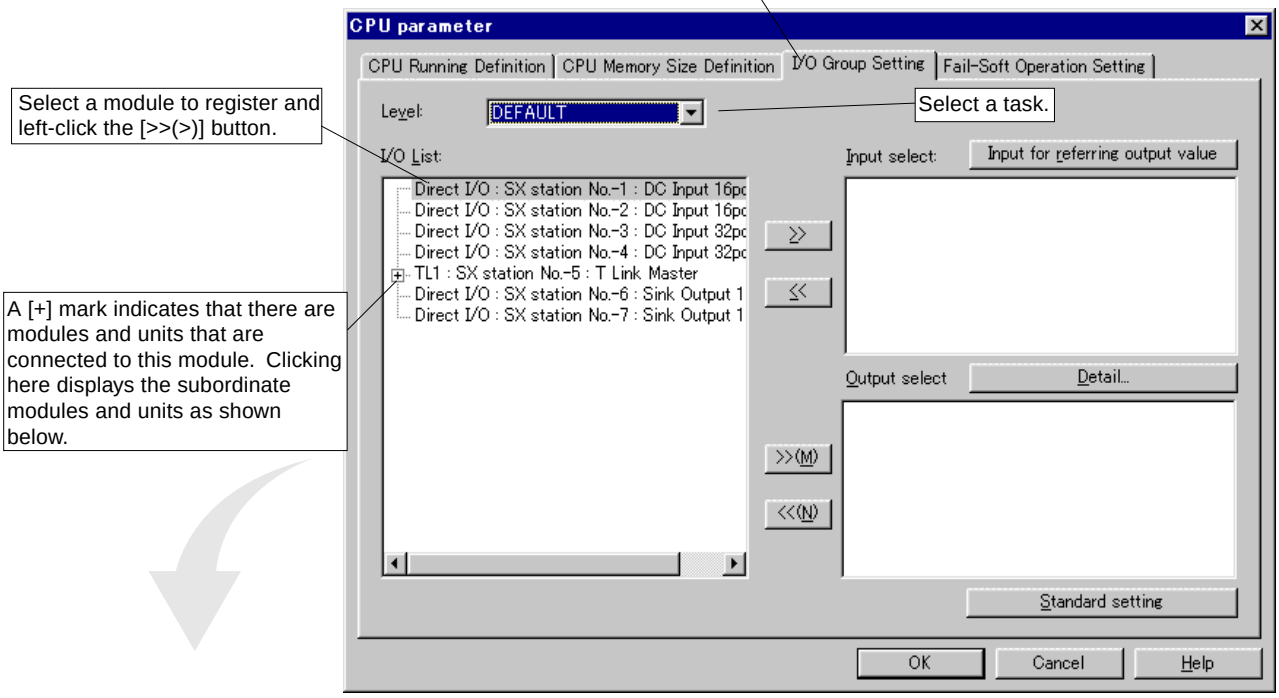
(2) When setting is completed, left-click the [OK] button, and you will return to the [Memory allocation setting] dialog.

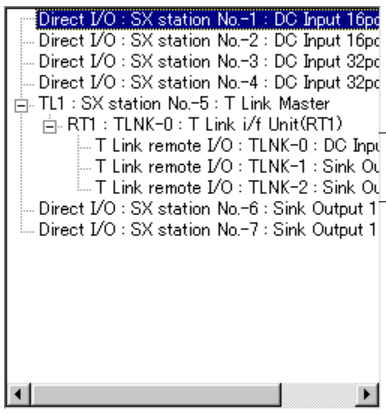
3-5-3 I/O group setup

The user can specify which I/O module in a configuration can be controlled by which task in which CPU module. Be sure to set a CPU I/O group for I/O modules. If any I/O module with no I/O group assigned is found, the CPU can detect no fault if it occurs.

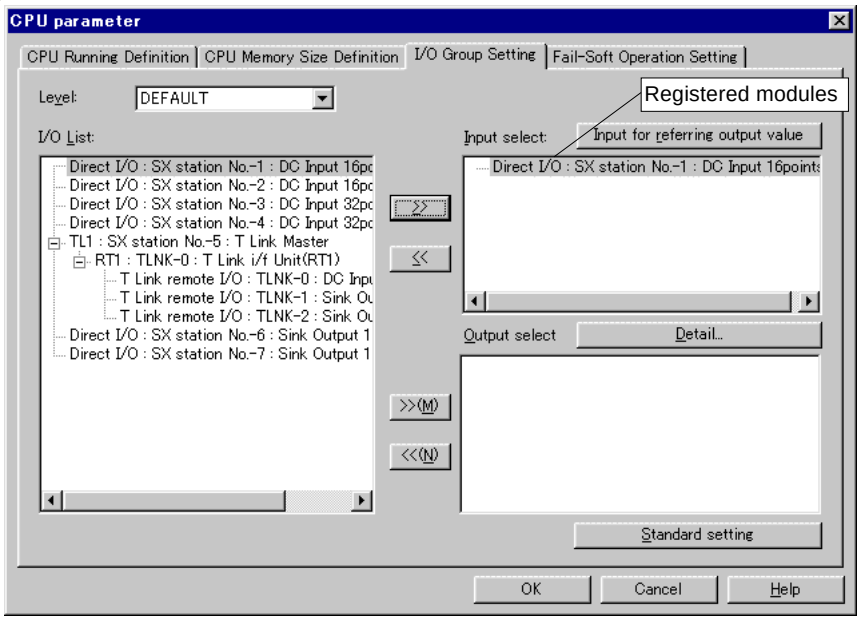
<Setup procedure (1)>

1) Click the [I/O Group Setting] tab in the "CPU Parameter" dialog box, and the following window appears.





You can register each I/O module on the NP1L-RT1 to an I/O group.

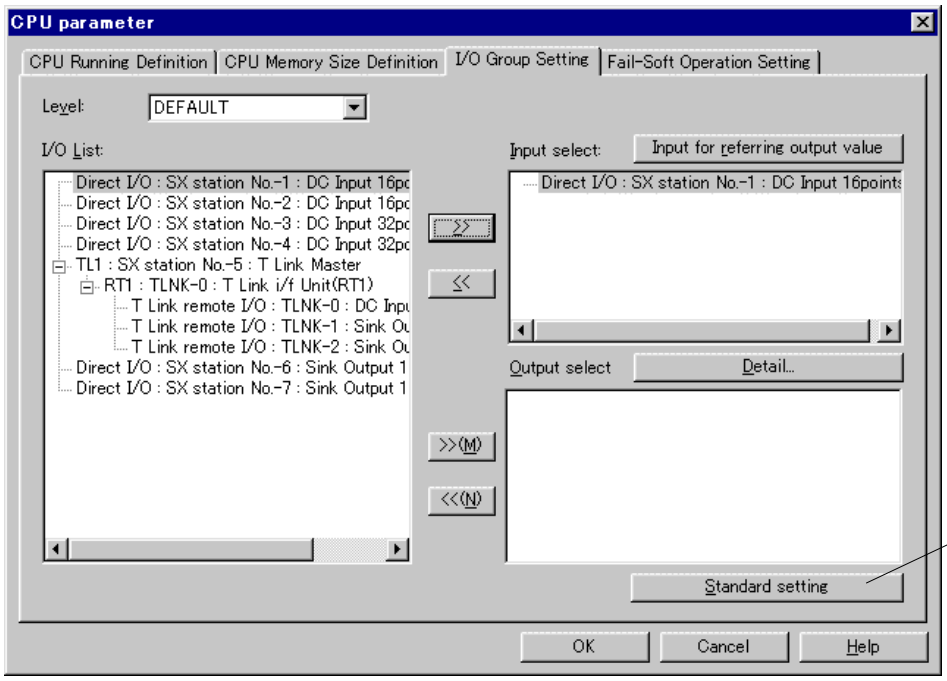


2) The modules are registered as shown in the upper right window. After setting the necessary items, click the [OK] button with the left mouse

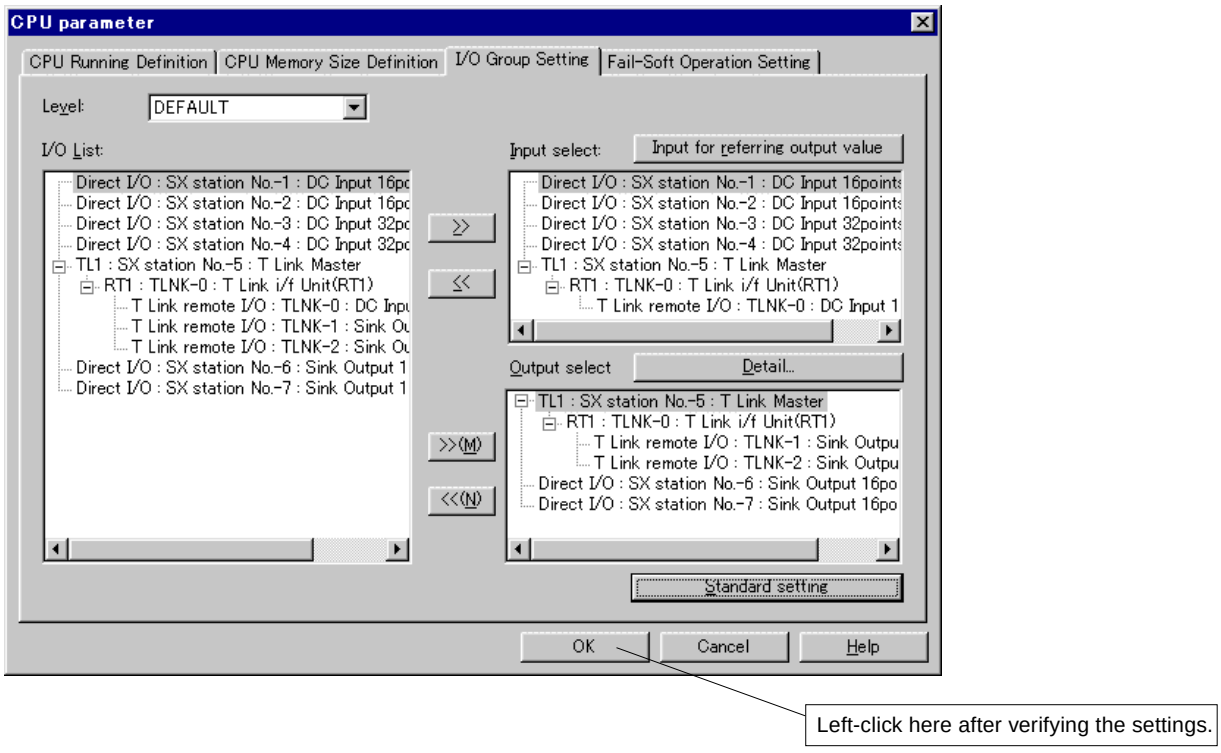
<Setup procedure (2)>

The user can use the [Standard Settings] button when registering modules to I/O groups only for the “DEFAULT” tasks in a single CPU system.

1) Make sure that the Level field is set to “DEFAULT” and click [Standard Setting] button with the left mouse button.



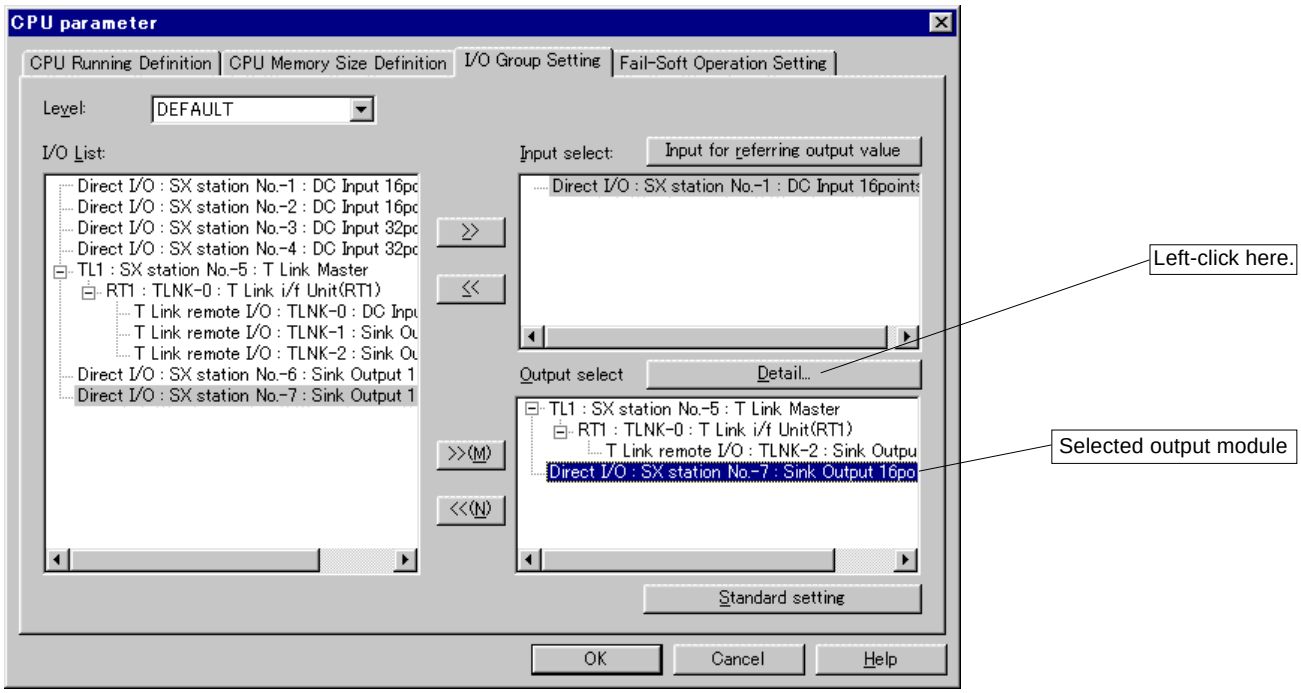
2) The input modules are automatically registered to the Input Selection (I) and the output modules to the Output Selection (O) as shown below.



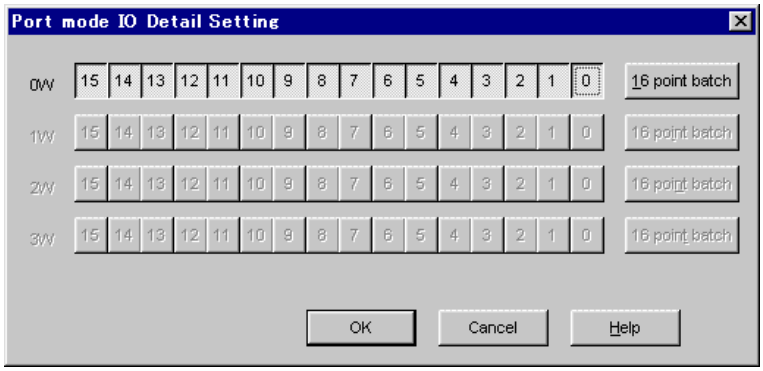
<Setup procedure (3)>

The user can register the output modules to I/O groups on a bit basis in a multi-CPU system.

1) Select the output modules that are to be registered on a bit basis from the Output Selection (O) window, and click [Detail...] with the left mouse button.

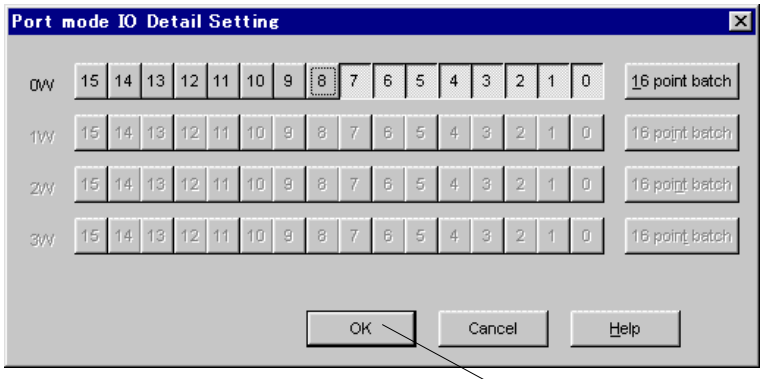


2) The system opens the window shown below. Initially, all bits are selected.



* A pressed button indicates that the corresponding bit is registered to the I/O group.

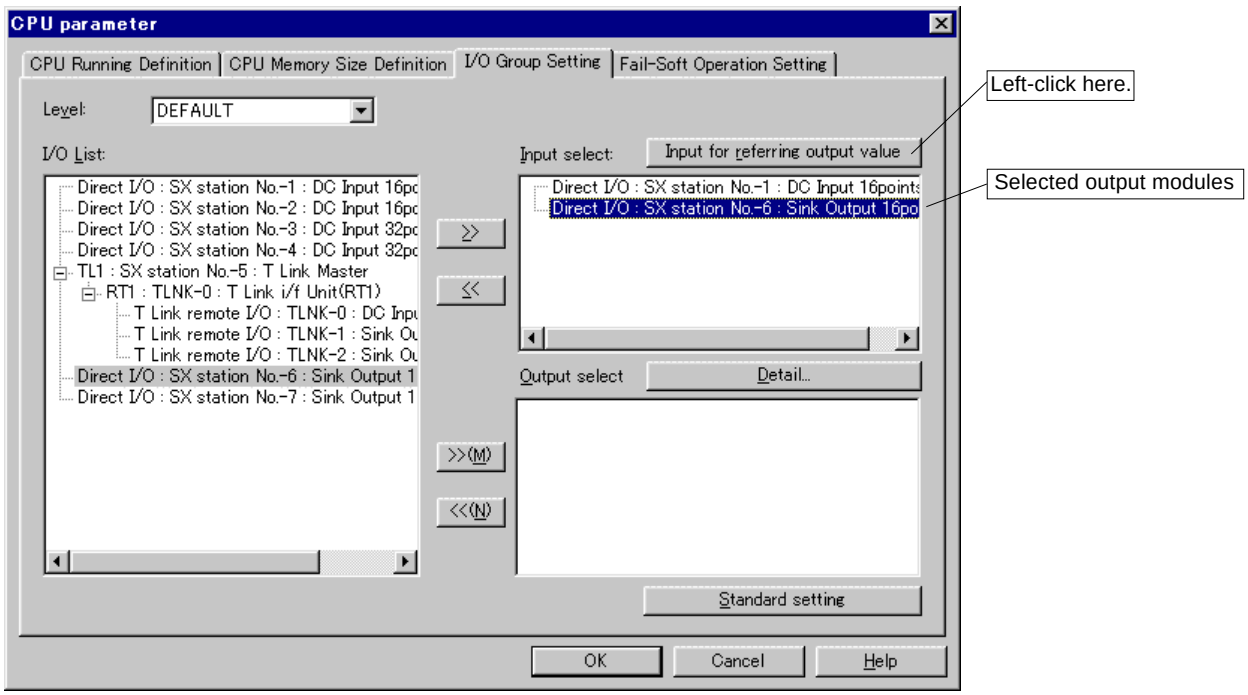
3) Click a bit that is not registered to the I/O group with the left mouse button to set it off. In the figure shown below, bits 8-15 are set off.



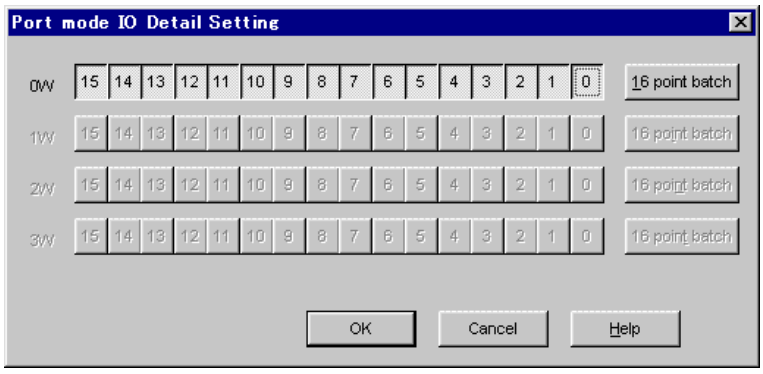
<Setup procedure (4)>

The user can register bits of an output module that is used by another CPU in a multi-CPU system to the input selection I/O group of the local CPU as “inputs” to an application program running on the local CPU.

1) Select an output module and click the [Input for referring output value] button with the left mouse button.

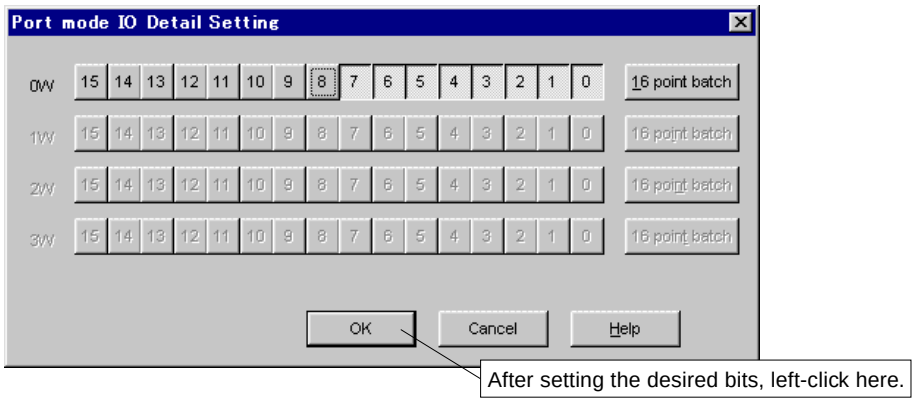


2) The system opens the window shown below. Initially, all bits are selected.



* A pressed button indicates that the corresponding bit is registered to the I/O group for output reference.

3) For a bit whose output is not to be referenced, click it with the left mouse button to set it off. In the figure shown below, bits 8-15 are set off.



3-5-4 Fail-soft running

Even if a fault occurs in the module with fail-soft enabled and goes down during system running, the entire system can continue running.

<Fail-soft operation of MICREX-SX system>

1) Module down

Even if the module with fail-soft enabled has a fault and goes down during system running, the CPU continues running with a non-fatal system fault (The RUN is On and the ALM is on). If the module with fail-soft disabled has a fault and goes down during system running, the CPU stops with a fatal system fault (RUN is off and ALM is on).

2) Module recovery

When the down module is recovered, the system returns to its normal state. Note that if more than one module has been down, no module can be activated unless all the down modules are recovered.

3) Modules with fail-soft enabled

Common modules with no I/O area	CPU, P/PE-link, and general communication modules	Unconditionally, fail-soft is applicable
Modules with I/O area	Digital I/O, analog I/O, and AS-I master modules	Register the modules with fail-soft enabled

Note: If you want to stop the system running when a fault occurs in the module to which fail-soft is unconditionally applicable, monitor any configuration fault information in system memory (%MW10.68 to %MW10.83) and set the user fatal fault flag (%MW10.14 to %MW10.16) to “1.”

4) I/O group and fail-soft registration

The module CPU does not control a module (having an I/O area assigned) not registered in an I/O group. Even if the module goes down, the CPU continues running normally.

Considerations in building the fail-soft system

The module and base unit versions, which support fail-soft, are listed below.
- Modules with internal firmware installed 1030
- modules with no internal firmware installed 10 (for example, a base unit)

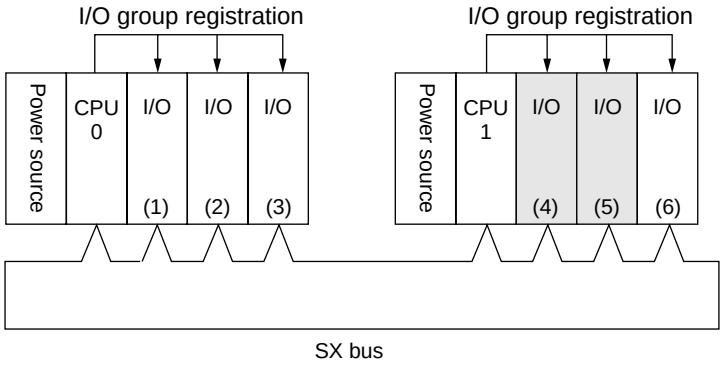
When any module, of a version earlier than that listed above, has been connected to the SX bus, no fail-soft can be implemented on it. The SX bus-connected module fail-soft mode flag (%MW10.0.13) is set to "0" in system memory.

Note: Even in a system in which a module incompatible with fail-soft has been configured, the remote I/O fail-soft feature can be used.

5) Fail-soft registration for the multi-CPU system

For the multi-CPU system, register the modules controlled by the self-CPU in the I/O group and also register them for fail-soft. The system operates as described below in case of a fault.

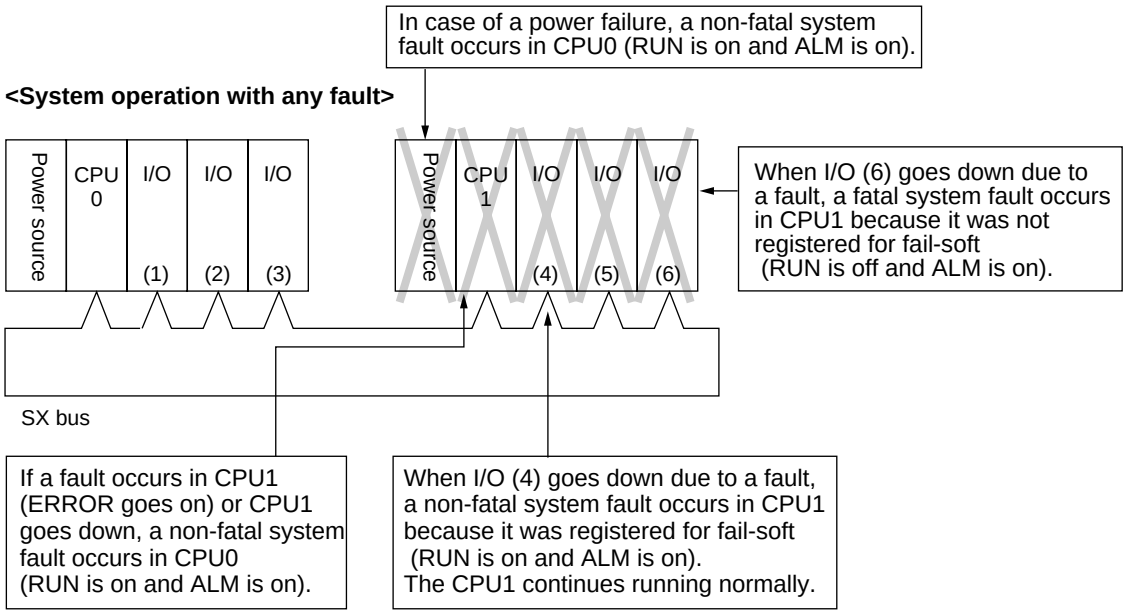
<Example of system configuration>



In CPU0, register I/O (1), (2), and (3) together as one I/O group and also register them for fail-soft.

In CPU1, register I/O (4) and (5) together as one I/O group and also register them for fail-soft.

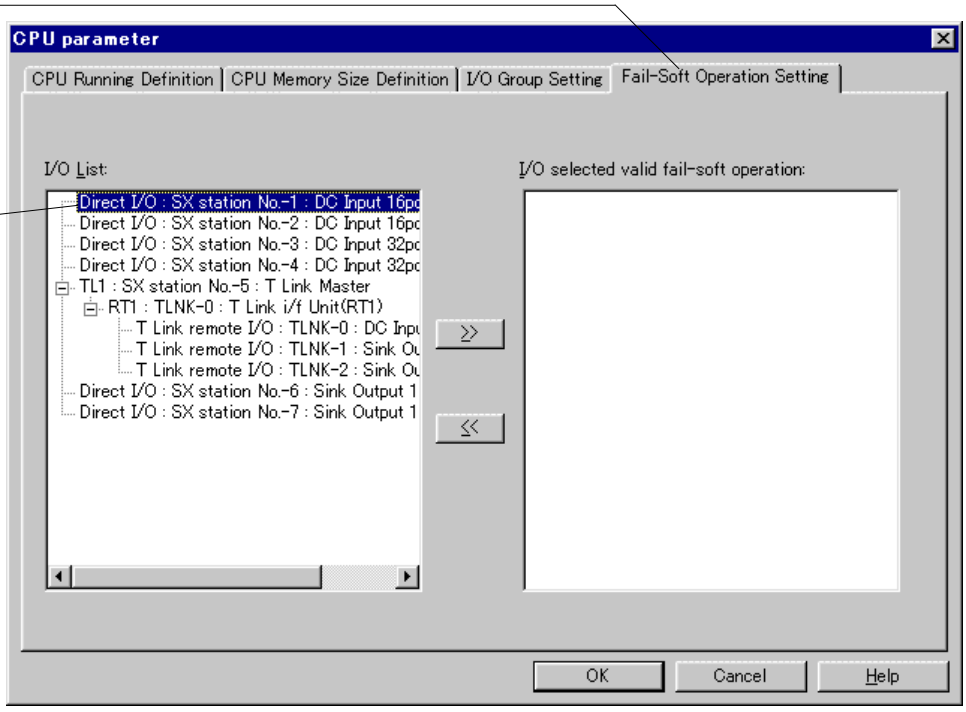
<System operation with any fault>



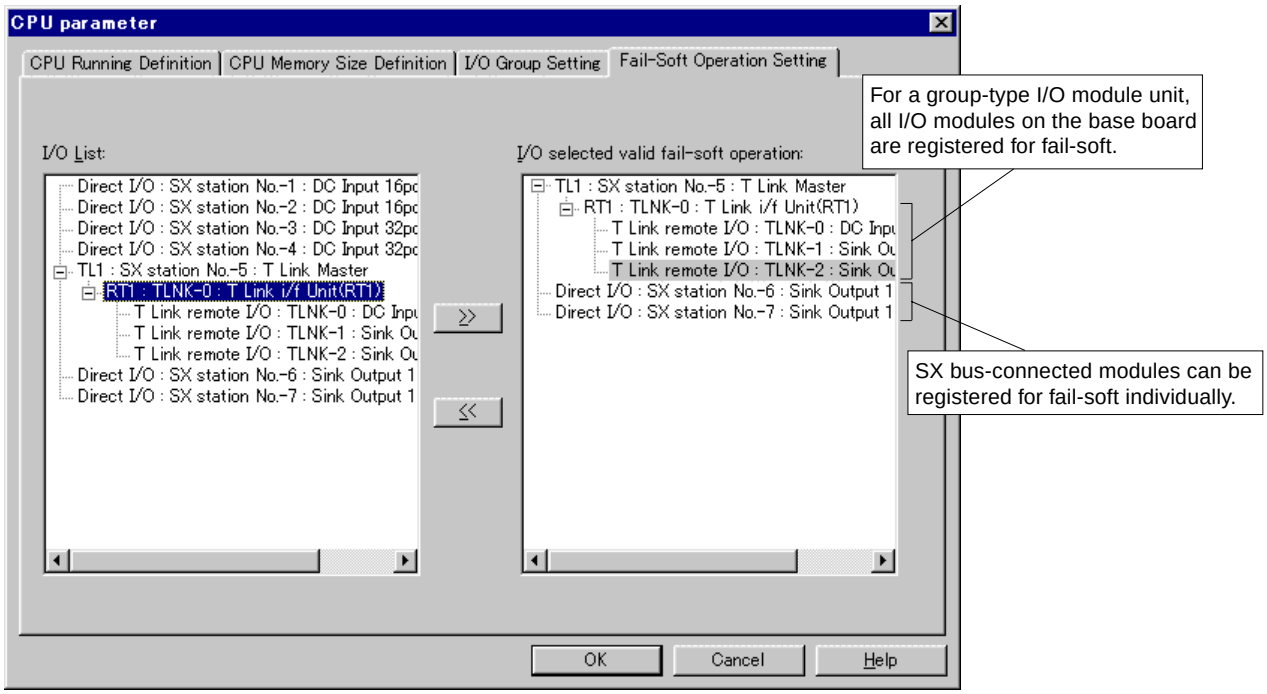
<Setup procedure>

1) Open the “CPU Parameter” dialog box and click the [Fail-soft operation setting] tab with the left mouse button.

Select the module (left click) for which fail-soft is to be set and left-click the [>>] button. You cannot set fail-soft individually for the I/O modules on a group-type I/O unit.



2) The modules are registered as shown below.

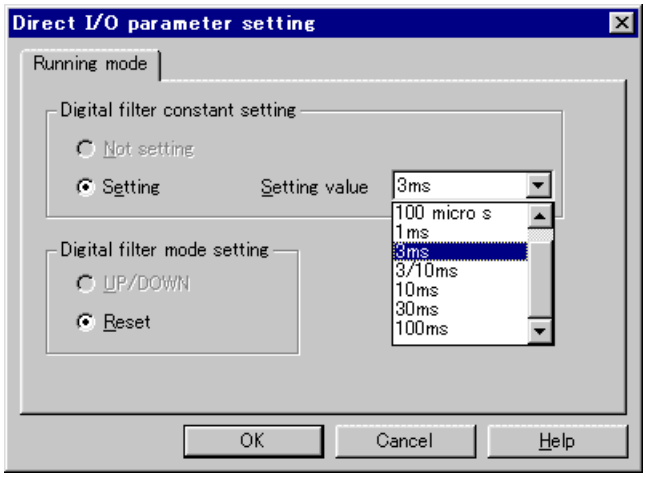


3-6-1 Input filtering time

The input filtering time is set for a DC-type digital input module. The time is set in the format (OFF-to-ON time) - (ON-to-OFF time). The user can select a value from 1-1ms, 3-3ms (default), 3-10ms, 10-10ms, 30-30ms, 100-100ms, and no filtering. Only for a fast input module (NP1X3206-A) can no filter or a 100-100µs input filtering time be set.

<Setup procedure>

1) From the system configuration definition tree, select a digital input module for which the input filtering time is to be set, and open the parameter set up dialog box.



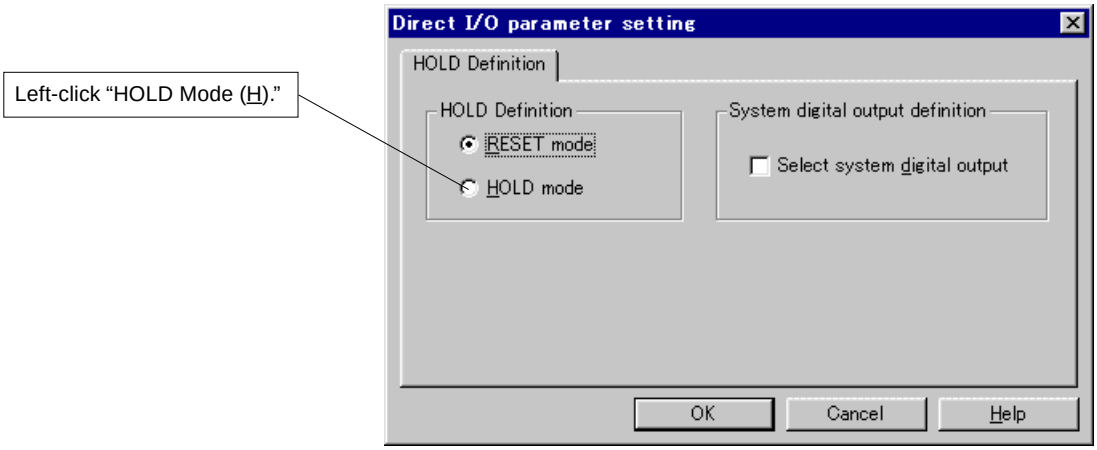
2) After specifying a filtering time, click the [OK] button with the left mouse button.

3-6-2 Output hold definition

The output hold definition is used to preserve the output state established immediately before a system error occurs or a CPU module is shut down or to preserve the output state established immediately before the CPU is stopped while the CPU is held stopped.

<Setup procedure>

1) Select, from the system configuration definition tree, a digital output module for which the output hold option is to be set, and open the parameter set up dialog box.



2) After selecting the hold mode, click the [OK] button with the left mouse button.

Note: For remote I/Os on JPCN-1, output hold cannot be set.

Section 4 CPU Duplex System

Page

Note: The CPU duplex system cannot be built using the standard CPUs.

Section 4 CPU Duplex System

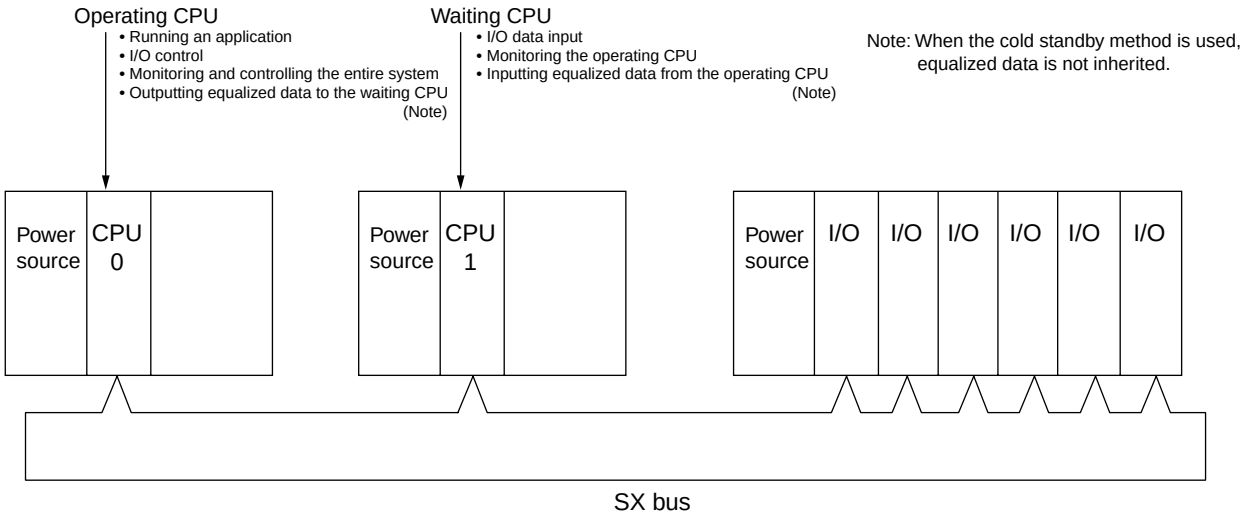
4-1 System operation in the duplex mode

This is called the duplex mode because dual devices are used to improve system safety and reliability in the control system. In the MICREX-SX Series, the power modules and CPU modules can be built into the dual systems. In this section, how to build the CPU modules into the duplex system is described.
Provided with MIXREX-SX, the CPU duplex mode includes 1 to 1 and N to 1 types.

4-1-1 1 to 1 duplex mode

In this mode, one operating CPU is associated with one waiting CPU. Combinations CPU0-CPU1, CPU2-CPU3, CPU4-CPU5, and CPU6-CPU7 are established as pairs of operating and waiting CPUs. In this case, the same application program is used.

<Example of 1 to 1 duplex system configuration>



(1) System operation

At system power-on, the system starts running, assuming that the CPU modules with even CPU numbers assigned are on the operating side while those with odd CPU numbers are on the waiting side. (In the above example, CPU0 is an operating CPU and CPU1 is a waiting one.)
When the operating CPU has a fault and goes down, the waiting CPU starts running.
The 1 to 1 duplex mode includes two types, warm standby in which the waiting CPU inherits data from the operating CPU and cold standby in which the waiting CPU does not do so. The data inherited by the waiting CPU is called equalized data, and its range is specified in the system definition.

(2) Replacing a faulty CPU with a new one

In the above example, since only the power module has been installed on the base unit with a CPU mounted on it, the CPU0 can be replaced while CPU1 is running instead of the downed CPU0. To replace the CPU0 with a new one, turn the CPU0 power off, replace it with a new one, and turn the power on. A new CPU0 is assumed to be on the waiting side.
When faults occur in both CPUs (on the operating and waiting sides), turn off the power on both of the systems and then re-start them.

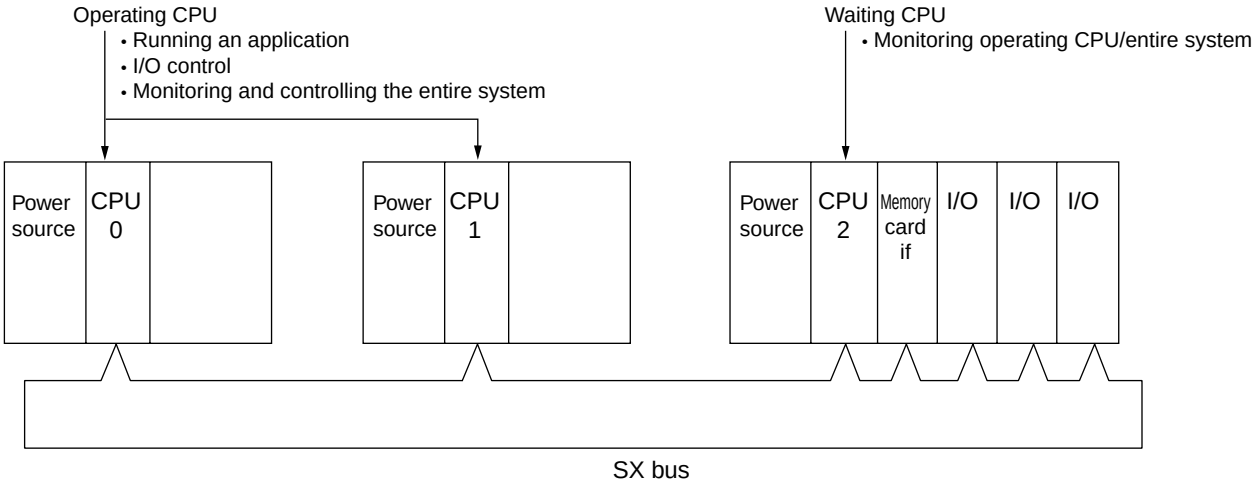
Kay points

- Install the same application on both the operating and waiting CPUs.
- In either case of the warm and cold standby methods, I/O data is inherited.
- The use of the loader allows you to switch between the operating and waiting CPUs.

4-1-2 N to 1 duplex mode

In this type of system, more than one (2-7) operating CPUs are associated with one waiting CPU. Up to two sets of N to 1 duplex groups can be defined for each configuration. The CPU with the largest CPU number among those in a group is a waiting CPU.

<Example of 2 : 1 duplex system configuration>



(1) System operation

At system power-on, the CPU module with the largest CPU number in the N to 1 duplex system is assumed to be a waiting CPU. (In the above example, CPU0 and CPU1 are operating CPUs and CPU2 is a waiting CPU.)
When the system goes down due to a fault in CPU0 or CPU1, the waiting CPU downloads the program of the faulty CPU from the memory card interface module and starts running.
In the N to 1 duplex mode, only the cold standby method can be applied. No data is inherited from the operating CPU.

(2) Replacing a faulty CPU with a new one

In the above example, since only the power module has been installed on the base unit with a CPU mounted on it, the CPU can be replaced while CPU2 is running instead of the downed CPU1.
To replace the CPU1 with a new one, turn the CPU1 power off, replace it with a new one, and turn the power on. A new CPU module is assumed to be on the waiting side, and is waiting for a switching instruction from the loader or system power reset. This means that the system is not in the N to 1 duplex system.

Key points

- N (the number of operating CPUs) application programs need to be stored on the memory card interface module.
- In the N to 1 duplex mode, only the cold standby method is applicable. No internal data and I/O data are inherited.
- The use of the loader allows you to switch between the operating and waiting CPUs. When the faulty CPU is replaced, the operating CPU needs to be switched to the waiting one.
- In the N to 1 duplex system, program read/write operation by the switches on the front face of the if memory card in the module is prohibited. Do not use the memory card if module for storing the application programs for the N to 1 duplex system with file memory for file read/write access from the application program running on the CPU. Prepare another memory card if module for file read/write.
  If file memory is used with the memory card if module, an access contention occurs and changeover may not be performed between the operating and waiting CPU.

4-2 Conditions for changeover between operating and waiting CPUs and performance

4-2-1 Conditions for changeover

The conditions in which changeover occurs between operating and waiting CPUs are shown below. The conditions are the same for both 1 to 1 and N to 1 modes.

○: changeover -: no changeover

Fatal fault in operating CPU	Fault in CPU	• Fault in application operation processor • Fault in OS processor	○
	Fault in memory	• Fault in system memory (ROM/RAM) • Fault in application memory (ROM/RAM) • Fault in memory battery backup	○
	Fault in SX bus	• Fault in SX bus control LSI • Fault in processor bus access (caused by self-module)	○
		• Duplicate station number • Excessive number of connected modules • Fault in SX bus transmission • Delay in I/O refresh	(Note 1)
Fatal fault in operating resource	Power failure	• Base power shutdown	○
	Application error	• User program error • Application WDT error • Application run error	○
	Fault in I/O module	• Fault in SX bus-connected I/O controlled by self CPU module • and remote I/O module (Fail-soft disabled)	–
	User fatal fault	• User fatal fault detected	–
Changeover instruction by loader	Changeover between operating and waiting CPUs by loader		○
Multi-CPU relay switch	In the multi-CPU duplex system (1 to 1 mode), one CPU is switched due to a fault, followed by another CPU.		○

Note:1) Since the SX bus is a common resource for the entire system, changeover cannot be done between operating and waiting CPUs when faults occur in both CPUs.
2) An intentional stopping of the CPU is excluded from the conditions for changeover.

4-2-2 System performance in the duplex mode

	Switching time	Takt time	Scan time
1 to 1 duplex mode cold standby	Within 130ms (Note 2)	The same Takt time as in the ordinary multi-CPU system	The same scan time as in the ordinary multi-CPU system
1 to 1 duplex mode warm standby	Within 130ms (Note 2)	Takt time in the ordinary multi-CPU system + 1-3ms	Scan time in the ordinary multi-CPU system + several ms-several tens of ms
N to 1 duplex mode cold standby	Several tens sec.	The same Takt time as in the ordinary multi-CPU system	The same scan time as in the ordinary multi-CPU system

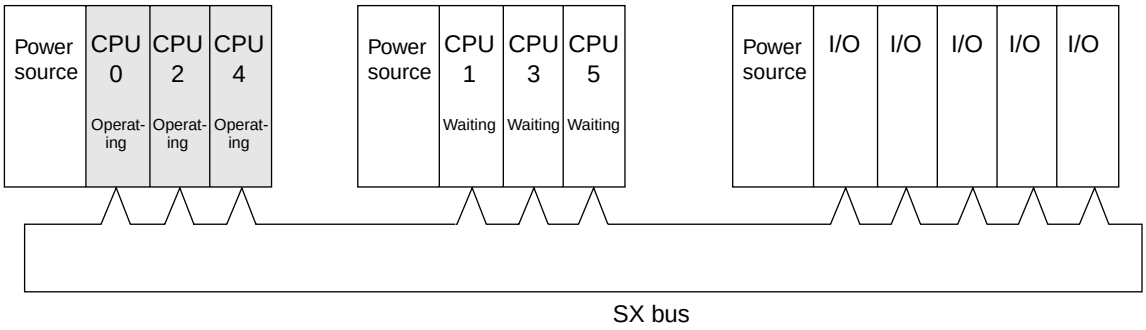
Note: 1) Depends on the quantity of equalized data.
2) If the message-related FB is used directly after changeover from the operating CPU to the waiting one, the busy status continues until the message closing process is completed (100-600ms).

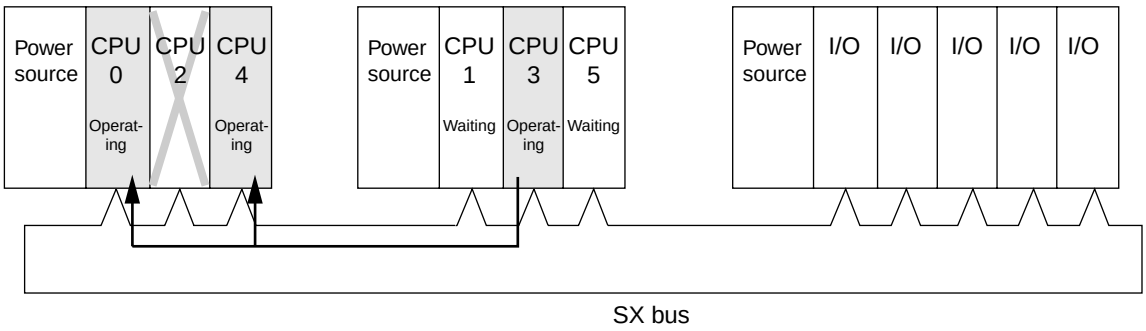
4-2 Conditions for changeover between operating and waiting CPUs and performance

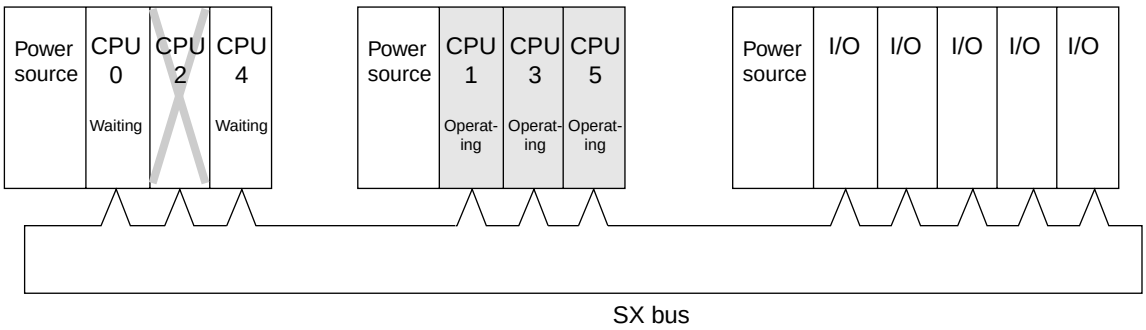
4-2-3 Multi-CPU relay switch

Usually, in the 1 to 1 duplex mode, changeover is done between a pair of operating and waiting CPUs. In the multi-CPU system, when you want to switch between another pair of CPUs following changeover between a pair of operating and waiting CPUs, set the relay switch mode. This enables the system to switch between both CPUs automatically.

<Example of three-pair relay switch system configuration in the 1 to 1 duplex mode>







In the above example, when CPU2 stops running due to a fault, CPU3 starts running instead. (The time for switching is 130ms max.) Then, CPU3 issues the switch command to CPU0 and CPU4. CPU0 and CPU4, when receiving the command, go into the waiting mode and CPU1 and CPU5 are switched to the operating mode. (The time for switching is 130ms max.)

Note: 1) While CPU2 has a fault in the above system configuration, the system cannot switch to the waiting CPU if a fault occurs in the operating CPU. The entire system goes down due to a fatal fault.

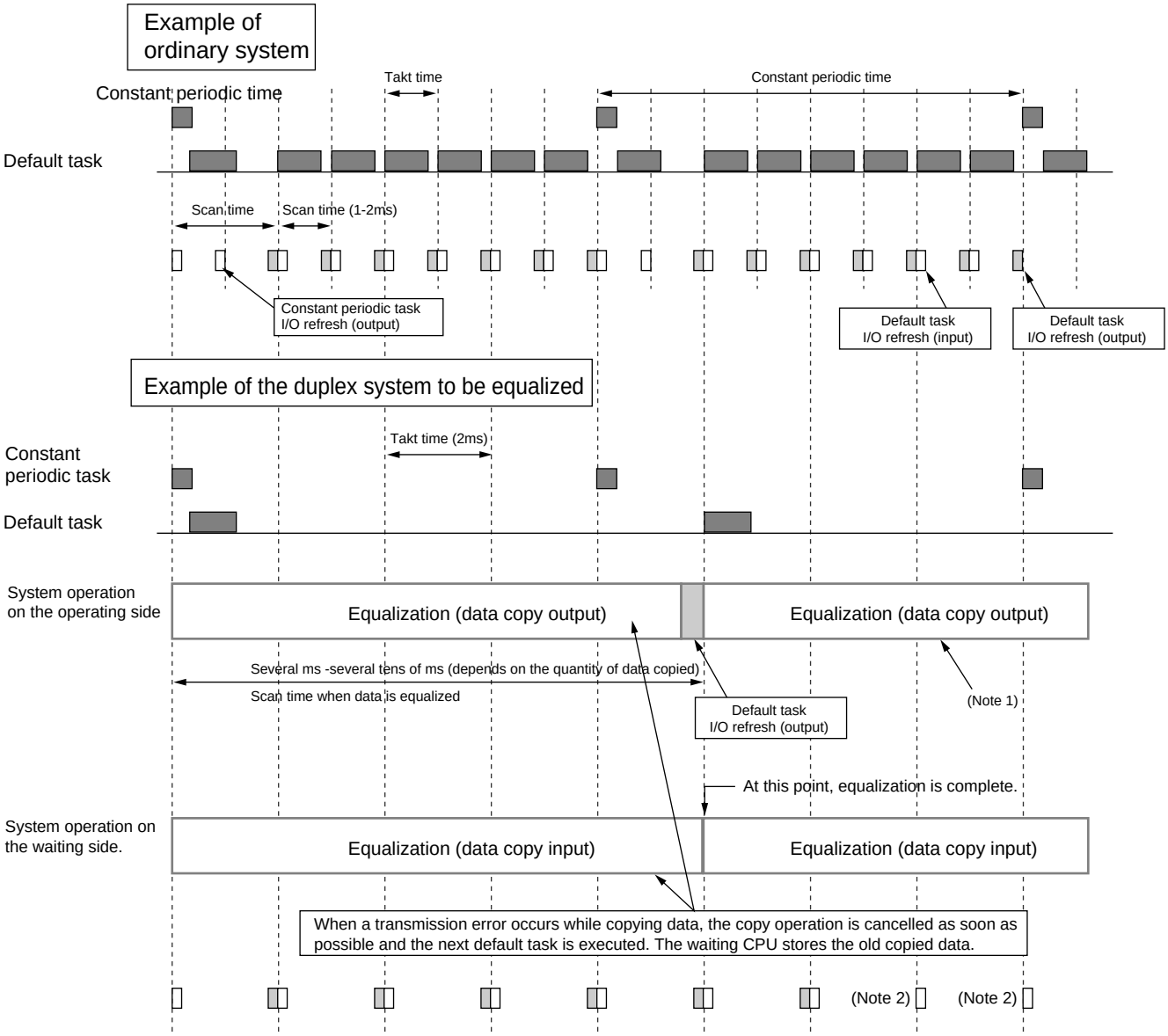
2) Relay switching can be done while both the operating and waiting systems are running normally.
For example, when changeover occurs during initialization (the operation and waiting sides are not undefined), even a CPU with an enabled relay switch may not be switched to another one. This means that both the operating and waiting CPUs run simultaneously in the CPU group for which the relay switch has been enabled.
Whether the relay switch has been operating normally can be determined by verifying that the resource running information (%MW10.48) bit, set to “1,” matches that for the CPU group for which the relay switch has been enabled.

4-2-4 Data equalization

In the 1 to 1 duplex mode warm standby system, the internal data of the operating CPU and waiting CPU with varying arithmetical operations may be equalized. This is called equalization. The specifications of equalization are described below.

(1) Timing for equalization

Data is equalized at the scan end of the default task (synchronized with the task). Data can be copied at the same timing as that of the scan end of the default task even if any task shorter than the scan time is executed.



Note: 1) When a switching factor occurs during scanning, the waiting CPU starts running with equalized data which has been scanned previously.

2) Data output stops at the first task point next to the point that a switching factor has occurred. No output from the I/O module is updated until the waiting CPU starts running.

Scan time for duplex system (outline)

= (Scan time of ordinary system) + {1 + (Total number of words for equalization) / 512 words} x Takt time

[Integral multiple of Takt time]

Fractions to be rounded

Note: The scan time increase when total number of words for equalization increase.

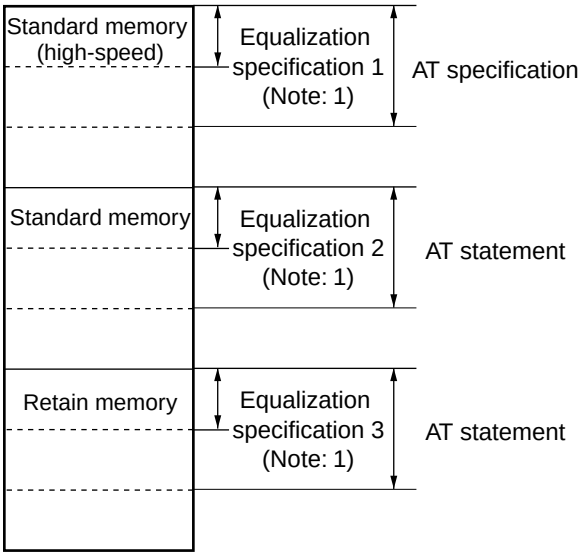
4-2 Conditions for changeover between operating and waiting CPUs and performance

(2) Equalized data area

The equalized data area includes the area in which data is equalized by user and the area equalized automatically by the system.

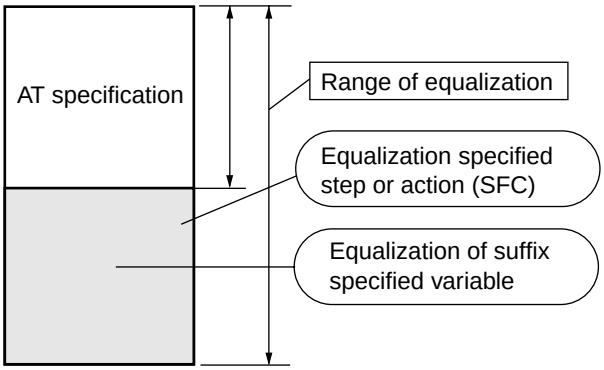
- User-specified equalized area → Standard memory (high-speed), standard memory, and retain memory
- System-equalized area → Variables defined for retain memory
  → Area with a retain attribute assigned in the system FB
  → Current timer value area in the system FB

1) User-specified area (variables used in the program)



Equalization size Spg
Spg=equalization 1 + equalization 2 + equalization 3

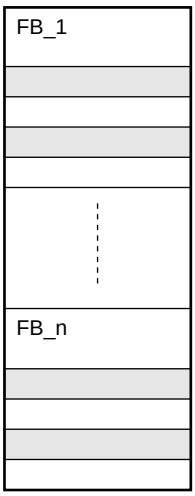
Note: 1) Set the equalization specification area so that it lies in the range specified in the AT statement and also specify the variables to be equalized in the AT statement.



Note: 2) When you use the function that automatically assigns variables having specified suffix or SFC step/action variables to the equalization area, the equalization area must be greater than the AT specified area.

2) Variables defined for retain among the user FB (User FB instance memory) variables

User FB instance memory



:Variables defined for retain

<Variable size >

- BOOL type variable ⟶ 1 word
- 16-bit data type variable ⟶ 1 word
- 32-bit data type variable ⟶ 2 words

Equalization size SUFB
SUFB= BOOL type variable + 16-bit data type variable + 32-bit data type variable

Note) In the 1 to 1 duplex mode, the maximum area which can be defined for retain in the user FB instance memory is 2048 words.

3) Retain attribute area and current timer value area in system FB instance memory

Equalized system FB	Equalized instance element	No. of words
Edge detection(R_TRIG, F_TRIG)	Old input value	2
Counter (CTU, CTD, CTUD, RCT)	Current counter value Old input value	2
Timer, totalizing timer (TP, TON, TOF, MR, TMR)	Current timer value, old input value, clocking flag	4

Equalization size SSFB
SSFB=No. of edge detection instructions x 2 + No. of counter instructions x 2 + No. of timer instructions x 4 (Note 1)
Note 1) Includes a totaling timer.
Note 2) The number of equalized instructions indicates the number of instructions available in each FB defined in the CPU memory size definition instead of that in the program.

(3) Equalized data size

The size of data to be equalized is up to 8192 words in the 1 to 1 duplex system.

$8192 \geq SPG + SUFB + SSFB$ (words)

Note: To use NP1PS-74 (powerful CPU74K product) in the 1 to 1 duplex mode warm standby system, be sure to define the memory boundaries considering equalized data. If default boundaries are used, equalization requires 819 words or more only for system FB instance memory.

4-2 Conditions for changeover between operating and waiting CPUs and performance

4-2-5 Memory operation at changeover between operating and waitin ing and waiting CPUs

System status	Memory or flag	1 to 1 duplex mode		N to 1 duplex mode
		Cold standby	Warm standby	
System power-on (warm running)	Standard memory	Cleared		
	Retain memory	Old retained value		
	I/O memory	Reset hold specification		
	Default task start flag	ON		
	Initial flag	OFF		
Operating ⟹ waiting	Standard memory	Values are retained during running		
	Retain memory			
	I/O memory			
Waiting ⟹ operating	Standard memory	Cleared	An area other than the area to be equalized is cleared, and the data in the area to be equalized is inherited by the operating CPU.	Cleared
	Retain memory	Old values are retained.	Old values are retained and data in the area to be equalized is inherited by the operating CPU.	Cleared
	I/O memory	Data remains unchanged	Data remains unchanged	Reset/hold
	Default task start flag	ON	ON	ON
	Initial flag	OFF (Note 1)	OFF (Note 1)	ON
	Operating/waiting changeover flag	ON	ON	ON
Cold running (at program download or initial start-up by loader)	Standard memory	Cleared		
	Retain memory	Cleared		
	I/O memory	Cleared		
	Default task start flag	ON		
	Initial flag	ON		

Note: 1) When the waiting CPU does not run after program download, the initial flag is set to “1.”
2) In the duplex system, to pass data to the bit variables used for semaphores at changeover between the operating and waiting CPUs:
• Assign the bit variables for semaphores to the standard memory area to avoid equalization.
(They are reset to 0 atchangeover.)
• When changeover occurs, make an attempt to get semaphores again.

Key points

<System operation at changeover>
• 1 to 1 duplex mode standby system
The system operates in the same manner as the system for single CPU warm running with the exception that data in I/O memory is inherited.
• 1 to 1 duplex warm standby system
This is the cold standby system with data equalization added.
• N to 1 duplex system
Since the program is downloaded from the memory card interface module, the system uses the cold standby method.

4-3 CPU module LEDs and output to display system

○: On, -: Off, △: Blinking

Faulty module / Module			The entire system operates normally	The entire system stops normally	Fatal fault in operating CPU module	Fatal fault in waiting CPU module	Fatal fault in operating CPU module	I/O fault (fail-soft enabled)	II/O fault (fail-soft disabled)
							Application error	Non-fatal fault in operating/waiting CPU resource	Fatal fault in operating/waiting CPU resource
Operating CPU	LED indication	ON	○	○	—	○	○	○	○
		ERR	—	—	○	—	—	—	—
		RUN	○	—	—	○	—	○	—
		ALM	—	—	○	○	○	○	○
	Flag		Normal run	Normal run	Fatal fault in self CPU resource	Fatal fault in waiting CPU resource	Fatal fault in self CPU resource	Fault in I/O module Non-fatal fault in self-CPU resource	Fault in I/O module Fatal fault in self-CPU resource
Waiting CPU	LED indication	ON	○	○	○	—	○	○	○
		ERR	—	—	—	○	—	—	—
		RUN	△	—	○	—	○	△	—
		ALM	—	—	○	○	—	○	○
	Flag		Normal run	Normal run	Fatal fault in operating CPU	Fatal fault in self-CPU	Fatal fault in operating CPU	Fault in I/O module Non-fatal fault in self-CPU resource	Fault in I/O module Fatal fault in self-CPU resource
System DO	LED indication	ON	○	○	○	○	○	○	○
		ERR	—	—	—	—	—	—	○
		System DO	○	—	○	○	○	○	—
	Contact output		ON (Running)	OFF (Stop)	ON (Running)	ON (Running)	ON (Running)	ON (Running)	OFF (Stop)

4-4 Application of the duplex system

4-4-1 Successively start of CPU module

In the duplex mode, CPUs other than CPU0 can be successively started without enabling fail-soft.

Note that CPU0 must be mounted before the system can start.

Note: If all the CPU modules have not been configured, download the program to them individually by using the loader.

4-4-2 1 to 1 duplex system

(1) Either an operating or waiting CPU has been configured

1) Even if non-default waiting CPUs (odd CPU numbers) have been configured, the system can start with a non-fatal system fault.

2) Even if non-default operating CPUs (even CPU numbers) have been configured, the system can start with a non-fatal system fault.

Note: In the case of 2), the system starts with all the default operating CPUs, for which the relay switch is enabled, switched to waiting CPUs.

(2) If neither an operating nor waiting CPU has been configured

In the 1 to 1 duplex system, the system starts with a non-fatal fault using the remaining CPUs, even if both the operating and waiting CPUs have not been configured.

Note: With the relay switch enabled, a fatal fault occurs in the system.

4-4-3 N to 1 duplex system

(1) Some operating CPUs or a waiting CPU have not been configured

1) With a non-default waiting CPU (the CPU with the largest number assigned) configured, the system starts with a non-fatal fault.

2) With some default operating CPUs not configured, the system starts with a non-fatal fault.

(2) Some operating CPUs and a waiting CPU have not been configured

The system starts with the remaining CPUs with a non-fatal fault.

4-4-4 System configuration definition

1) In the duplex system (1 to 1 and N to 1), specify the same data for all the CPUs in the system configuration definition. System configuration definition information is transferred to all the CPUs by one download operation.

2) When the actual configuration is different from that in the system configuration definition, a system definition error occurs in the entire system. If some CPUs have information different from that for others, a system definition error occurs only in those CPUs.

Appendix 1 Instruction Proces Speed Chart

Page

Appendix 1 Instruction Processing Speed Chart

1-1 High-performance CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed (1)	Processing Speed (2)
IL language	Load LD	0.02	0.06
	Load not LDN	0.02	0.06
	Store ST	0.02	0.06
	Store not STN	0.02	0.06
	Set S	0.04	0.12
	Reset R	0.04	0.12
	Logical product AND	0.02	0.06
	Logical product AND(	0.02	0.02
	Logical inverted product ANDN	0.02	0.06
	Logical inverted product ANDN(	0.02	0.02
	Logical add OR	0.02	0.06
	Logical add OR(	0.02	0.02
	Logical inverted add ORN	0.02	0.06
	Logical inverted add ORN(	0.02	0.02
	Exclusive OR XOR	0.02	0.06
	Exclusive OR XOR(	0.02	0.02
	Exclusive NOR XORN	0.02	0.06
	Exclusive NOR XORN(	0.02	0.02
	Addition ADD	0.04	0.08
	Addition ADD(	0.04	0.04
	Subtraction SUB	0.04	0.08
	Subtraction SUB(	0.04	0.04
	Multiplication MUL	0.06	0.10
	Multiplication MUL(	0.06	0.06
	Division DIV	1.08	1.12
	Division DIV(	1.08	1.08
	Comparison GT (>)	0.04	0.08
	Comparison GT((>)	0.04	0.04
	Comparison GE (≥)	0.04	0.08
	Comparison GE((≥)	0.04	0.04
	Comparison EQ (=)	0.04	0.08
	Comparison EQ((=)	0.04	0.04
	Comparison NE (≠)	0.04	0.08
	Comparison NE((≠)	0.04	0.04
	Comparison LE (≠)	0.04	0.08
	Comparison LE((≤)	0.04	0.04
	Comparison LT (<)	0.04	0.08
	Comparison LT (<)	0.04	0.04
	Unconditional jump JMP	0.08	0.08
	TRUE conditional jump JMPC	0.10	0.10

*Processing speed (1) indicates the execution speed of the instruction as executed with operands of the data type that is processed most quickly and stored in the high-speed memory area and/or input/output memory area.

Processing speed (2) indicates the execution speed of the instruction as executed with operands of the data type that is processed most quickly and stored in areas other than the high-speed memory area and/or input/output memory area.

1-1 High-performance CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed (1)	Processing Speed (2)
IL language	FALSE conditional jump JMPCN	0.10	0.10
	Label of the destination address	0.02	0.02
	Unconditional call CAL	FCT, FB(0.28) User FCT (3.5) User FB (3.0)	
	TRUE conditional call CALC		
	FALSE conditional call CALCN		
	Unconditional return RET	User FCT (3.5) User FB (3.0)	
	TRUE conditional return RETC		
	FALSE conditional return RETCN		
	Closing parenthesis	-	-
ST language	Assignment statement :=	Note: In the ST language, the number of instructions (steps) of a control group which is made up of statements after compilation, varies from application to application. Consequently, it is difficult to compute the control processing speed from an ST language list. When it is necessary to grasp processing speed, it is recommended that the processing speed be measured by using an actual machine.	
	IF statement		
	CASE statement		
	FOR statement		
	WHILE statement		
	REPEAT statement		
	Rturn statement		
	Exit statement		
LD language	Normal open contact (NO contact)	0.02	0.06
	Normal close contact (NC contact)	0.02	0.06
	Coil	0.04	0.12
	Inverted coil	0.04	0.12
	Set	0.04	0.12
	Reset	0.04	0.12
	Connect to/from	0	0
	Jump	0.06	0.06
Type conversion function	Type conversion DINT_TO_INT	0.52	0.60
	Type conversion UINT_TO_INT	0.38	0.46
	Type conversion UDINT_TO_INT	0.38	0.46
	Type conversion REAL_TO_INT	0.70	0.78
	Type conversion TIME_TO_INT	0.38	0.46
	Type conversion WORD_TO_INT	0.06	0.14
	Type conversion INT_TO_DINT	0.06	0.14
	Type conversion UINT_TO_DINT	0.06	0.14
	Type conversion UDINT_TO_DINT	0.40	0.48
	Type conversion REAL_TO_DINT	0.10	0.18
	Type conversion TIME_TO_DINT	0.40	0.48
	Type conversion DWORD_TO_DINT	0.06	0.14
	Type conversion INT_TO_UINT	0.42	0.50
	Type conversion DINT_TO_UINT	0.52	0.60
	Type conversion UDINT_TO_UINT	0.38	0.46
	Type conversion REAL_TO_UINT	0.66	0.74
	Type conversion TIME_TO_UINT	0.38	0.46
	Type conversion WORD_TO_UINT	0.06	0.14

1-1 High-performance CPU Instruction Processing Speed Chart

All values in µs.

Language	Name	Processing Speed (1)	Processing Speed (2)
Type conversion function	Type conversion INT_TO_UDINT	0.38	0.46
	Type conversion DINT_TO_UDINT	0.38	0.46
	Type conversion UINT_TO_UDINT	0.06	0.14
	Type conversion REAL_TO_UDINT	0.76	0.84
	Type conversion TIME_TO_UDINT	0.06	0.14
	Type conversion DWORD_TO_UDINT	0.06	0.14
	Type conversion DT_TO_UDINT	0.06	0.14
	Type conversion DATE_TO_UDINT	2.00	2.08
	Type conversion TOD_TO_UDINT	0.06	0.14
	Type conversion INT_TO_REAL	0.34	0.42
	Type conversion DINT_TO_REAL	0.10	0.18
	Type conversion UINT_TO_REAL	0.34	0.42
	Type conversion UDINT_TO_REAL	0.46	0.54
	Type conversion TIME_TO_REAL	0.46	0.54
	Type conversion WORD_TO_BOOL	0.06	0.14
	Type conversion DWORD_TO_BOOL	0.06	0.14
	Type conversion BOOL_TO_WORD	0.06	0.14
	Type conversion DWORD_TO_WORD	0.06	0.14
	Type conversion INT_TO_WORD	0.06	0.14
	Type conversion UINT_TO_WORD	0.06	0.14
	Type conversion BOOL_TO_DWORD	0.06	0.14
	Type conversion WORD_TO_DWORD	0.06	0.14
	Type conversion DINT_TO_DWORD	0.06	0.14
	Type conversion UDINT_TO_DWORD	0.06	0.14
	Type conversion INT_TO_TIME	0.38	0.46
	Type conversion DINT_TO_TIME	0.38	0.46
	Type conversion UINT_TO_TIME	0.06	0.14
	Type conversion UDINT_TO_TIME	0.06	0.14
	Type conversion REAL_TO_TIME	0.76	0.84
	Type conversion UDINT_TO_DT	0.06	0.14
	Type conversion UDINT_TO_DATE	2.00	2.08
	Type conversion UDINT_TO_TOD	2.00	2.08
	Type conversion TRUNC_INT	0.70	0.78
	Type conversion TRUNC_DINT	0.10	0.18
	Type conversion TRUNC_UINT	0.66	0.74
	Type conversion TRUNC_UDINT	0.76	0.84
	Type conversion W_BCD_TO_INT	0.12	0.20
	Type conversion D_BCD_TO_INT	0.58	0.66
	Type conversion W_BCD_TO_DINT	0.48	0.56
	Type conversion D_BCD_TO_DINT	0.12	0.20
	Type conversion INT_TO_W_BCD	0.12	0.20
	Type conversion DINT_TO_W_BCD	0.56	0.64
	Type conversion INT_TO_D_BCD	0.48	0.56
	Type conversion DINT_TO_D_BCD	0.12	0.20

1-1 High-performance CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed (1)	Processing Speed (2)
Arithmetic function	Absolute value ABS_INT	0.50	0.50
	Absolute value ABS_DINT	0.50	0.50
	Absolute value ABS_REAL	0.10	0.10
	Square root (√) SQRT	4.24	4.32
	Natural logarithm LN	5.44	5.52
	Common logarithm LOG	5.84	5.92
	Exponent EXP	9.64	9.72
	Sine SIN	6.24	6.32
	Cosine COS	6.24	6.32
	Tangent TAN	11.04	11.12
	Arc sine ASIN	11.84	11.92
	Arc cosine ACOS	11.84	11.92
	Arc tangent ATAN	6.84	6.92
	Addition ADD	0.08 + (No. of inputs -1) x 0.04	0.16 + (No. of inputs -1) x 0.08
	Subtraction SUB	0.08	0.20
	Multiplication MUL	0.08 + (No. of inputs -1) x 0.06	0.16 + (No. of inputs -1) x 0.10
	Division DIV	1.12	1.24
	Division remainder MOD	1.12	1.24
	Exponent EXPT	16.86	16.98
	Move MOVE	0.08	0.16
	Negation NEG	0.06	0.14
Bit string function	Shift left SHL_WORD	0.08	0.20
	Shift left SHL_DWORD	0.08	0.20
	Shift right SHR_WORD	0.08	0.20
	Shift right SHR_DWORD	0.08	0.20
	Rotate left ROL_WORD	0.08	0.20
	Rotate left ROL_DWORD	0.08	0.20
	Rotate right ROR_WORD	0.08	0.20
	Rotate right ROR_DWORD	0.08	0.20
	Logical product AND	0.08 + (No. of inputs -1) x 0.04	0.16 + (No. of inputs -1) x 0.08
	Logical add OR		
	Exclusive XOR		
	Logical negation NOT	0.06	0.14
	Negation NOT_BOOL	0.06	0.14
	Negation NOT_WORD	0.06	0.14
	Negation NOT_DWORD	0.06	0.14

1-1 High-performance CPU Instruction Processing Speed Chart

All values in µs.

Language	Name	Processing Speed (1)	Processing Speed (2)
Selection/ comparison function	Select SEL_BOOL	0.30	0.50
	Select SEL_INT	0.30	0.50
	Select SEL_DINT	0.30	0.50
	Select SEL_UINT	0.30	0.50
	Select SEL_UDINT	0.30	0.50
	Select SEL_REAL	0.30	0.50
	Select SEL_WORD	0.30	0.50
	Select SEL_DWORD	0.30	0.50
	Select SEL_TIME	0.30	0.50
	Select SEL_STRING	0.30	0.50
	Maximum value MAX_INT	0.04 + (No. of inputs - 1) x 0.38	0.12 + (No. of inputs -1) x 0.42
	Maximum value MAX_DINT		
	Maximum value MAX_UINT		
	Maximum value MAX_UDINT		
	Maximum value MAX_REAL		
	Minimum value MIN_INT		
	Minimum value MIN_DINT		
	Minimum value MIN_UINT		
	Minimum value MIN_UDINT		
	Minimum value MIN_REAL		
	Limit LIMIT_INT	0.88	1.00
	Limit LIMIT_DINT	0.88	1.00
	Limit LIMIT_UINT	0.88	1.00
	Limit LIMIT_UDINT	0.88	1.00
	Limit LIMIT_REAL	0.88	1.00
	Comparison (>) GT	0.08 + (No. of inputs - 1) x 0.06	0.14 + (No. of inputs- 1)) x 0.1
	Comparison (≥) GE		
	Comparison (=) EQ		
	Comparison (≤) LE		
	Comparison (<) LT		
	Comparison (≠) NE	0.08	0.20

1-1 High-performance CPU Instruction Processing Speed Chart

All values in µs.

Language	Name	Processing Speed (1)	Processing Speed (2)
String function	Get Length LEN	0.84 + 0.3 x (No. of characters)	0.84 + 0.34 x (No. of characters)
	Get left sub-string LEFT	1.44 + 0.48 x (No. of characters to extract)	1.48 + 0.56 x (No. of characters to extract)
	Get right sub-string RIGHT	1.6 + 0.16 x (No. of characters) + 0.18 x (No. of characters to extract)	
	Get middle sub-string MID	1.66 + 048 x (No. of characters to extract)	1.7 + 0.56 x (No. of characters to extract)
	Concatenate CONCAT	2.8 + 0.48 x (No. of characters of first input + No. of characters of second input)	2.8 + 0.56 x (No. of characters of first input + No. of characters of second input)
	Insert string INSERT	2.04 + 1.06 x (No. of input characters) + 0.48 x (No. of characters to insert)	2.16 + 1.22 x (No. of input characters) + 0.56 x (No. of characters to insert)
	Delete string DELETE	2.04 + 1.06 x (No. of input characters - No. of characters to delete) + 0.36 x (No. of characters to delete)	2.16+1.22 x (No. of input characters - (No. of characters to delete) + 0.56 x (No. of characters to delete)
	Replace string REPLACE	2.42 + 1.06 x (No. of input characters - No. of replacement characters) + 0.6 x (No. of replacement characters)	2.7 + 1.22 x (No. of input characters - (No. of replacement characters) + 0.68 x (No. of replacement characters)
	Find String FIND	0.96 + (0.26 + 0.54 x (No. of input characters) x (No. of characters to search)	0.96 + (0.3 + 0.58 x (No. of input characters) x (No. of characters to search)
	Compare String (>) GT_STRING	1.24 + 0.58 x (No. of input characters)	1.32 + 0.66 x (No. of input characters)
	Compare String (≥) GE_STRING		
	Compare String (=) EQ_STRING		
	Compare String (≤) LE_STRING		
	Compare String (≤) LT_STRING		
	Compare String (≠) NE_STRING	1.14 + 0.3 x (No. of input characters)	
Time type data function	Add time ADD_T_T	0.12	0.24
	Add time ADD_TD_T	3.00	3.18
	Add time ADD_DT_T	1.22	2.06
	Subtract time SUB_T_T	0.12	0.24
	Subtract time SUB_D_D	0.18	0.34
	Subtract time SUB_TD_T	3.00	3.18
	Subtract time SUB_TD_TD	0.18	0.34
	Subtract time SUB_DT_T	1.22	2.06
	Subtract time SUB_DT_DT	0.18	0.34
	Multiply time MUL_T_N	0.50	0.62
	Multiply time MUL_T_R	0.54	0.66
	Divide time DIV_T_N	2.10	2.36
	Divide time DIV_T_R	2.10	2.36
	Concatenate time CONCAT_D_D	0.12	0.24
	Convert DT to TOD DT_TO_TOD	1.16	1.28
	Convert DT to DATE DT_TO_DATE	1.76	2.06

1-1 High-performance CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed (1)	Processing Speed (2)
Original function	Set bit SBIT_WORD	0.16	0.28
	Set bit SBIT_DWORD	0.16	0.28
	Reset bit RBIT_WORD	0.16	0.28
	Reset bit RBIT_DWORD	0.16	0.28
	Test bit TBIT_WORD	0.14	0.26
	Test bit TBIT_DWORD	0.14	0.26
	Decode DECODE_WORD	0.06	0.14
	Decode DECORD_DWORD	0.08	0.16
	Encode ENCODE_WORD	0.06	0.14
	Encode ENCODE_DWORD	0.08	0.16
	Bit count BITCOUNT_WORD	1.82	1.90
	Bit count BITCOUNT_DWORD	3.42	3.50
	Convert string to number STR_TO_UINT	0.62 + 0.76 x (No. of input characters)	0.66 + 0.80 x (No. of input characters)
	Convert number to string UINT_TO_STR	1.36 + 1.82 x (No. of output characters)	1.36 + 1.94 x (No. of output characters)
	Convert shift-JIS to string SJ_TO_STR	2.76 + 1.98 x (No. of output characters)	3.02 + 2.1 x (No. of output characters)
	Convert string to shift-JIS STR_TO_SJ	1.46 + 0.94 x (No. of input characters)	1.52 + 1.02 x (No. of input characters)
	Byte length BYTE_LEN	0.66 + 0.46 x (No. of input characters)	0.7 + 0.5 x (No. of input characters)
	Dead band DBAND_INT	1.34	1.46
	Dead band DBAND_DINT	1.34	1.46
	Dead band DBAND_REAL	1.46	1.58
	Bias BIAS_INT	1.14	1.26
	Bias BIAS_DINT	1.14	1.26
	Bias BIAS_REAL	1.20	1.32
	Step sequence coil SC_COIL/SC	1.12/1.12	1.28/1.28
	32-bit addition with carry ADC/ADCO	1.02/1.16	1.18/1.32
	32-bit subtraction with borrow SBB/SBBO	1.04/1.16	1.20/1.32
	64-bit multiplication MULL/MULU	1.14/1.14	1.26/1.26
	64-bit division DIVL/DIVU	4.60/4.60	4.76/4.76
	Shift left 32-bits with carry SLC/SLCO	1.14/1.08	1.26/1.16
	Shift right 32-bits with carry SRC/SRCO	1.14/1.08	1.26/1.26
Standard function block	Set reset flip-flop SR	0.16	0.16
	Reset-set flip-flop RS	0.16	0.16
	Rising edge trigger R_TRIG	0.16	0.16
	Falling edge trigger F_TRIG	0.16	0.16
	Up counter CTU	0.22	0.22
	Down counter CTD	0.22	0.22
	Up down counter CTUD	0.22	0.22

1-1 High-performance CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed (1), (2)
Standard function block	Pulse TP	0.44
	On-delay timer TON	0.44
	Off-delay timer TOF	0.44
	Real-time clock RTC	1.76
Original function block	Ring counter RCT	0.22
	Integrating timer TMR	0.44
	Retriggerable MR	0.44
	Open channel M_OPEN	1.60
	Send message M_SEND	1.76
	Receive message M_RECEIVE	1.92
	Direct read READ_BOOL	1.50
	Direct read READ_WORD	1.50
	Direct write WRITE_BOOL	1.30
	Direct write WRITE_WORD	1.30
	Remote data read R_READ	1.50
	Remote data write R_WRITE	1.30
	File data read F_READ	1.50
	File data write F_WRITE	1.30
	Extension test & set EXT_T_S	3.00
	Sequential file store FFST	3.08 + (No. of words) x 0.12
	Sequential file load first FIFO	2.68 + (No. of words) x 0.12
	Sequential file load last FILO	2.68 + (No. of words) x 0.12
	Filter FILTER_DINT	4.32
	Filter FILTER_REAL	2.98
	Integrate INT_DINT	4.60
	Integrate INT_REAL	4.02
	Differentiate DIF_DINT	3.24
	Differentiate DIF_REAL	3.48
	Pulse count PULSE_CNT	2.80
	Pulse output PULSE_OUT	2.28
	Pulse PWM	2.48
	Hardware RTC (Real-time clock) HW_RTC	1.20
	Test & set T_S	0.86
	Change bank BANK_CHG	4.72

Processing speed of <SFC elements>

For SFC elements, only steps that are currently active are executed. Consequently, it is difficult to discuss the processing speed of SFC elements using only the worksheet on which the SFC elements are programmed. When it is necessary to grasp processing speed, it is recommended that the processing speed be measured by using an actual machine.

1-2 Standard CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed
IL language	Load LD	0.07
	Load not LDN	0.07
	Store ST	0.14
	Store not STN	0.14
	Set S	0.14
	Reset R	0.14
	Logical product AND	0.07
	Logical product AND(	0.07
	Logical inverted product ANDN	0.07
	Logical inverted product ANDN(	0.07
	Logical add OR	0.07
	Logical add OR(	0.07
	Logical inverted add ORN	0.07
	Logical inverted add ORN(	0.07
	Exclusive OR XOR	0.07
	Exclusive OR XOR(	0.07
	Exclusive NOR XORN	0.07
	Exclusive NOR XORN(	0.07
	Addition ADD	0.21
	Addition ADD(	0.21
	Subtraction SUB	0.21
	Subtraction SUB(	0.21
	Multiplication MUL	10.14
	Multiplication MUL(	10.14
	Division DIV	10.14
	Division DIV(	10.14
	Comparison GT (>)	0.63
	Comparison GT((>)	0.63
	Comparison GE (≥)	0.63
	Comparison GE((≥)	0.63
	Comparison EQ (=)	0.63
	Comparison EQ((=)	0.63
	Comparison NE (≠)	0.49
	Comparison NE((≠)	0.49
	Comparison LE (≠)	0.7
	Comparison LE((≤)	0.7
	Comparison LT (<)	0.7
	Comparison LT (<)	0.7
	Unconditional jump JMP	0.21
	TRUE conditional jump JMPC	0.14 (Non-JMP) /0.21(JMP)

* The processing speed depends on the data type to be used. The data types for which operands are processed at the highest speeds are listed in the table.

All values in μs.

Language	Name	Processing Speed (1)
IL language	FALSE conditional jump JMPCN	0.14 (Non-JMP) /0.21 (JMP)
	Label of the destination address	0.14
	Unconditional call CAL	FCT(8.14), FB(9.14) User FCT (19.14) User FB (32.14) — The value enclosed in ( ) is the value when a call is executed. The value is 0.4μs when a call is not executed.
	TRUE conditional call CALC	
	FALSE conditional call CALCN	
	Unconditional return RET	User FCT (15.14) User FB (18.14) — The value enclosed in ( ) is the value when a call is executed. The value is 0.4μs when a call is not executed.
	TRUE conditional return RETC	
	FALSE conditional return RETCN	
	Closing parenthesis	-
ST language	Assignment statement :=	Note: In the ST language, the number of instructions (steps) of a control group which is made up of statements after compilation, varies from application to application. Consequently, it is difficult to compute the control processing speed from an ST language list. When it is necessary to grasp processing speed, it is recommended that the processing speed be measured by using an actual machine.
	IF statement	
	CASE statement	
	FOR statement	
	WHILE statement	
	REPEAT statement	
	Rturn statement	
	Exit statement	
LD language	Normal open contact (NO contact)	0.07
	Normal close contact (NC contact)	0.07
	Coil	0.14
	Inverted coil	0.14
	Set	0.14
	Reset	0.14
	Connect to/from	0
	Jump	0.14 (Non-JMP) /0.21 (JMP)
Type conversion function	Type conversion DINT_TO_INT	10.42
	Type conversion UINT_TO_INT	7.35
	Type conversion UDINT_TO_INT	8.42
	Type conversion REAL_TO_INT	12.42
	Type conversion TIME_TO_INT	8.42
	Type conversion WORD_TO_INT	7.35
	Type conversion INT_TO_DINT	8.42
	Type conversion UINT_TO_DINT	7.42
	Type conversion UDINT_TO_DINT	7.49
	Type conversion REAL_TO_DINT	23.49
	Type conversion TIME_TO_DINT	7.49
	Type conversion DWORD_TO_DINT	6.7
	Type conversion INT_TO_UINT	8.35
	Type conversion DINT_TO_UINT	8.42
	Type conversion UDINT_TO_UINT	8.42
	Type conversion REAL_TO_UINT	12.42
	Type conversion TIME_TO_UINT	8.42
	Type conversion WORD_TO_UINT	7.35

1-1 Standard CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed
Type conversion function	Type conversion INT_TO_UDINT	8.42
	Type conversion DINT_TO_UDINT	7.49
	Type conversion UINT_TO_UDINT	7.42
	Type conversion REAL_TO_UDINT	21.49
	Type conversion TIME_TO_UDINT	6.7
	Type conversion DWORD_TO_UDINT	6.7
	Type conversion DT_TO_UDINT	0.7
	Type conversion DATE_TO_UDINT	0.7
	Type conversion TOD_TO_UDINT	0.7
	Type conversion INT_TO_REAL	14.42
	Type conversion DINT_TO_REAL	17.49
	Type conversion UINT_TO_REAL	12.42
	Type conversion UDINT_TO_REAL	14.49
	Type conversion TIME_TO_REAL	14.49
	Type conversion WORD_TO_BOOL	0.42
	Type conversion DWORD_TO_BOOL	0.49
	Type conversion BOOL_TO_WORD	0.35
	Type conversion DWORD_TO_WORD	0.42
	Type conversion INT_TO_WORD	0.35
	Type conversion UINT_TO_WORD	0.35
	Type conversion BOOL_TO_DWORD	0.42
	Type conversion WORD_TO_DWORD	0.42
	Type conversion DINT_TO_DWORD	0.49
	Type conversion UDINT_TO_DWORD	0.49
	Type conversion INT_TO_TIME	8.42
	Type conversion DINT_TO_TIME	7.49
	Type conversion UINT_TO_TIME	7.42
	Type conversion UDINT_TO_TIME	6.7
	Type conversion REAL_TO_TIME	21.49
	Type conversion UDINT_TO_DT	0.7
	Type conversion UDINT_TO_DATE	34.12
	Type conversion UDINT_TO_TOD	14.91
	Type conversion TRUNC_INT	12.42
	Type conversion TRUNC_DINT	21.49
	Type conversion TRUNC_UINT	12.42
	Type conversion TRUNC_UDINT	21.49
	Type conversion W_BCD_TO_INT	13.35
	Type conversion D_BCD_TO_INT	12.42
	Type conversion W_BCD_TO_DINT	18.42
	Type conversion D_BCD_TO_DINT	21.49
	Type conversion INT_TO_W_BCD	12.35
	Type conversion DINT_TO_W_BCD	14.42
	Type conversion INT_TO_D_BCD	13.42
	Type conversion DINT_TO_D_BCD	25.49

1-1 Standard CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed
Arithmetic function	Absolute value ABS_INT	9.35
	Absolute value ABS_DINT	11.49
	Absolute value ABS_REAL	7.49
	Square root (√) SQRT	613.49
	Natural logarithm LN	1700.49
	Common logarithm LOG	1726.49
	Exponent EXP	244.49
	Sine SIN	3856.49
	Cosine COS	3866.49
	Tangent TAN	7606.49
	Arc sine ASIN	3606.49
	Arc cosine ACOS	3706.49
	Arc tangent ATAN	2806.49
	Addition ADD	0.35 + (No. of inputs -1) x 0.21
	Subtraction SUB	0.20
	Multiplication MUL	10.28 + (No. of inputs -1) x 10.14
	Division DIV	10.14
	Division remainder MOD	10.14
	Exponent EXPT	2206.49
	Move MOVE	0.56
	Negation NEG	7.49
Bit string function	Shift left SHL_WORD	9.42
	Shift left SHL_DWORD	9.56
	Shift right SHR_WORD	9.42
	Shift right SHR_DWORD	9.56
	Rotate left ROL_WORD	9.42
	Rotate left ROL_DWORD	10.56
	Rotate right ROR_WORD	9.42
	Rotate right ROR_DWORD	10.56
	Logical product AND	0.14 + (No. of inputs -1) x 0.07
	Logical add OR	
	Exclusive XOR	
	Logical negation NOT	0.35
	Negation NOT_BOOL	0.42
	Negation NOT_WORD	0.35
	Negation NOT_DWORD	0.49

1-1 Standard CPU Instruction Processing Speed Chart

All values in μs.

Language	Name	Processing Speed
Selection/ comparison function	Select SEL_BOOL	1.12
	Select SEL_INT	1.05
	Select SEL_DINT	1.19
	Select SEL_UINT	1.05
	Select SEL_UDINT	1.19
	Select SEL_REAL	1.19
	Select SEL_WORD	1.05
	Select SEL_DWORD	1.19
	Select SEL_TIME	1.19
	Select SEL_STRING	87.47 + 0.64 x (Total numbers of No.of characters)
	Maximum value MAX_INT	0.49 + (No. of inputs -1) x 0.35
	Maximum value MAX_DINT	0.7 + (No. of inputs -1) x 0.49
	Maximum value MAX_UINT	0.49 + (No. of inputs -1) x 0.35
	Maximum value MAX_UDINT	0.7 + (No. of inputs -1) x 0.49
	Maximum value MAX_REAL	14.63 + (No. of inputs -1) x 14.42
	Minimum value MIN_INT	0.49 + (No. of inputs -1) x 0.35
	Minimum value MIN_DINT	0.7 + (No. of inputs -1) x 0.49
	Minimum value MIN_UINT	0.49 + (No. of inputs -1) x 0.35
	Minimum value MIN_UDINT	0.7 + (No. of inputs -1) x 0.49
	Minimum value MIN_REAL	14.63 + (No. of inputs -1) x 14.42
	Limit LIMIT_INT	0.98
	Limit LIMIT_DINT	1.26
	Limit LIMIT_UINT	0.98
	Limit LIMIT_UDINT	1.26
	Limit LIMIT_REAL	29.12
	Comparison (>) GT	0.7 + (No. of inputs -1) x 0.49
	Comparison (≥) GE	0.7 + (No. of inputs -1) x 0.49
	Comparison (=) EQ	0.7 + (No. of inputs -1) x 0.49
	Comparison (≤) LE	0.7 + (No. of inputs -1) x 0.49
	Comparison (<) LT	0.7 + (No. of inputs -1) x 0.49
	Comparison (≠) NE	0.56 + (No. of inputs -1) x 0.35

All values in µs.

Language	Name	Processing Speed
String function	Get Length LEN	53.42 + 0.5 x (No. of characters)
	Get left sub-string LEFT	83.98+ 0.5 x (No. of characters) + 0.14 x (No. of characters to extract)
	Get right sub-string RIGHT	85.98+ 0.5 x (No. of characters) + 0.14 x (No. of characters to extract)
	Get middle sub-string MID	87.98+ 0.5 x (No. of characters) + 0.14 x (No. of characters to extract)
	Concatenate CONCAT	49.19 + 40 x (No. of characters of input) + 0.5 x (Total numbers of input characters) x (No. of characters of input) + 0.14 x (No. of characters of output)
	Insert string INSERT	138.19 + 0.64 x (No. of output characters)
	Delete string DELETE	110.76 + 0.5 x (No. of input characters) + (No. of characters to delete) + 0.14 x (No. of output characters)
	Replace string REPLACE	139.47 + (No. of input characters) + 0.5 x (No. of characters of replacemant - No. of replacement characters) + 0.14 x (No. of input characters)
	Find String FIND	92.63 + 1.5 x (No. of characters of first input + No. of characters of second input)
	Compare String (>) GT_STRING	99.56 + 2.3 x (Position where comparison result is output) + 0.5 x (Total numbers of input characters)
	Compare String (≥) GE_STRING	
	Compare String (=) EQ_STRING	99.56 + 3.5 x (Position where comparison result is output) + 0.5 x (Total numbers of input characters)
	Compare String (≤) LE_STRING	
	Compare String (≤) LT_STRING	99.56 + 2.3 x (Position where comparison result is output) + 0.5 x (Total numbers of input characters)
	Compare String (≠) NE_STRING	
Time type data function	Add time ADD_T_T	0.98
	Add time ADD_TD_T	30.68
	Add time ADD_DT_T	16.47
	Subtract time SUB_T_T	0.48
	Subtract time SUB_D_D	19.19
	Subtract time SUB_TD_T	30.96
	Subtract time SUB_TD_TD	19.19
	Subtract time SUB_DT_T	16.47
	Subtract time SUB_DT_DT	19.19
	Multiply time MUL_T_N	18.91
	Multiply time MUL_T_R	81.05
	Divide time DIV_T_N	15.91
	Divide time DIV_T_R	131.05
	Concatenate time CONCAT_D_D	0.98
	Convert DT to TO DT_TO_TOD	14.91
	Convert DT to DATE DT_TO_DATE	39.98

1-1 Standard CPU Instruction Processing Speed Chart

All values in µs.

Language	Name	Processing Speed
Original function	Set bit SBIT_WORD	0.49
	Set bit SBIT_DWORD	0.63
	Reset bit RBIT_WORD	0.49
	Reset bit RBIT_DWORD	0.63
	Test bit TBIT_WORD	0.56
	Test bit TBIT_DWORD	0.63
	Decode DECODE_WORD	8.35
	Decode DECORD_DWORD	16.49
	Encode ENCODE_WORD	10.35
	Encode ENCODE_DWORD	18.49
	Bit count BITCOUNT_WORD	10.35
	Bit count BITCOUNT_DWORD	13.42
	Convert string to number STR_TO_UINT	63.35 + 6.5 x (No. of input characters)
	Convert number to string UINT_TO_STR	61.77 + 0.14 x (No. of output characters)
	Convert shift-JIS to string SJ_TO_STR	71.98 + 7 x (No. of output characters)
	Convert string to shift-JIS STR_TO_SJ	68.7 + 5 x (No. of input characters)
	Byte length BYTE_LEN	56.35 + 0.5 x (No. of input characters)
	Dead band DBAND_INT	19.7
	Dead band DBAND_DINT	21.91
	Dead band DBAND_REAL	33.91
	Bias BIAS_INT	22.7
	Bias BIAS_DINT	24.91
	Bias BIAS_REAL	28.91
	Step sequence coil SC_COIL/SC	18.98/19.05
	32-bit addition with carry ADC/ADCO	20.19/20.19
	32-bit subtraction with borrow SBB/SBBO	21.19/21.19
	64-bit multiplication MULL/MULU	31.91/31.91
	64-bit division DIVL/DIVU	42.26/42.26
	Shift left 32-bits with carry SLC/SLCO	18.84/17.56
	Shift right 32-bits with carry SRC/SRCO	18.84/17.56
Standard function block	Set reset flip-flop SR	30
	Reset-set flip-flop RS	30
	Rising edge trigger R_TRIG	32
	Falling edge trigger F_TRIG	32
	Up counter CTU	35 to 39
	Down counter CTD	35 to 39
	Up down counter CTUD	40 to 45

All values in μs.

Language	Name	Processing Speed
Standard function block	Pulse TP	33 to 45
	On-delay timer TON	37 to 42
	Off-delay timer TOF	32 to 42
	Real-time clock RTC	39
Original function block	Ring counter RCT	36 to 42
	Integrating timer TMR	34 to 44
	Retriggerable MR	38 to 47
	Open channel M_OPEN	At OPEN request: 210 Response waiting time: 39, Response time: 65 At CLOSE request: 192 Response waiting time: 40, Response time: 74
	Send message M_SEND	At request: 190 + 0.14 x (No. of send words), Response waiting time: 35, Response time: 39
	Receive message M_RECEIVE	At request: 169, Response waiting time: 34, Response time: 86 + 0.14 (No. of receive words)
	Direct read READ_BOOL	At request: 213 + 4 x (No. of receive request bits), Response waiting time: 35, Response time: 57 + 10 x (No. of receive bits)
	Direct read READ_WORD	At request: 211 + 3 x (No. of receive request words), Response waiting time: 35, Response time: 60 + 7 x (No. of send words)
	Direct write WRITE_BOOL	At request: 230 + 7 x (No. of send bits), Response waiting time: 37, Response time: 45
	Direct write WRITE_WORD	At request: 217 + 0.14 x (No. of send words), Response waiting time: 37, Response time: 49
	Remote data read R_READ	At request: 232, Response waiting time: 39, Response time: 93 + 0.14 x (No. of receive words)
	Remote data write R_WRITE	At request: 251 + 0.14 x (No. of send words), Response waiting time: 39, Response time: 78
	File data read F_READ	At request: 168, Response waiting time: 35, Response time: 158 + 0.14 x (No. of receive words)
	File data write F_WRITE	At request: 278 + 0.14 x (No. of send words), Response waiting time: 36, Response time: 59
	Extension test & set EXT_T_S	-
	Sequential file store FFST	81 + 0.14 x (No. of words)
	Sequential file load first FIFO	81 + 0.14 x (No. of words)
	Sequential file load last FILO	81 + 0.14 x (No. of words)
	Filter FILTER_DINT	101 to 217
	Filter FILTER_REAL	281
	Integrate INT_DINT	102 to 216
	Integrate INT_REAL	239
	Differentiate DIF_DINT	84
	Differentiate DIF_REAL	259
	Pulse count PULSE_CNT	46 to 49
	Pulse output PULSE_OUT	45 to 57
	Pulse PWM	44 to 49
	Hardware RTC (Real-time clock) HW_RTC	33 to 75
	Test & set T_S	47
	Change bank BANK_CHG	61 to 154

Processing speed of <SFC elements>

For SFC elements, only steps that are currently active are executed. Consequently, it is difficult to discuss the processing speed of SFC elements using only the worksheet on which the SFC elements are programmed. When it is necessary to grasp processing speed, it is recommended that the processing speed be measured by using an actual machine.

Appendix 2 Setting High-performance CPU Takt Periods

Appendix 2 Setting High-performance CPU Takt Periods

On the SX bus, data is exchanged between the CPU and I/O modules in synchronization with the Takt period. The application program on the CPU module performs 1) I/O data input updating, 2) arithmetic operations, and 3) I/O data output updating in execution units called tasks (default task, periodic task, and event task). These operations are carried out concurrently with data exchange over the SX bus.

This appendix introduces approximation formulas for calculating the scan time for each takt period. Basically, the Takt period of the SX bus is dependent on the system configuration. In a system that requires a task period which is based on the Takt period, the Takt period depends on the number of steps of the application. To obtain an exact execution time, it is necessary to measure it on an actual machine.

<System configuration that depends on the Takt period>

- Number of I/O points
- Number of communication modules
- Number of CPUs
- Number of remote I/O master modules

(1) Approximation formulas for calculating the Takt period from the system configuration data

Takt period T (μ s)

1) 1 CPU + direct I/O configuration: T = Tb [Base time (Tb) based on the number of direct connection input/ output points]

No. of direct connection I/O points (points)	0	32	128	256	512	1024	2048	3072	4096	6144	8192
Base time Tb (μs)	418	504	507	510	556	695	1042	1388	1520	1711	1911

The above numbers are based on the assumption that the ratio of I/O input to I/O output is 1 to 1. The more outputs there are, the longer the base time is, and vice versa. The range of fluctuation is approximately ± 20%.

Note: 1) 0.5 ms tact period is possible under the condition that there is a single CPU, the number of direct connection I/O points is 256 or less, and no communication module is used.
2) The user can select a Takt period from 0.5, 1, 2, 3, ..., 18, 19, and 20 ms. The user should select a value that is obtained by rounding up the corresponding value listed in the above table.

2) Multi-CPU configuration: [No. of CPUs: n]

- T= Tb + 210n (No. of direct connection input/output points: 2048 or less)
- T= Tb + 200n + 190 (No. of direct connection input/output points: more than 2048)

3) Single CPU + remote I/O: [No. of remote I/O master modules: m]

- T= Tb + 250n + 430 (No. of direct connection input/output points: 2048 or less)
- T= Tb + 280n + 730 (No. of direct connection input/output points: more than 2048)

4) Multi-CPU + remote I/O: [No. of CPUs: n, number of remote I/O master modules: m]

- T= Tb + 340n + 200m + 400 (No. of direct connection input/output points: 2048 or less)
- T= Tb + 405n + 260m + 340 (No. of direct connection input/output points: more than 2048)
  remote master module]

Note: 3) 2048 points / 1 line is assumed for remote I/O.

5) When communication modules are added to 1 CPU + direct connection I/O (1): [No. of communication module: p]

- T = Tb + 40p + 250

6) When communication modules are added to configurations ((2) to (4)): [No. of communication modules: p]

- T = (Time calculated for ((2) to (4)) + 85p [when there is no remote master module]
- T = (Time calculated for ((2) to (4)) + 128p [when there is a

(2) Formula for calculating the performance when scanning by the Takt period

Computing time
= [Takt period] - [SBM overhead time (200μs)] - [I/O refresh time] - [POU control time]

• I/O refresh time = (2n + m + 60) μs
[n: number of I/O modules, m: total number of I/O words]

• POU control time
= PG control time + user FB control time + user FCT control time
= (4a + 6b + 7c) μs
[a: No. of PGs, b: No. of user FB calls, c: No. of user FCT calls]

Computing time
= [Takt period] - (2n + m + 60) - (4a + 6b + 7c) μs

No. of program steps = [computing time / single instruction execution time / 1024] k steps
No. of program steps = [computing time / 20.48] k steps [when single instruction execution time = 20ns]
= [computing time / 61.44] k steps [when single instruction execution time = 60ns]

Note: 1) Refer to Appendix 1, "Instruction Processing Speed Chart," for the execution time of the individual instructions.
2) The instruction execution time varies depending on the time required to access the memory to which variables to be processed are assigned. Consequently, it is necessary to add the following access time increments to the instruction execution time according to the number of variables that the instruction accesses.
The memory access times are calculated as follows:

1) I/O memory and standard memory (high speed): Base time (20ns)
2) Standard memory, retained memory, user FB memory, system FB memory, system memory: Add 40ns.
3) Memory in another CPUs access via the processor bus: 3μs

Note: 3) Since processing an operand that has two or more variable elements such as an array, structure, or character string involves multiple memory accesses, it is necessary to take the time increments associated with the multiple memory accesses into consideration when computing the execution time of an instruction having such an operand.

(3) Sample time calculations

1) Single CPU

Scan Time (Takt Time)	System Configuration	POU Control Time (No. of PGs/FBs/FCTs)	Program Executable Time (in 20 ns steps)
0.5ms	CPU... 1 module (communication module disabled) Direct connection I/O... 256 points	68μs (4/4/4)	124μs (6k steps)
1ms	CPU... 1 module Direct connection I/O... 1024 points	136μs (8/8/8)	412μs (20k steps)
2ms	CPU... 1 module Direct connection I/O... 2048 points	480μs (16/32/32)	876μs (42k steps)
	CPU... 1 module Remote I/O ...1 module... 2048 points	480μs (16/32/32)	876μs (42k steps)
	CPU... 1 module Remote I/O ...2 modules: 4096 points Direct connection I/O... 2048 points	480μs (16/32/32)	492μs (24k steps)

2) Multi-CPU configuration

Scan Time (Takt Time)	System Configuration	POU Control Time (No. of PGs/FBs/FCTs)	Program Executable Time (in 20 ns steps)
4ms	CPU... 4 modules Remote I/O ... 2 modules: 4096 points Direct connection I/O: 2048 points	480µs (16/32/32)	2492µs (484k steps = 121k x 4)
	CPU... 4 modules Remote I/O ... 2 modules: 4096 points Direct connection I/O: 1024 points Communication modules... 2 modules	480µs (16/32/32)	2620µs (508k steps = 127k x 4)

(4) Estimation of the Takt periods in the 1 to 1 warm standby duplex system

In the 1 to 1 warm standby system, the Takt time is longer than that in the ordinary multi-CPU system. This is because, in this system, equalized data should be transferred between the operating and waiting CPUs. The estimate expression is described below. In the expression, the large Takt times are used for duplex system Takt time 1 (T_{R1}) and duplex system Tact time 2 (T_{R2}).

Duplex system Takt time 1: T_{R1} [µs]

T_{R1} = Usual Takt time note) + 596 x N + 430 (No. of directly connected I/Os: 2048 or less)
T_{R1} = Usual Takt time note) + 626 x N + 730 (No. of directly connected I/Os: more than 2048)
N: No. of pairs in the duplex system

Note: The usual Takt time is the time found by expression (1). The number of CPUs can be calculated using the number of CPU pairs in the duplex system.

Duplex system Takt time 2: T_{R2} [µs]

T_{R2} = (I/O refresh time) + T_{DMA} + T_{CPY} + 200 [µs]
• I/O refresh time : (2n + m + 60) [É s] [n: No. of I/O modules, m: Total No. of I/O words]
• T_{DMA} = [(No. of SX bus modules excluding CPUs) + (No. of CPU modules) x 2 + (No. of remote master modules x 55) + (No. of total words for all connected/remote I/Os) + 512] x 0.5 [µs]

T_{CPY} = (No. of words for equalized variables in high-speed, standard, retain memory areas) x 0.3
+ (No. of words for user retain variables) x 0.35
+ (No. of edge detection instructions and counter instructions) x 0.3
+ (No. of timer instructions) x 0.45
+ 10 [µs]

Appendix 3 Setting Standard CPU Takt Periods

Appendix 3 Setting Standard CPU Takt Periods

On the SX bus, data is exchanged between the CPU module and I/O module in synchronization with the Takt period. For the application program on the CPU module, I/O data input update, arithmetical operations, and I/O data output update are performed in each task (process unit) in parallel with data exchange on the SX bus.
The standard CPU divides system software in one Takt period into the processes executed in every Takt period as well as those executed whenever an application is executed or a default task has been done. The loader process is performed in one excessive Takt period. The estimate expression of the scan time for each Takt period is shown below. The Takt period depends on the system configuration. Execution of system software also depends on the Takt period. Thus, the executable Takt periods must be set by system software.

<System configuration components depending on the Takt period>

- No. of I/O points
- No. of remote I/O modules
- No. of stations simultaneously issuing loader command

The Takt period can be selected among 1, 2, 3, ...9, 10ms. (1) The Takt period calculated is based on SX bus performance (No. of I/O modules) and (2) the necessary Takt period based on the run time of system software, whichever is larger, is determined by truncation

(1) The Takt period calculated is based on SX bus performance (No. of I/O modules)

1) One CPU + Connected I/O : T = Tb

No. of connected I/O points	0	32	128	256	512	1024	2048	3072	4096	6144	8192
Base time Tb (μs)	418	504	507	510	556	695	1042	1388	1520	1711	1911

The times listed above are calculated under the condition that the ratio of I/O input to output = 1 : 1. The time increases with an increased number of outputs and decreases with a reduced number of outputs. Its fluctuation ranges from + 20% to -20%.

2) One CPU + remote I/O mater: [No. f remote I/O masters]

- T = Tb + 250 m + 430 (No. of direct connection input/ output points: 2048 or less)
- T = Tb + 280 m + 730 (No. of direct connection input/ output points: more than 2048)

Note: 2048 points/1 line is assumed for remote I/O.

(2) The necessary Takt period based on the run time of system software

System software processes I/O transmission, tasks, and loader commands. Any of these processes should be done in one Takt period. Based on the number of stations simultaneously issuing the loader commands, recommended preset times and calculation times (the application program processing time) for the Takt periods are shown below.

Recommended Takt time (ms)	1	2	3	4	5	6	7	8
No. of stations simultaneously issuing loader commands	1	4	16	27	27	27	27	27
Calculation time (μs)	409	1209	2009	2809	3609	4409	5209	6009

Note: The stations simultaneously issuing loader commands include the loader, PODs, and the modules associated with message-related instructions (if the module has two ports, they are both counted.)

Key-point:

- To execute the task processes in synchronization with the Takt periods, the application program should be written so that it does not use more than the calculation time listed below.
- Compared with the recommended Takt time, even when the number of stations simultaneously issuing the loader commands is smaller, the calculation time does not vary. (The responsibility to the loader commands improves.)
  Compared with the recommended Takt time, when the number of stations simultaneously issuing the loader commands is larger, the calculation time is reduced. (The responsibility to the loader commands deteriorates.)

Appendix 4 Calculating the Size of Arrays and Structures

Appendix 4 Calculating the Size of Arrays and Structures

How to calculate the size of arrays and structures is shown below.

Size of Data Occupied by the Data Types

	Data Type	Occupied words
Bit	BOOL	1 word if the bit string is 16 bits or less. If the bit string is longer than 16 bits, 1 word is occupied for every 16 bits. Another word is occupied for the remaining bit fragment.
Word	INT, UINT, WORD	1 word
2 words	DINT, UDINT, REAL, DWORD, TIME, DATE, TOD, DT	2 words

Boundary consideration

The starting address of an array or structure assigned a 2-word boundary. The boundary of the individual data item varies according to the data type established before the data is defined.

		Preceding Stage Data		
		Bit data	Word data	Double word data
Current	Bit data	Within same word	Word boundary	Word boundary
	Word data	Word boundary	Word boundary	Word boundary
	Double word data	Double word boundary	Double word boundary	Double word boundary

Example 1)

```
TYPE                                  (No. of occupied words)
    a1    : BOOL ;   ------------►        1/16
    a2    : BOOL ;   ------------►        1/16
    a3    : INT ;    ------------►          1
    a4    : DINT ;   ------------►          2
    a5    : WORD ;   ------------►          1
    a6    : BOOL ;   ------------►        1/16
    a7    : I NT ;   ------------►          1
    a8    : DWORD ;  ------------►          2
END_TYPE
```

Total 7 + 3/16 = 8 words

Example 2)

```
TYPE                                  (No. of occupied words)
    b1    : BOOL ;   ------------►        1/16
    b2    : BOOL ;   ------------►        1/16
    b3    : UINT ;   ------------►          1
    b4    : TIME ;   ------------►          2
    b5    : WORD ;   ------------►          1
    b6    : INT ;    ------------►          1
    b7    : BOOL ;   ------------►        1/16
    b8    : WORD ;   ------------►          1
```

(Note)
Total 6 + 3/16→ 7 + (1) = 8 words

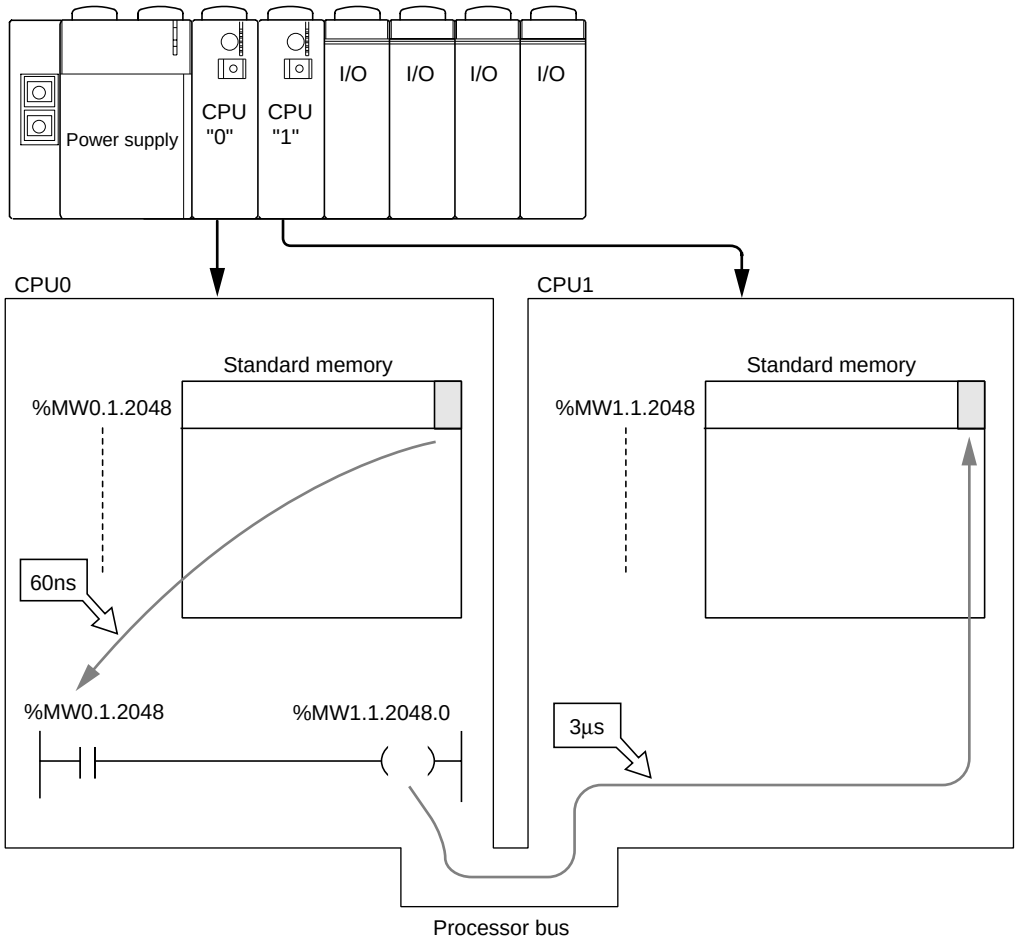
Note: The structure is adjusted so that the total of the sizes is an even-number word.

Appendix 5 Accessing the Processor Bus

Note: The standard CPU does not support processor bus access.

Appendix 5 Accessing the Processor Bus

The processor bus can be used to read and write memory between CPUs in the multi-CPU system and to read and write memory between the CPU and the P/PE-link memory.



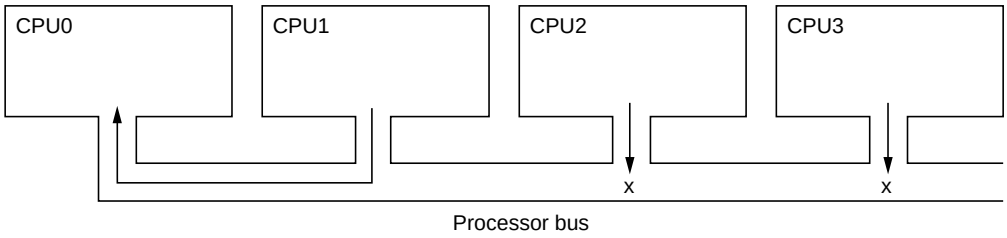
<Access time to each memory from an application (CPU)>
The access time to each memory in or out of the CPU is shown below.

Accessed memory	Access time (/word)
High-speed memory in self-CPU (%MW1.0 to %MW1.2047)	20ns
Memory other than those in self-CPU	60ns
I/O area (I/Q)	20ns
Memory in any other CPU in a multi-CPU system (Note 1)	3µs (Note 2)
P/PE-link memory	3µs (Note 2)

Note: 1) The high-speed memory area cannot be accessed in any other CPU.
2) The access time through the processor bus.

<Considerations in reading /writing memory through the processor bus>

The processor bus cannot be accessed simultaneously by more than one CPU. If more than one CPU accesses the processor bus simultaneously, it can be sequentially used by the CPUs starting from the one with the highest priority assigned. The CPU with a lower priority waits for a long time. Any such delay affects the CPU processing speed.

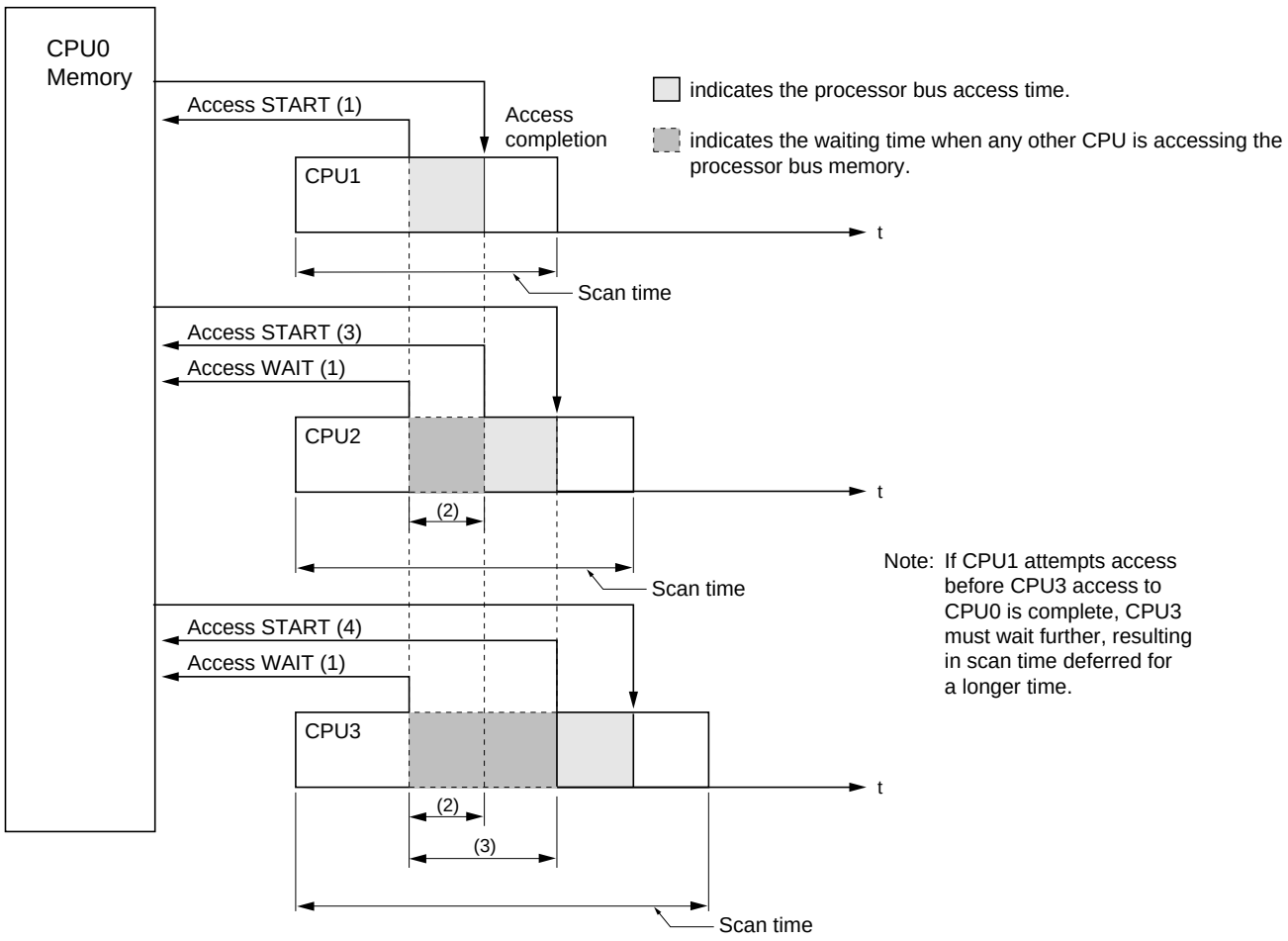


Note: The priority is determined in the ascending order of numbers, with the highest priority assigned to the smallest one.

(Example)

In the example below, a multi-CPU system consisting of four CPUs is shown. With reference to this example, the relationship between processor bus access and the CPU scan time is described.

1) When CPU1, CPU2, and CPU3 simultaneously access CPU0 memory, the first CPU1 with the highest priority has initial access to it. CPU2 and CPU3 are kept waiting.
2) The scan times for CPU2 and CPU3 are delayed by the CPU1 access time.
3) When CPU1 access is complete, the CPU2 with the secondly highest priority tries to gain access to CPU0 memory. CPU3 must wait further.
4) When CPU2 access is complete, CPU3 starts gaining access to memory. (Note)



Key-point:

- In the multi-CPU system and P/PE-link system, design the system to minimize the number of accesses to the processor bus. The recommended number of accesses is 128/ms.

Appendix 6 List of Reserved Words

Appendix 6 List of Reserved Words

Reserved words cannot be used for variable and data names. These reserved words include the following, such as function names and function block names.

	Upper case	Lower case
Symbol	)	
A	ABS ABS(ACOS ACOS(ADD ADD(AND AND(ANDN ANDN(ANY ANY_BIT ANY_DATE ANY_INT ANY_NUM ANY_REAL ARRAY ASIN AT ATAN	abs abs(acos acos(add add(and and(andn andn(any any_bit any_date any_int any_num any_real array asin at atan
B	BOOL BOOL# BOOL8 BY BYTE BYTE#	bool bool# bool8 by byte byte#
C	CAL CALC CALCN CASE CONCAT CONCAT(CONFIGURATION CONSTANT COS COS(CSV	cal calc calcn case concat concat(configuration constant cos cos(csv
D	DATE DATE# DATE_AND_TIME DATE_AND_TIME# DELETE DELETE(DEVICE DINT DINT# DIV DIV(DIV_T_AN DIV_T_AN DO DT DT# DWORD DWORD#	date date# date_and_time date_and_time# delete delete(device dint dint# div div(div_t_an div_t_an do dt dt# dword dword#
E	ELEMENTARY ELSE ELSEIF EN END_CASE END_FOR END_IF END_PROGRAM END_REPEAT END_STRUCT END_TYPE END_VAR END_WHILE ENO ENUM EQ EQ(EXIT EXP EXP(EXPT EXPT(	elementary else elseif en end_case end_for end_if end_program end_repeat end_struct end_type end_var end_while eno enum eq eq(exit exp exp(expt expt(
F	FALSE FIND FIND(FOR FUNCTION FUNCTION_BLOCK	false find find(for function function_block
G	GE GE(GT GT(	ge ge(gt gt(
I	IF INSERT INSERT(INT INT#	if insert insert(int int#
J	JMP JMPC JMPCN	jmp jmpc jmpcn
L	LD LDN LE LE(LEFT LEFT(LEN LEN(LIMIT LIMIT(LINT LINT# LN LN(LOG LOG(LREAL LREAL# LT LT(LWORD LWORD#	ld ldn le le(left left(len len(limit limit(lint lint# ln ln(log log(lreal lreal# lt lt(lword lword#

	Upper case	Lower case
M	MAX	max
	MAX(	max(
	MID	mid
	MID(	mid(
	MIN	min
	MIN(	min(
	MOD	mod
	MOD(	mod(
	MOVE	move
	MOVE(	move(
	MUL	mul
	MUL(	mul(
	MUL_T_AN	mul_t_an
	MUL_T_AN(	mul_t_an(
	MUX	mux
	MUX(	mux(
N	NE	ne
	NE(	ne(
	NOT	not
	NOT(	not(
O	OF	of
	OR	or
	OR(	or(
	ORN	orn
	ORN(	orn(
P	PDD	pdd
	PROGRAM	program
R	R	r
	REAL	real
	REAL#	real#
	REPEAT	repeat
	REPLACE	replace
	REPLACE(	replace(
	RESOURSE	resourse
	RET	ret
	RETAIN	retain
	RETC	retc
	RETCN	retcn
	RETURN	return
	RIGHT	right
	RIGHT(	right(
	ROL	rol
	ROL(	rol(
	ROR	ror
	ROR(	ror(

	Upper case	Lower case
S	S	s
	SEL	sel
	SEL(	sel(
	SHL	shl
	SHL(	shl(
	SHR	shr
	SHR(	shr(
	SIN	sin
	SIN(	sin(
	SINT	sint
	SINT#	sint#
	SQRT	sqrt
	SQRT(	sqrt(
	ST	st
	STN	stn
	STRING	string
	STRING#	string#
	STRING_HANDLE	string_handle
	STRUCT	struct
	SUB	sub
	SUB(	sub(
	SUB_T_T	sub_t_t
	SUB_T_T(	sub_t_t(
T	T#	t#
	TABLE	table
	TASK	task
	THEN	then
	TIME	time
	TIME#	time#
	TIME_OF_DAY	time_of_day
	TIME_OF_DAY#	time_of_day#
	TO	to
	TOD	tod
	TOD#	tod#
	TRUE	true
	TYPE	type

	Upper case	Lower case
U	UDINT UDINT# UINT UINT# ULINT ULINT# UNTIL USINT USINT#	udint udint# uint uint# ulint ulint# until usint usint#
V	VAR VAR_DUPLICATE VAR_EXTERNAL VAR_EXTERNAL_FB VAR_EXTERNAL_PG VAR_GLOBAL VAR_GLOBAL_FB VAR_GLOBAL_PG VAR_IN_OUT VAR_INPUT VAR_OUTPUT	var var_duplicate var_external var_external_fb var_external_pg var_global var_global_fb var_global_pg var_in_out var_input var_output
W	WEIGHT WEIGHT# WHILE WORD WORD#	weight weight# while word word#
X	XOR XOR(XORN XORN(	xor xor(xorn xorn(

Fuji Electric FA Components & Systems Co., Ltd.

Gate City Ohsaki, East Tower, 11-2, Osaki 1-chome, Shinagawa-ku, Tokyo, 141-0032, Japan
Phone: +81-3-5435-7135 ~ 8
Fax: +81-3-5435-7456 ~ 9
URL http://www.fujielectric.co.jp/fcs/eng/