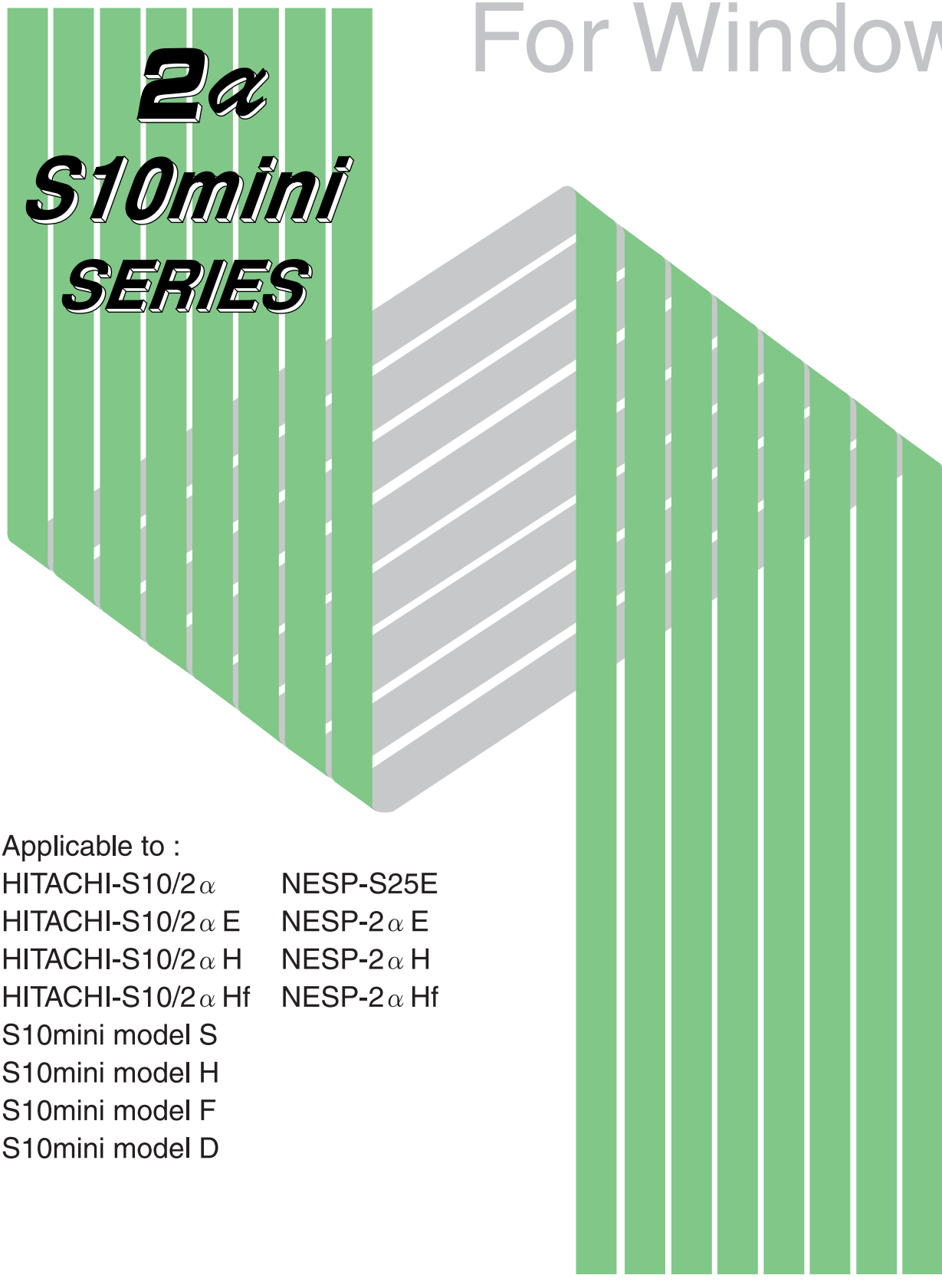
HITACHI
S10α SERIES

SOFTWARE MANUAL
PROGRAMMING

HI-FLOW For Windows®

2α S10mini SERIES



Applicable to :

HITACHI-S10/2α	NESP-S25E
HITACHI-S10/2α E	NESP-2α E
HITACHI-S10/2α H	NESP-2α H
HITACHI-S10/2α Hf	NESP-2α Hf
S10mini model S	
S10mini model H	
S10mini model F	
S10mini model D	

SAE-3-122 (D)

HITACHI

NOTE

All information in this manual is based on the latest product information available at the time of printing. Hitachi has reviewed the accuracy of this manual, but assumes no responsibility for any omissions or errors which may appear. The design of the product is under constant review and, while every effort is made to keep this manual up to date, the right is reserved to change specifications and equipment at any time without prior notice.

PROHIBITION

These products should not be used for medical, power supply, nuclear, water supply, drainage plants, traffic control, military, space, nor disaster prevention equipment.

Diversion and/or resale of these products without this manual is prohibited.

TRADEMARKS

HITACHI-S10/2α, S10/4α and PSEα are registered trademarks of Hitachi, Ltd.

FIRST EDITION, JUNE, 1997, SAE - 3 - 122 (A) (out of print)
SECOND EDITION, OCTOBER, 2000, SAE - 3 - 122 (C)
THIRD EDITION, SEPTEMBER, 2003, SAE - 3 - 122 (D)

BI-KB-TN<IC-NS> (FL-MW20)

LIMITED WARRANTY

Hitachi, Ltd., warrants its products to be manufactured in accordance with published specifications and free from defects in materials and/or workmanship.

Hitachi, Ltd., warrants its products against defects in parts and workmanship for one full year from date of purchase.

HITACHI, LTD., MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED EXCEPT AS PROVIDED HEREIN, INCLUDING WITHOUT LIMITATION THEREOF, WARRANTIES AS TO MARKETABILITY FOR A PARTICULAR PURPOSE OF USE, OR AGAINST INFRINGEMENT OF ANY PATENT. IN NO EVENT SHALL HITACHI BE LIABLE FOR ANY DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY NATURE, OR COSTS, CHARGES, LOSSES OR EXPENSES RESULTING FROM ANY DEFECTIVE PRODUCT OR THE USE OF ANY PRODUCT.

SOFTWARE UP–TO DATE POLICY

Hitachi, Ltd., constantly reviews its software so as to incorporate the latest technology. Hitachi reserves the right to make changes to any software to improve reliability, function, or design. Hitachi cannot be held responsible for any errors in its software.

SAFETY PRECAUTIONS

- Read this manual thoroughly and follow all the safety precautions and instructions given in this manual before operations such as system configuration and program creation.
- Keep this manual handy so that you can refer to it any time you want.
- If you have any question concerning any part of this manual, contact your nearest Hitachi branch office or service engineer.
- Hitachi will not be responsible for any accident or failure resulting from your operation in any manner not described in this manual.
- Hitachi will not be responsible for any accident or failure resulting from modification of software provided by Hitachi.
- Hitachi will not be responsible for reliability of software not provided by Hitachi.
- Make it a rule to back up every file. Any trouble on the file unit, power failure during file access or incorrect operation may destroy some of the files you have stored. To prevent data destruction and loss, make file backup a routine task.
- Furnish protective circuits externally and make a system design in a way that ensures safety in system operations and provides adequate safeguards to prevent personal injury and death and serious property damage even if the product should become faulty or malfunction or if an employed program is defective.
- If an emergency stop circuit, interlock circuit, or similar circuit is to be formulated, it must be positioned external to the programmable controller. If you do not observe this precaution, equipment damage or accident may occur when the programmable controller becomes defective.
- Before changing the program, generating a forced output, or performing the RUN, STOP, or like procedure during an operation, thoroughly verify the safety because the use of an incorrect procedure may cause equipment damage or other accident.

"RUN/STOP" SWITCH CAUTION

The "RUN/STOP" switch only stops execution of the ladder logic program or HI-FLOW program. Digital and analog outputs are left in the active state when execution stops, unless the optional rungs described in the CPU manual have been added. The "RUN/STOP" switch does not affect the operation of C-language or FA-BASIC language programs. Outputs can still be produced in response to C-language or FA-BASIC programs, or by the action of programmers typing in commands in these languages, while the "RUN/STOP" switch is in the "STOP" position.

DO NOT DEPEND ON THE STOP SWITCH TO STOP MOVING PARTS OR TO PREVENT UNEXPECTED MOTION OR ENERGIZATION. USE HARDWIRED SAFETY DISCONNECT AND LOCK OUT POWER AND CONTROL VOLTAGES BEFORE WORKING ON ELECTRICAL CIRCUITS OR PARTS THAT CAN MOVE.

PREFACE

Flowchart type programming language HI-FLOW was developed to allow the user to easily code programs for the programmable controller.

This manual describes instructions for programming in HI-FLOW. For ladder programs, refer to the following manual.

<Related manual>

SOFTWARE MANUAL PROGRAMMING LADDER CHART For Windows® (Manual number SAE-3-121)

See the following list when you use the NESP (Nissan Electronic Sequence Processor) series.

【HITACHI-S10α series】		【NESP series】
HITACHI-S10/2α	·····	NESP-S25E
HITACHI-S10/2αE	·····	NESP-2αE
HITACHI-S10/2αH	·····	NESP-2αH
HITACHI-S10/2αHf	·····	NESP-2αHf

<Trademarks>

- Microsoft® Windows® operating system, Microsoft® Windows® 95 operating system, Microsoft® Windows® 98 operating system, Microsoft® Windows® 2000 operating system, Microsoft® Windows® XP operating system are registered trademarks of Microsoft Corporation in the United States and/or other countries.
- Ethernet is a registered trademark of Xerox Corp.

Other product names written in this manual are the trademarks of each manufacturer.

CONTENTS

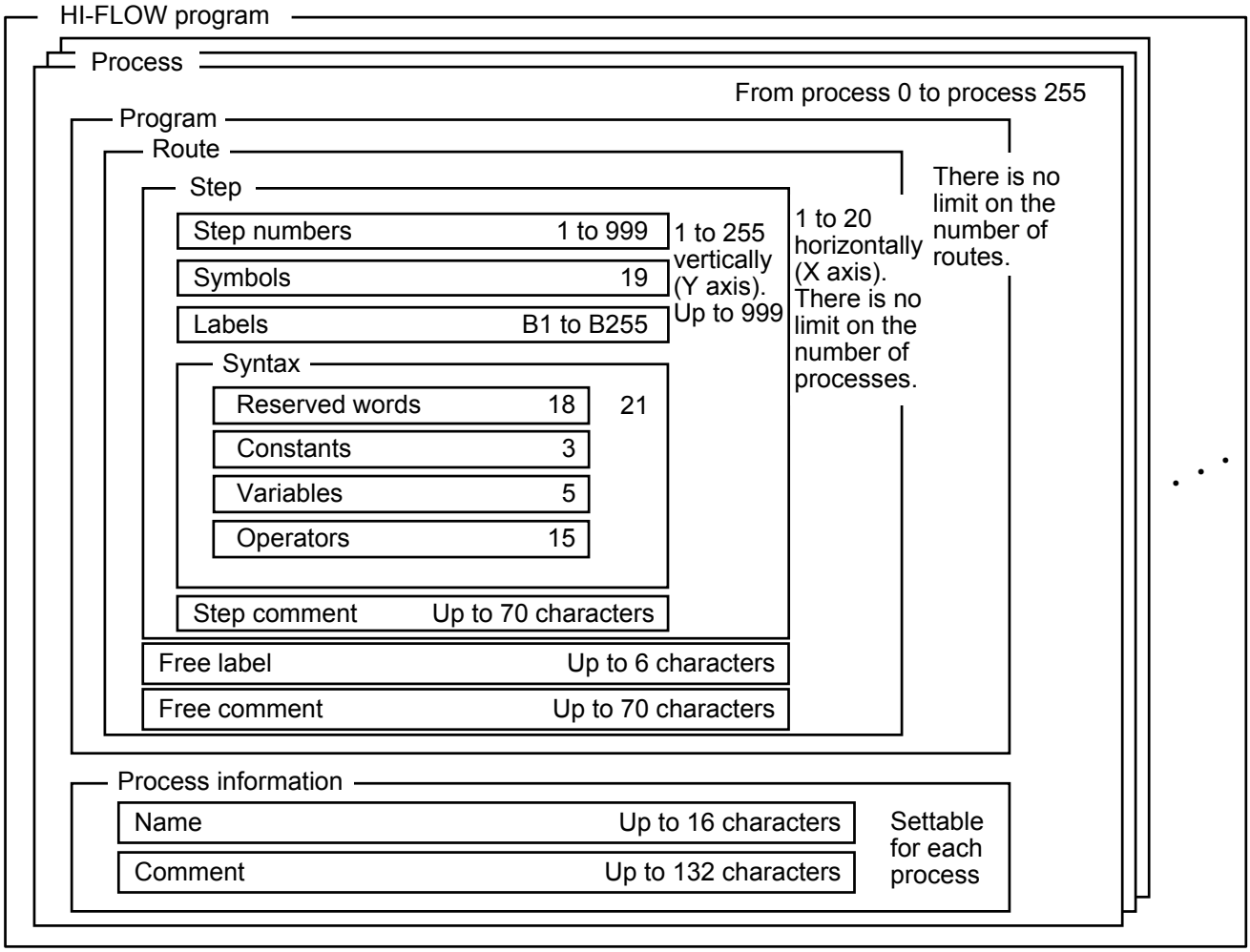
THIS PAGE INTENTIONALLY LEFT BLANK.

1 CONFIGURATION OF HI-FLOW PROGRAMS

1 CONFIGURATION OF HI-FLOW PROGRAMS

This manual describes the specifications of the new HI-FLOW language. When creating actual programs, refer to this manual at the necessary times.

HI-FLOW programs you create consist of the following components:



2 HOW TO USE THIS MANUAL

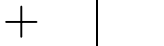
2.1 Overview

This manual is prepared according to the configuration shown in Chapter 1. The following table shows relationships between individual items and their corresponding chapters or sections and pages.

Item	Chapter or section	Page
• Process	3	9
• Program	3.2	15
• Route		15
• Step		19
• Step number		20
• Symbol		20
• Label		23
• Syntax		23
• Reserved word		24
• Constant		24
• Variable		24
• Operator		26
• Step comment		26
• Free label		27
• Free comment		27
• Process information	3.3	28
• Name		28
• Comment		28

2.2 Outline of the Syntax

After "2.1 Overview," this manual describes details of the syntax for each function. The following table shows relationships between functions and their corresponding chapters or sections and pages.

Item	Symbol	Chapter or section	Page
Explanation of syntax		4	29
• Process Start		4.1	30
Process End			
• STP			30
• RST			31
• CLR			32
• ACT			32
• Route Start		4.2	34
Route End			
• Wait		4.3	35
• Condition expression			35
• Timer			35
• Output bit			35
• Wait timer			35
• Box		4.4	37
• Assignment expression			37
• ON			38
• OFF			39
• Parallel timer			39
• TUP			40
• TRS			41
• Control Box		4.5	42
• ACT			42
• RST			43
• STP			44
• CLR			44
• Repeat Start		4.6	46
Repeat End			
• If		4.7	47
• Jump		4.8	49
• Escape		4.9	50
• Parallel Start		4.10	51
Parallel End			
• Select		4.11	52
Wait in Selective Branching			
Select End		4.12	53
• Multi-entry			
• Call		4.13	54
• Function		4.14	55
• Wait with Previous State Cleared		4.15	55

2.3 Application Instructions

HI-FLOW supports application instructions of functions as well as ladder diagrams. The following table shows application instructions and their functions.

Major class	Class	Symbol	Function	Page
Arithmetic operation instructions	Addition	ADD	S+D → R	63
	Subtraction	SUB	S - D → R	64
	+1	INC	S+1 → S	65
	-1	DEC	S - 1 → S	66
	Multiplication	MUL	S*D → R	67
	Division	DIV	S/D → R	68
	Residue	MOD	Residual of S/D → R	69
	Scale conversion	SCL	S*D1/D2 → R	70
Logical operation instructions	Logical Product	AND	S AND D → R	71
	Logical Sum	OR	S OR D → R	72
	Exclusive Logical Sum	EOR	S EOR D → R	73
	Not	NOT	NOT S → R	74
Compare operation instructions	=	EQU	True or false of S = D → R	75
	< >	NEQ	True or false of S < > D → R	76
	>	GT	True or false of S > D → R	77
	> =	GE	True or false of S >= D → R	78
	<	LT	True or false of S < D → R	79
	< =	LE	True or false of S <= D → R	80
	Test	TST	Sign of S → R	81
Data move instructions	Move	MOV	S → D	82
	Block Move	MOM	S ~ Sn → D ~ Dn	83
	Exchange	EXC	S ↔ D	84
	FIFO Write	PSH	S → D (FIFO table)	85
	FIFO Read	POP	S (FIFO table) → D	86
	Address Set	AST	S address → D	87
	Search	SCH	S = D(n) → n is set in R.	88
Data conversion instructions	BIN-BCD	BTD	BIN → BCD S ——→ R	89
	BCD-BIN	DTB	BCD → BIN S ——→ R	90
	BIN-7SEG	SEG	BIN → 7 segments S ——→ R	91

Major class	Class	Symbol	Function	Page
Data conversion instructions	BIN-ASC	ASP	BIN → ASCII (packed, unpacked)	92
		ASU	S → (R, R+1), (R, R+1, R+2, R+3)	93
	ASC-BIN	APB	ASCII (packed, unpacked) → BIN	94
		AUB	(S, S+1), (S, S+1, S+2, S+3) → R	95
	Absolute Value	ABS	| S | → R	96
	+/-	NEG	_S → R	97
	Decode	DCD	2^11 to 2^15 of S → Bit 2^n of R is turned on.	98
	Encode	ECD	Number of the first turned-on bit of S → 2^11 to 2^15 of R	99
Shift instructions	Logical Right Shift	LSR	S Logical Right Shift D → R	100
	Logical Left Shift	LSL	S Logical Left Shift D → R	101
	Arithmetic Right Shift	ASR	S Arithmetic Right Shift D → R	102
	Arithmetic Left Shift	ASL	S Arithmetic Left Shift D → R	103
Rotate instructions	Right Rotate	ROR	S Right Rotate R	104
	Left Rotate	ROL	S Lift Rotate R	105
Function process instructions	Limiter	LIM		106
	Dead Band	BND		107
	Dead Zone	ZON		108
	Root	ROT		109
	MAX	MAX		110
	MIN	MIN		111
Special instructions	Clear	XCLR YCLR GCLR RCLR KCLR TCLR UCLR CCLR VCLR ECLR FCLR JCLR QCLR HHCLR		112

THIS PAGE INTENTIONALLY LEFT BLANK.

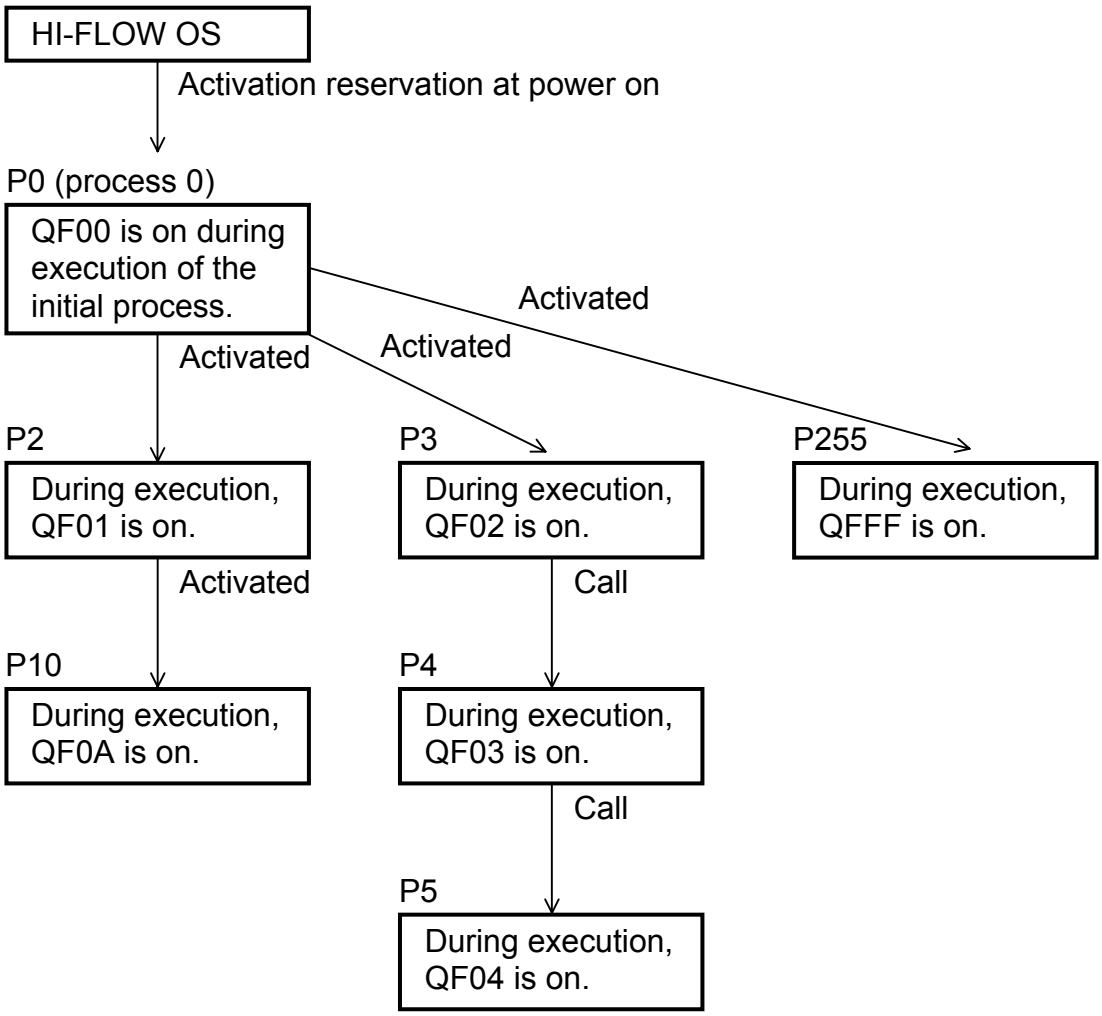
3 PROCESSES

3.1 About the Process

A process is the largest configuration unit in a HI-FLOW program. It starts with Process Start (●) and ends with Process End (●). A process consists of programs having at least one route and process information. You can control production lines with one or more processes created by function.

A process (P0 to P255) is recognized with "P+process number (in decimal)." P0 is called the initial process. It is reserved for activation by the HI-FLOW operating system that controls execution of the HI-FLOW program when the PCs is turned on. After the initial process is activated, processes P1 to P255 can be controlled.

During process execution, the specified PI/O register is turned on, enabling process execution to be monitored. (See the description of standard QF00 to QFFF and the system bit assignment command.)



Process States

Each process in the PCs is in one of nine states.

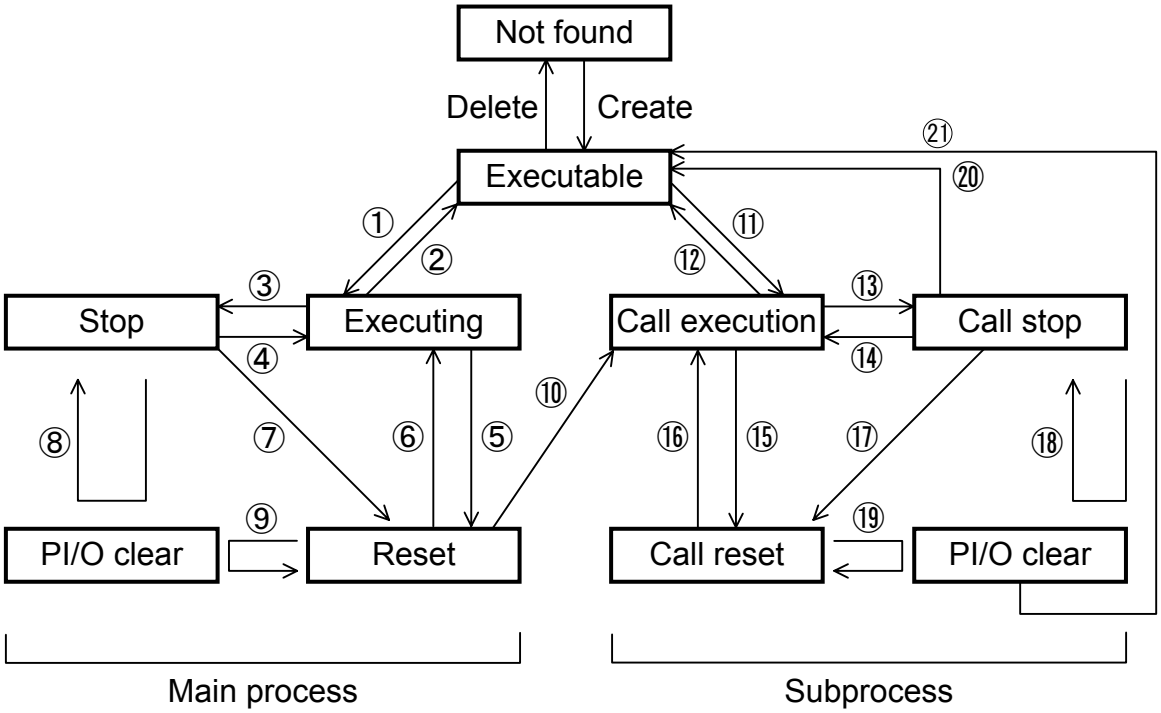
State	Description
Not found	There is no HI-FLOW process.
Executable	A HI-FLOW process can be operated when it is activated.
Executing	A HI-FLOW process was with ACT by another process and is being executed.
Stop	A HI-FLOW process is being stopped in the middle because some conditions were satisfied. The process information and PI/O values are held. For the timer, you can specify whether you hold the timer value or continue measurement.
Reset	Execution of a HI-FLOW process was canceled because some conditions were satisfied, and it is stopping at the process start point. The process information is initialized. The PI/O values are held. For the timer, you can specify whether you reset the timer at predetermined reset time or continue measurement.
Clear	When some conditions are satisfied while a HI-FLOW process is being stopped, reset, call-stopped, or callreset, the bit-type PI/O (in the ON statement or parallel timer) being used in the process is cleared to 0.
Call execution	A HI-FLOW process was subroutine-called by another process and is being executed.
Call stop	A HI-FLOW process is being stopped at a point in it because some conditions were satisfied during execution of a call. The process information and PI/O values are held. For the timer, you can specify whether you hold the timer value or continue measurement.
Call reset	Execution of a HI-FLOW process was canceled because some conditions were satisfied during execution of a call, and it is stopping at the process start point. The process information is initialized. The PI/O values are held. For the timer, you can specify whether you reset the timer at reset time or continue measurement.

The process enters into the stop and reset state once conditions are satisfied. Even if these conditions are released, the process continues the states. The clear state is entered each time the conditions are satisfied.

Process states change

There are nine process states. The following figure shows how these states are changed by what (circled numbers in the figure).

[State change]



① Control box ACT (■)
② Escape (✕)
③ Process Start STP (●), Control Box STP (■)
④ Process Start ACT (●), Control Box ACT (■)
⑤ Process Start RST (●), Control Box RST (■)
⑥ Process Start ACT (●), Control Box ACT (■)
⑦ Process Start RST (●), Control Box RST (■)
⑧ Process Start CLR (●), Control Box CLR (■)
⑨ Process Start CLR (●), Control Box CLR (■)
⑩ Process Call (▭)
⑪ Process Call (▭)
⑫ Process End (●), Escape (✕)
Control Box RST to calling process (■)
Process Start RST to calling process (●)
⑬ Process Start STP (●)
Control Box STP to calling process (■)
Process Start STP to calling process (●)
⑭ Process Start ACT (●)
Control Box ACT to calling process (■)
Process Start ACT to calling process (●)
⑮ Process Start RST (●)
⑯ Process Start ACT (●)
⑰ Process Start RST (●)
⑱ Process Start CLR (●)
Control Box CLR to calling process (■)
Process Start CLR to calling process (●)
⑲ Process Start CLR (●)
⑳ Control Box RST to calling process (■)
Process Start RST to calling process (●)
㉑ Control Box RST to calling process (■)
Process Start RST to calling process (●)

When a process is changed to the executing or call execution state, you can specify either of two activation methods: master reset or zone. When you omit specification, zone activation is assumed.

When a process is changed to the process end (●), escape (✕), or executable state, the handing of the PI/O values (whether they are held or cleared to 0) and the timer (whether the timer is forcibly timed out or reset or measurement is continued) depends on the activation method.

Relationships between the setting of the key switch on the PCs and the states of processes

Below is an explanation of how processes in the PCs change according to the setting of the key switch on the PCs and when the PCs are recovered from a power failure. HI-FLOW does not recognize the difference in the setting of the key switch between RUN and SIM RUN. With the key switch set to RUN or SIM RUN, the states of processes depend on the operation of the PCs.

(a) When the PCs are recovered from a power failure (when the key switch is reset)
When the PCs are recovered from a power failure, all processes in the PCs are initialized.
<At initialization>
- The processes become executable.
- The timers are stopped.
- PI/Os are turned off. (The states of DW, FW, K, and KW remain unchanged.)

Process 0 (initial process) is reserved for activation. It is executed when the key switch on the PCs is set to RUN next time.

(b) When the key switch is set to STOP
With the key switch on the PCs set to STOP, the states of the processes remain unchanged even when the state of a PI/O or timer in the PCs changes.

(c) When the key switch is set to RUN or SIM RUN
With the key switch on the PCs set to RUN or SIM RUN, the states of the processes change accordingly when the state of a PI/O and/or timer in the PCs changes.

(d) When the key switch is changed from STOP to RUN or SIM RUN
When the key switch on the PCs is changed from STOP to RUN or SIM RUN, the processes change from the state in (b) to that in (c). If the PCs are in the state immediately after power is recovered, process 0 is executed. You can also set the processes in the state in (c) when the PCs are not in the state immediately after power is recovered. To do this, specify the same effect for HI-FLOW as after the PCs are recovered from a power failure. (See the system edition commands.)

(e) When key switch is changed from RUN or SIM RUN to STOP
When the key switch on the PCs is changed from RUN or SIM RUN to STOP, the processes changed from the state in (c) to that in (b). At the same time, timers WT and PT stop measurement.

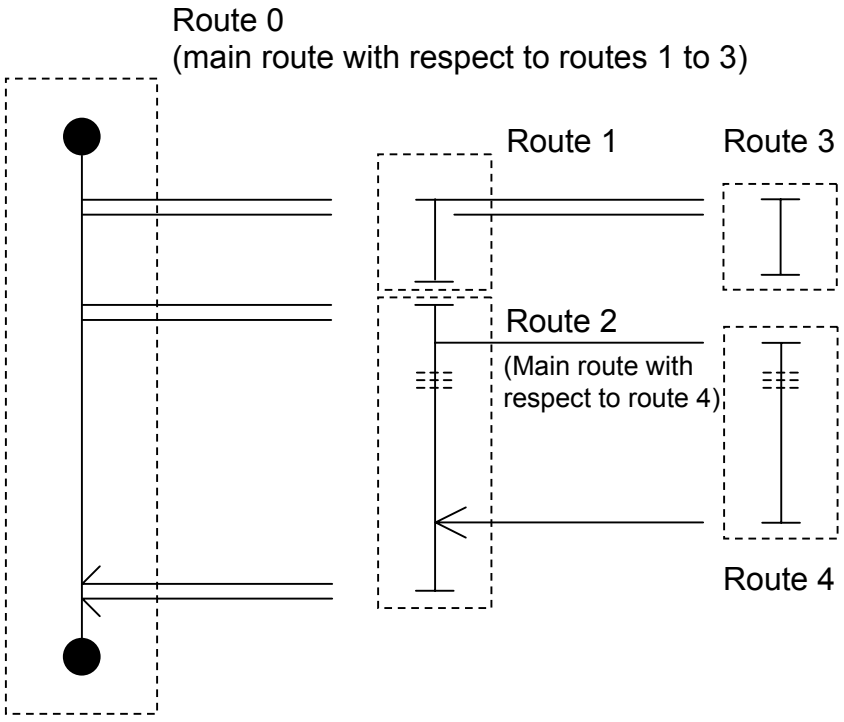
3.2 Program

A process consists of programs and process information. Programs actually control production lines. A program consists of one or more routes.

Route

A route which flows vertically starts with Process Start (●) or Route Start (⊤) and ends with Process End (●) or Route end (⊥). A route is the minimum unit of a process program. With multiple routes, a process can be subject to synchronous or selective processing. A branch occurs at a main route. Branching routes are called subroutes. A subroute branches at Parallel Start (⊨) or Select (⊢). Subroutines are joined at Parallel End (⊨) or Select End (⊭).

You do not need to recognize routes with route numbers, so these numbers are managed only by the system.



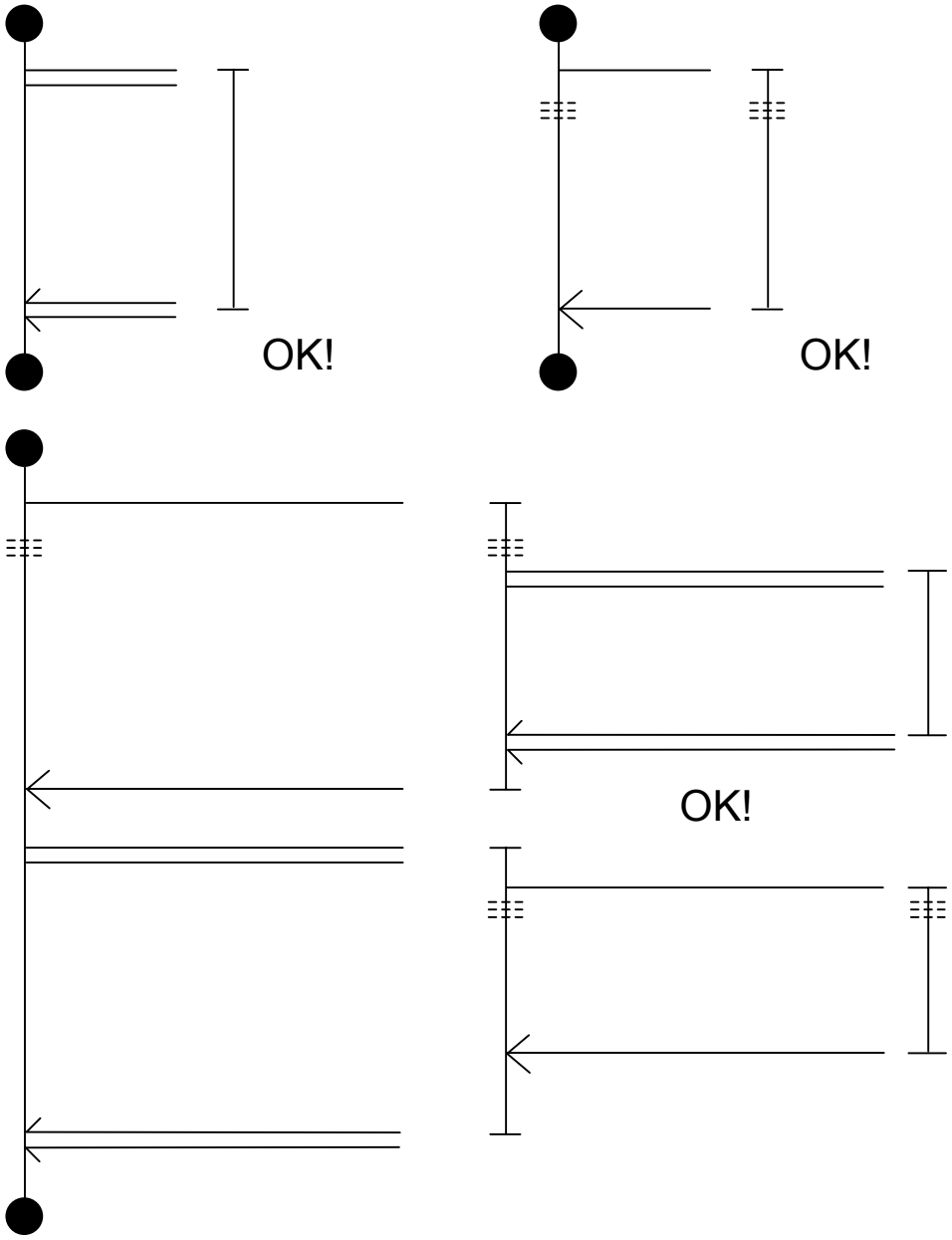
Synchronous routes do not always need to be joined. When they are not joined, the effect of the main route is just to activate routes. A selection route must be joined with another route even if there is an unconditional branch.

(1) Use of both the synchronization syntax and selection syntax

When the synchronization syntax and selection syntax are programmed independently, there is no problem. When using them together, however, care must be taken.

(a) When the same route is used as a route at which a branch starts and a route at which routes are joined

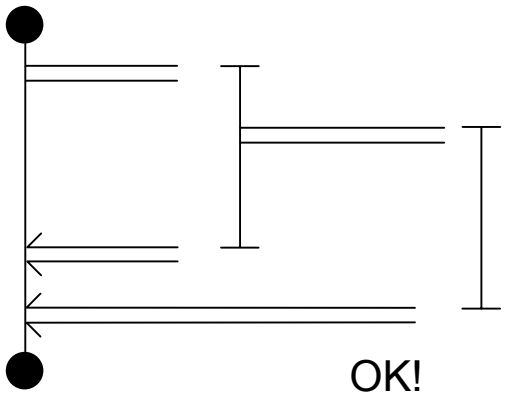
For both synchronization and selection routes, all possible patterns are allowed.

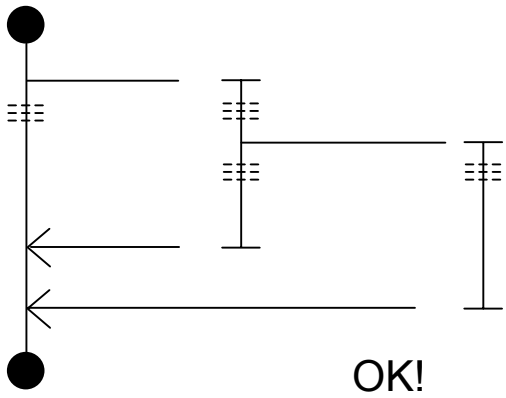


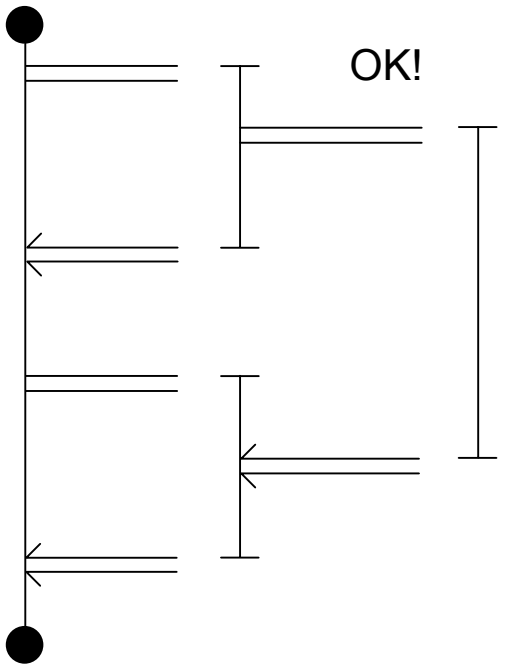
(b) When different routes are used as a route at which a branch starts and a routed at which routes are joined

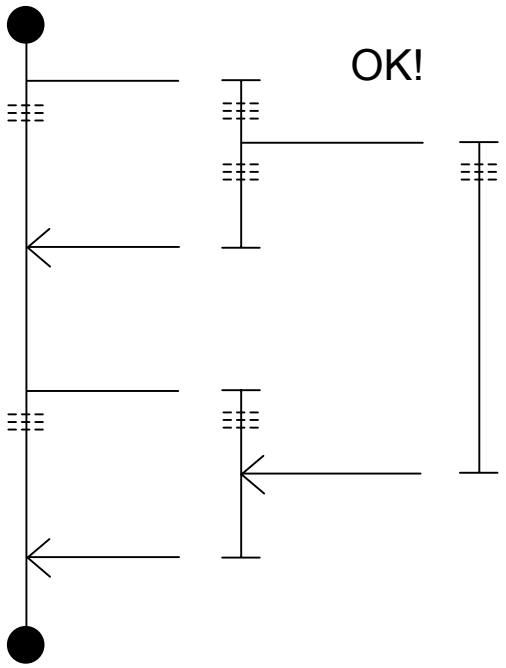
When the synchronization syntax and selection syntax are programmed independently, operation is possible. Otherwise, a program can be created but it cannot be operated correctly.

[The program runs correctly.]

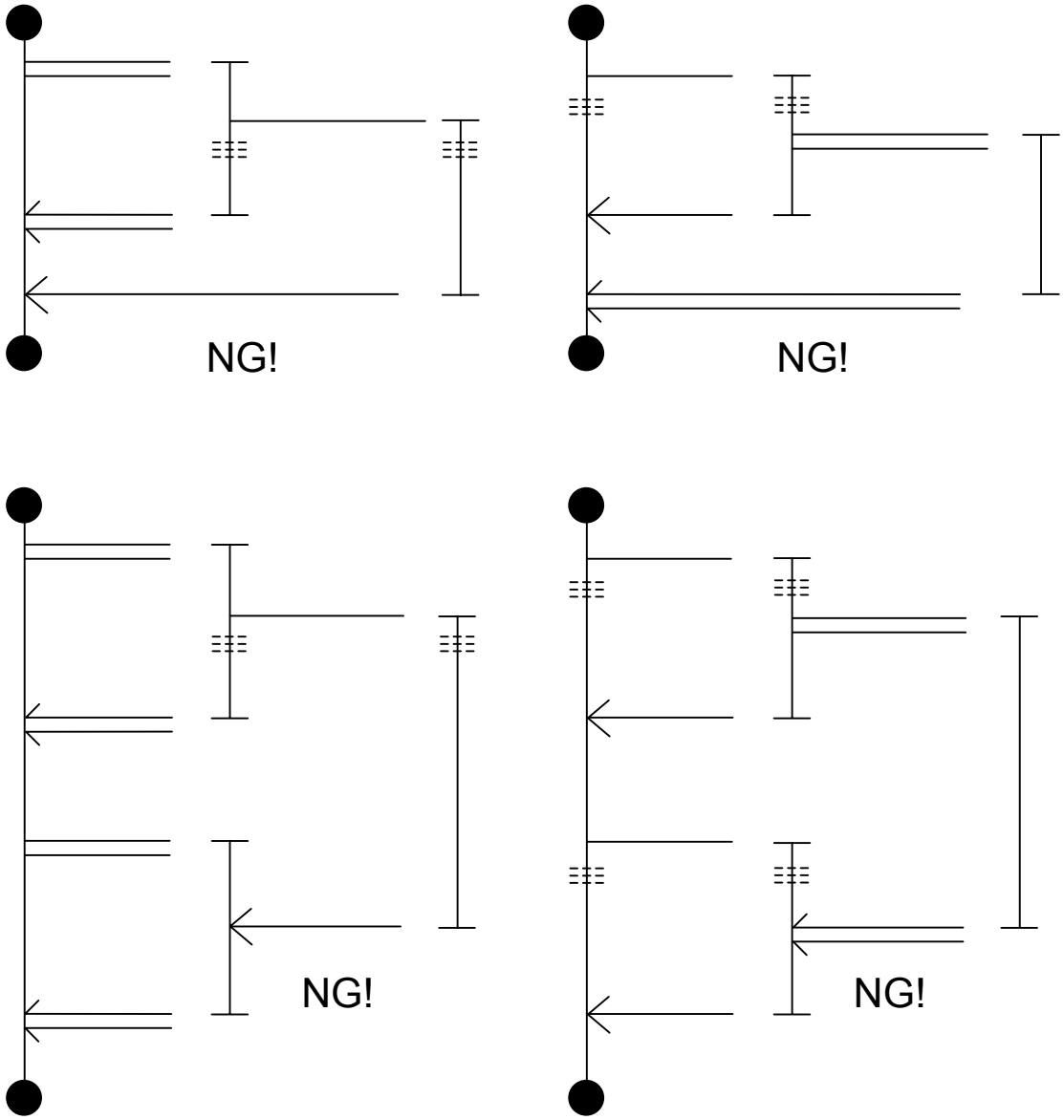






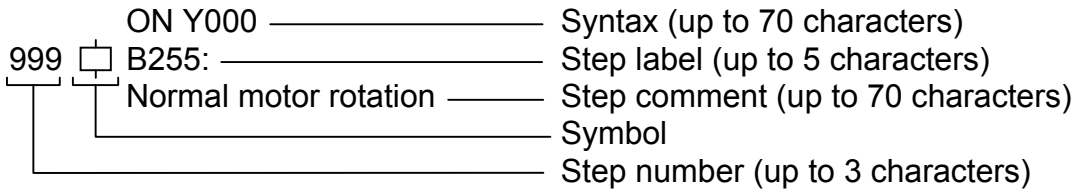


[The program does not run correctly.]

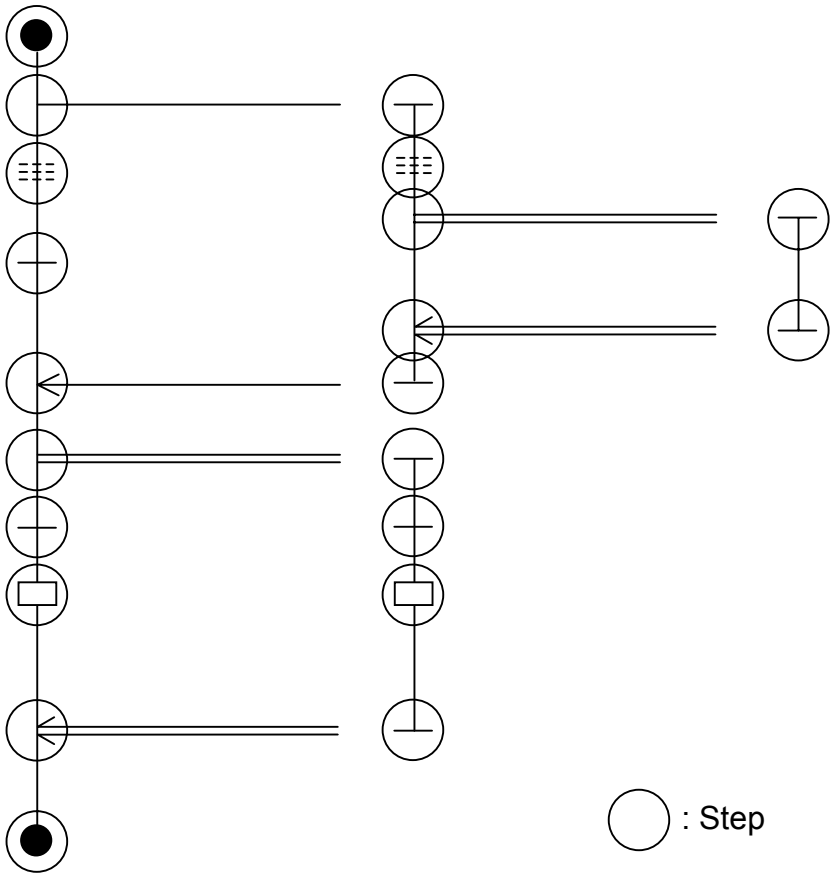


Step

A step is an instruction unit in HI-FLOW. As with a free label and free comment, a step is a unit of a route. A step consists of a step number, symbol, label, syntax, and step comment.

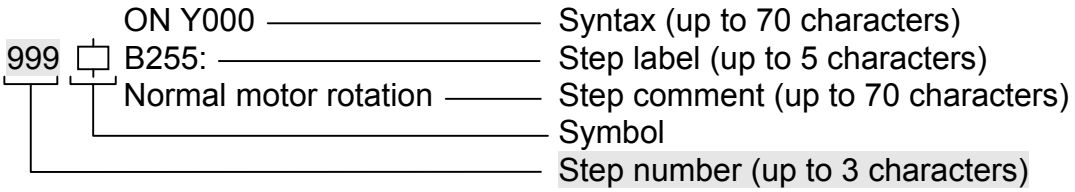


* Note that, as regards the combined use of syntax, label, and comment strings, the total length of the strings that is acceptable is up to 70 characters. Note, also, that a logical operator within a syntax string is considered to be two characters long in counting although it is regarded as a string of only one character in editing.



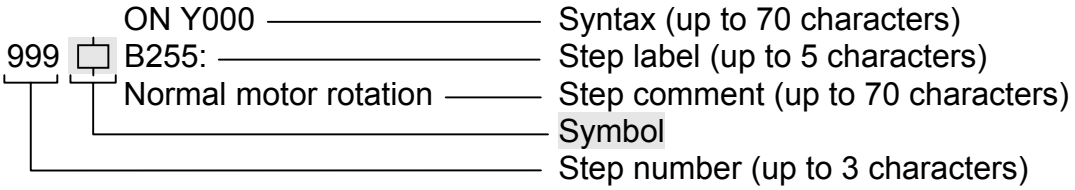
Step number

A step number is a unique number in the process. It is automatically assigned by the system when the program is created. (A process number is from 1 to 999. This means that up to 999 steps can be created in one process.)



Symbol

A symbol outlines a condition, branch, or other control. A symbol is always required when a step is created. Some steps consist of only a symbol. Other steps consist of a symbol and syntax.



There are 19 symbols (listed below). Symbols themselves have meanings.

* Note that, as regards the combined use of syntax, label, and comment strings, the total length of the strings that is acceptable is up to 70 characters. Note, also, that a logical operator within a syntax string is considered to be two characters long in counting although it is regarded as a string of only one character in editing.

[Symbols usable in HI-FLOW]

(1/2)

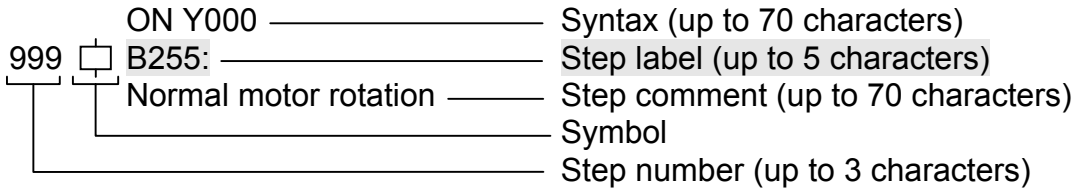
No.	Symbol	Name	Function	Syntax	Remarks
1		Process Start	Starts a process.	Yes	
2		Process End	Stops the process.	No	
3		Route Start	Starts a subroute.	No	
4		Route End	Stops the subroute.	No	
5		Repeat Start	Starts a repetitive process.	Yes	End is decided with >=.
6		Repeat End	Stops the repetitive process.	No	
7		If	Conditionally branches program control.	Yes	A branch to another route can be made.
8		Jump	Unconditionally branches program control.	No	A branch to another route can be made.
9		Escape	Forrcibly stops the local process.	No	A subprocess returns to the main process in the same scan.
10		Parallel Start	Branches to a synchronous subroute.	No	
11		Parallel End	Waits for the end of a synchronous subroute.	No	When program control is returned from the selectively branched subroutine, control is passed to the next step in the same scan. (With the previous model, a delay of one scan occurred.)
12		Select	Branches to a selectively branched subroutine.	No	
13		Wait in Selective Branching	Provides a route selection condition in selective branching.	Yes	Use Route Start and Select as a pair.
14		Select End	Returns program control form the selectively branched subroutine.	No	The subroute may not join the main route. (This is not possible in the previous model.) Control is passed to the next step with no scan delay. (With the previous model, a delay of one scan was required before the next step was executed.)

(2/2)

No.	Symbol	Name	Function	Syntax	Remarks
14	⊭	Multi-entry	Starts reexecution from this step when the set conditions are satisfied.	Yes	
15	┼	Wait	Waits for the shift condition to be satisfied.	Yes	
			Wait for the specified time to expire.	Yes	It is possible to monitor that the condition for consecutive PI/Os are satisfied.
16	□	Box	PI/O Output.	Yes	With interlocked Y-output.
			Assignment expression	Yes	
			Sends PI/O waveforms.	Yes	
			Resets timers.	Yes	Up to 7 timers can be reset. This function is equivalent to the conventional forcible timeout function.
			Causes a forcible timeout.	Yes	New function added to this model.
17	■	Control Box	Controls the state of another process.	Yes	The master reset function is provided. The function to specify a step is added. The function to specify STP timer measurement is added. The function to forcibly timeout or reset the RST timer is added.
			Controls a task.	Yes	
18	▭	Call	Call a subroute of another process.	Yes	The master reset function is provided. The function to specify a step is added.
19	○	Function	Application instruction	Yes	New function added to this model.
20	┼*	Condition with previous State Cleared	Clears PI/O when a condition is changed.	Yes	New function added to this model. Use this function together with the wait function.
		Wait Timer with previous State Cleared	Clears PI/O when a forcible timeout occurs.	Yes	

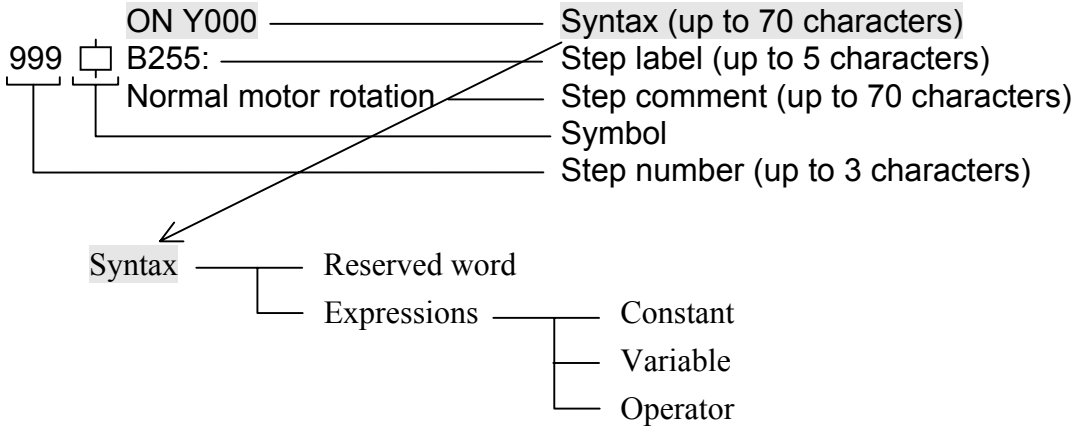
Label

A label represents a destination to which a symbol branches. A label is represented by a number from B1 to B255 followed by a colon (:). (These numbers can be created for each process. A branch to another process cannot be specified.) Labels can be assigned only to steps.



Syntax

A syntax clarified the contents of a condition expression, assignment expression, or control statement. Some symbols have no syntax. A syntax consists of a reserved word and an expression which consists of a constant, variable and operator.



* Note that, as regards the combined use of syntax, label, and comment strings, the total length of the strings that is acceptable is up to 70 characters. Note, also, that a logical operator within a syntax string is considered to be two characters long in counting although it is regarded as a string of only one character in editing.

Reserved word

Each reserved word has a specific meaning provided by the system. You cannot use reserved words as symbol names.

Reserved words

ACT, CLR, MRST, ON, OFF, RST, STP, TASK
TUP, TRS, TCNT, CNxxx, PTxxx, WTxxx,
Bxxx, Pxxx, H????????
Application instruction names (See “5 APPLICATION INSTRUCTIONS.”)

xxx: 3-digit decimal constant
????????: 8-digit hexadecimal constant

Constant

Long-word constants can be specified in HI-FLOW.

- Constants
  - Bit type : 0, 1
  - Word type: Decimal numbers from -32768 to 32767
    Hexadecimal numbers from H0 to HFFF
  - Long type : Usable only in application instructions
    Decimal numbers from [-2147483648] to [2147483647]
    Hexadecimal numbers from [H0] to [H7FFFFFFF]

Variable

You can use physical PI/O registers such as X and Y.
In application instructions, @ can be prefixed to PI/O for indirect specification of a variable, and a variable can be enclosed by brackets ([ ]) to handle it as long word.
The physical PI/O registers usable in HI-FLOW are listed below.

- Variables
  - Bit type
    - Simple
    - One-dimensional array such as X000 (5)
  - Word type
    - Simple
    - One-dimensional array such as XW000 (FW000)
  - Long type
    - Simple
    - Usable only in application instructions.
      When [FW000] is specified, FW000 and FW001 are handled as long words.

[PI/O registers]

Item	Symbol		Range	Type	Remarks
Registers	Extemal input registers	X	000 to FFF	Bit	
		XW	000 to FF0	Word	
	Extemal output registers	Y	000 to FFF	Bit	
		YW	000 to FF0	Word	
	Communication link registers	G	000 to FFF	Bit	
		GW	000 to FF0	Word	
		A	000 to FFF	Bit	
		AW	000 to FF0	Word	
	Internal registers	R	000 to FFF	Bit	
		RW	000 to FF0	Word	
		K	000 to FFF	Bit	
		KW	000 to FF0	Word	
		M	000 to FFF	Bit	
		MW	000 to FF0	Word	
		E	000 to FFF	Bit	
		EW	000 to FF0	Word	
		Z	000 to 3FF	Bit	
		ZW	000 to 3F0	Word	
		S	000 to BFF	Bit	
		SW	000 to BF0	Word	
	Other registers	J	000 to FFF	Bit	These registers establishes a link to the ladder program.
		JW	000 to FF0	Word	
		Q	000 to FFF	Bit	
		QW	000 to FF0	Word	
		HH	000 to 1FF	Bit	These registers establishes a link to another process.
		DW	000 to FFF	Word	
		FW	000 to BFF	Word	
	Timers	WT	000 to 255		Decimal notation
		PT	000 to 255		
	Counter	CN	000 to 127		Decimal notation
	Label	B	001 to 255 (You can set a label consisting of six or less charactera.)		Decimal notation. Specifiable for each process.

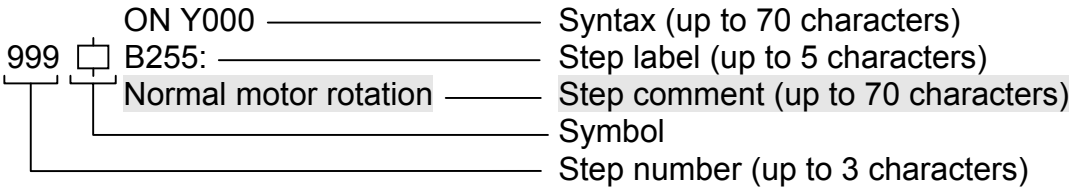
Operator

As with the previous model, there are four operators: parentheses, arithmetic operators, relational operators and logical operators.

Item		Description	Priority
Operators	Logical	& (AND) | (OR) ~ (NOT) ∧ (Exclusive OR)	5
	Arithmetic	* /	2
		+ -	3
	Relational	=, < >, <, >, >=, <=	4
	Parentheses	Parentheses can be nested at up to 7 levels.	1

Step comment

A step comment is a character string consisting of a combination of letters, numbers and special characters. You can write a statement comment as long as it fits in one line (74 columns). However, you do not always need to write a statement comment to the full length of a line.



* Note that, as regards the combined use of syntax, label, and comment strings, the total length of the strings that is acceptable is up to 70 characters. Note, also, that a logical operator within a syntax string is considered to be two characters long in counting although it is regarded as a string of only one character in editing.

Free label

You can create, in other than steps, labels to which symbols branch. These labels are called free labels. They are optional. A free label can be assigned any name (other than reserved words) beginning with a letter consisting of six or less characters and ending with a colon (:). You can use free labels only in other than steps.

LABEL: ——————————— Free label (up to 6 characters)
Joining point ——————————— Free comment (up to 70 characters)

Free comment

You can create comments in other than steps. These comments are called free comments. They are optional. A free comment is a character string consisting of a combination of letters, numbers and special characters. You can write a free comment as long as it fits in one line. You can add free comments to more identifiable points.

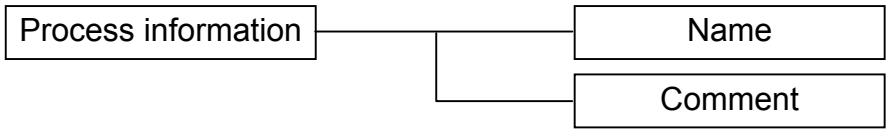
LABEL: ——————————— Free label (up to 6 characters)
Joining point ——————————— Free comment (up to 70 characters)

* As regards the combined use of a free label and free comment, the total length of the strings that is acceptable is up to 70 characters (including the colon ":" for a free label).

3.3 Process Information

A process consists of programs and process information.

In process information, additional information on processes is defined. Process information consists of five elements that you can change freely with process information commands.



Name

Name used in process information. You can assign unique name to the process with up to 16 characters.

Comment

Comment used in process information. You can assign comments to the process with up to 132 characters.

4 EXPLANATION OF SYNTAXES

This chapter explains syntaxes consisting of symbols and destination labels with typical examples.
In the following syntaxes, optional items are enclosed in brackets ([ ]).
Of the items enclosed in braces ({ }), select a desired one. The items followed by ~ are repeated.

4.1 Process Start and Process End

Process Start starts a process. Process End ends the process. Symbols for them are automatically added. You do not need to enter them.
Process Start sets a condition under which the process is stopped, reset or restarted, or PI/O is initialized. (See the descriptions of STP, RST, ACT and CLR.)
Process End performs processing if all routes except the local route have been ended. If not, Process End waits for them to be ended. At activation, Process End clear to 0 bit type PI/Os to be turned on by the local process if master reset is specified. (See the ON statement and parallel timer.)
The timers being used by the local process are handled according to the activation method. If the timer was activated with the TUP option specified, it is forcibly timed out. If the TRS option was specified for activation, the timer is reset to 0 after it expires. When no option was specified, the timer continues measurement.

[Syntax]

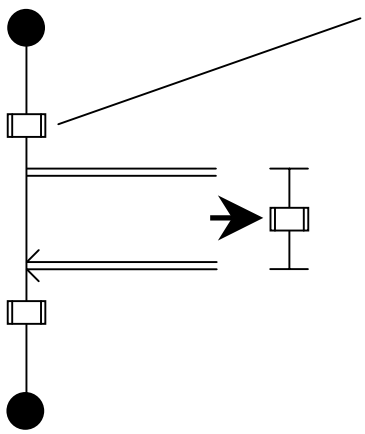
● [{STP-condition-expression [, TCNT] ┌{ON PI/O-bits [:OFF PI/O-bits]}┐ }
└{OFF PI/O-bits [:ON PI/O-bits]}┘
{,RST-condition-expression [, TUP] ┌{ON PI/O-bits [:OFF PI/O-bits]}┐ }
└{OFF PI/O-bits [:ON PI/O-bits]}┘
{,CLR-condition-expression}
{,ACT-condition-expression}]
*PI/O-bits - PI/O-bit-expression [, PI/O-bit-expression] ~

● No syntax

STP

- If the condition expression is satisfied during process execution, execution of the local process is stopped under the current execution state. (The process enters the stop state.)

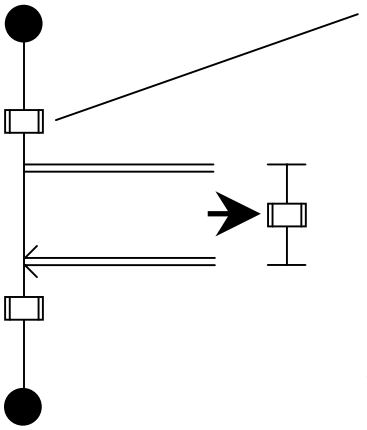
- When the condition expression for STP is satisfied, the value of the timer and the values of bit-type PI/Os to be turned on by the local process are retained. (See the ON statement and parallel timer.) However, note that it is unavoidable that the local process is turned on or off by other processes.
- When the condition expression for STP is satisfied, PI/O bits with option specification are turned on or off as specified. (If the condition expression is not satisfied, PI/O bits are turned on or off at each scan, contrary to the specification.)
- With the [,TCNT] option specified, the timer continues measurement even when the process enters the stop state. If the option is not specified, the timer value is retained.
- When the condition expression for STP is satisfied, the called process enters the stop state as with the calling process. However, the processes for which calls are ended or calls are not made are not affected.



RST

- If the condition expression is satisfied while the process is being executed or stopped, the execution of the local process is stopped and the process enters the wait state with Process Start. (The process enters the reset state.)
- When the condition expression for RST is satisfied, the values of bit-type PI/Os to be turned on or off by the local process are retained. (See the ON and OFF statement and parallel timer.) However, note that it is unavoidable that the local process is turned on or off by other processes.
- When the condition expression for RST is satisfied, PI/O bits with option specification are turned on or off as specified. (If the condition expression is not satisfied, PI/O bits are turned on or off at each scan, contrary to the specification.)
- With the [,TUP] option specified, the value of the timer is set to the specified value and the timer is forcibly timed out. If the option is not specified, the timer is cleared to 0 and measurement is canceled.

• When the condition expression for RST is satisfied, the called process enters the executable state. In this case, PI/Os and timers are handled according to the activation method. The processes for which calls are ended or not made are not affected.



If a parallel timer is performing measurement when the condition expression for RST is satisfied, the timer is not affected. (This is because a call has been ended.)

If the condition expression for RST is satisfied during a call of this process, the called process enters the executable state.
If master reset activation is specified, PI/Os are cleared. If zone activation is specified, PI/Os are retained. If a call is made with the [,TUP] option specified, the timer is forcibly timed out. If a call is made with the [,TRS] option specified, the timer is reset to 0 after it expires. When no option was specified, the timer continues measurement.

CLR

If the condition expression is satisfied in the stop or reset state, bit-type PI/Os which are used in the ON statement or parallel timer and are to be turned on by the local process are cleared to 0.

ACT

When the condition expression for STP or RST is not satisfied in the stop or reset state, process execution is resumed after the condition expression is satisfied. (The process enters the execution state.)

[Sample programs containing Process Start (●)]
(~ : Indicated that the same line is repeated.)

1. ● STP X000, RST X001, CLR X002, ACT X003

When X000 is on, the process enters the stop state. (The timer value is retained.)
When X001 is on, the process is reset. (The timer is reset to 0 after it expires.)
When X002 is on in the stop or reset state, the bit-type PI/Os in the ON statement or parallel timer used in the process are cleared to 0.
When X000 and X001 are off and X003 is on, the process is executed.

2. ● STP G000 & X002, TCNT [ON J000: OFF J001] ~ , RST Q000, TUP

When both G000 and X020 are on, the process is stopped. (The timer continues measurement.)
When the process is stopped, J000 is turned on and J001 is turned off.
When Q000 is on, the process is reset. (The timer is forcibly timed out.)
During process execution, J000 is turned off and J001 is turned on, at each scan.

3. ● RST FW000<DW000 [OFF G100], ACT FW001=0

When FW000 becomes smaller than DW000, the process is reset. (The timer is reset to 0 after it expires.)
When the process is reset, G100 is turned off.
When FW000 is greater than or equal to DW000 and FW001 is 0 in the stop or reset state, the process is executed.
G100 is turned on at each scan during process execution.

4. ● RST Q001, TUP [ON J001, G200], CLR X200

When Q001 is on, the process is reset. (The timer is reset to 0 after it expires.)
When the process is reset, J001 and G200 are turned on. When X200 is on in the stop or reset state, the ON statement and bit-type PI/Os in the parallel timer used in the process are cleared to 0.
J001 and G200 are turned off at each scan during process execution.

STP, RST, CLR, and ACT can be specified in any order in Process Start.

4.2 Route Start and Route End

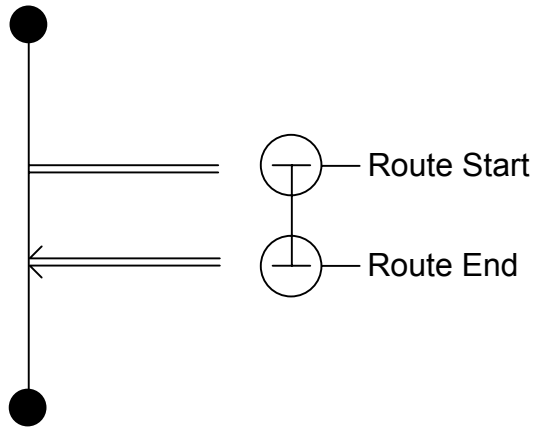
⊤ No syntax

⊥ No syntax

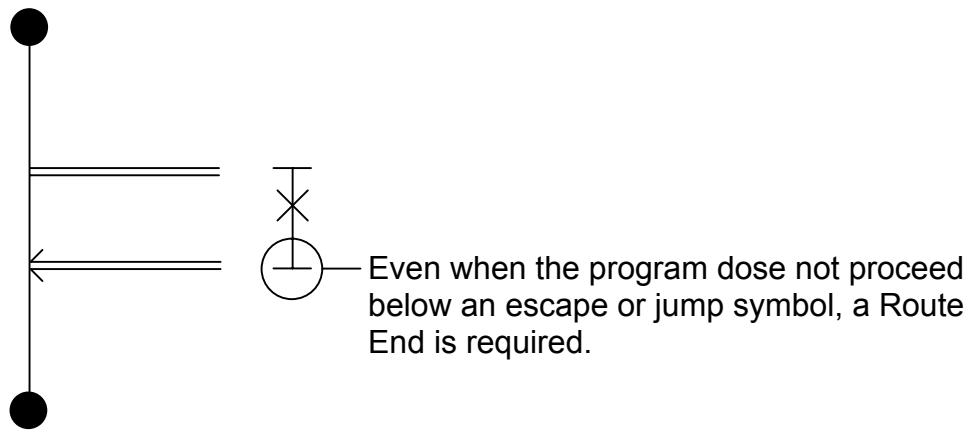
Route Start starts a subroute. Route End ends the subroute. Be sure to use Route Start and Route End as a pair. Creation of subroutes enables synchronous syntaxes and selective branch syntaxes to be implemented. For operation after Route End is executed, see the chapter describing the synchronous/selective syntax execution order.

[Sample programs containing Route Start (⊤) and Route End (⊥)]

1.



2.



4.3 Wait

The program waits at this step until the condition to proceed to the next step is satisfied. The condition is specified by a condition expression or the wait timer that makes the program wait until the specified time is reached.

[Syntax]

┼ { condition-expression[, timer, output-bit] }
{ WTxxx (expression[, condition-expression]) }

Condition expression

A condition expression consists of bit-type or word-type numbers and operators.

Timer (usable on HI-FLOW system version 07-00 or later only)

- Monitoring timer that checks whether the condition expression yields an output ("true") within the specified period of time. A timer setting is possible in 100-millisecond units.
- Enter a constant in decimal notation.
- The acceptable range of settings is from 0 to 32767. If you set the range of -32768 through -1, the program runs as if 32768 through 65535 were set.
- The maximum number of timers that can be monitored simultaneously is 64. Do not put more than 64 timers under monitoring at the same time.

Output bit (usable on HI-FLOW system version 07-00 or later only)

- This bit is set if the condition is not met within the time period specified by the timer.
- The following registers can be designated as the output bit:
  (Y, G, A, R, K, M, E, Z, S, J, Q)
- The output bit is automatically reset unconditionally at the beginning of the monitoring process.
- Switching to the next step does not take place unless the condition is met within the timer-specified period of time.
- Even if the condition is met after output bit ON (set), the output bit is not turned OFF (reset).

Wait timer

- A wait timer delays execution for the specified time at the desired step. WT0 to WT255 (numbers are in decimals) can be used. A delay can be specified in 100 ms increments in the range from 0 to 32767 in decimals. If you set the range of -32768 through -1, the program runs as if 32768 through 65535 were set.
- When wait timers identified by the same number make the program wait at multiple steps, the step that occupies the timer first is awaited as specified. The other steps turn on the specified PI/O (by default, HH1FA) and wait until the step releases the timer. Therefore, the other steps are awaited longer than the specified time.
- A condition expression can be specified for a wait timer. In this case, execution is awaited until the condition expression is satisfied continuously for the specified time.

[Sample programs containing Wait (┼)]

1. ┼ X000

When X000 is on, the program proceeds to the next step.

2. ┼ GW000<H2000

When GW000 becomes smaller than H2000, the program proceeds to the next step.

3. ┼ X001 (FW000)

When the X register having the value of FW000 as a subscript is turned on during condition check, the program proceeds to the next step. (The condition check may vary at every time.)

4. ┼ WT000 (100)

The program proceeds to the next step 10 seconds after the program reaches this step first.

5. ┼ WT255 (10, X01F)

The program proceeds to the next step after it reaches this step then X01F is turned on continuously for one second.

6. ┼ GW000>H2000, 100, Y000

Switching to the next step takes place when the GW000 value is greater than the H2000 value. If the GW000 value remains smaller than the H2000 value for a period of 10 seconds or longer, the Y000 is turned ON (set).
Even if the GW000 value is greater than the H2000 value after Y000 ON, the Y000 is not turned OFF (reset).

4.4 Box

Box controls PI/O output, data processing, and timers. Multiple Boxes separated by semicolons (;) can be specified.

[Syntax]

```
□ {ON  PI/O-bit-expression [, PI/O-bit-expression] ~}
  {OFF PI/O-bit-expression [, PI/O-bit-expression] ~}
  {assignment-expression}
  {PT-number (t1 [, t2] ], {ON bit-PI/O [, bit-PI/O] ~ [:OFF bit-PI/O [, bit-PI/O] ~] } ) }
                           {OFF bit-PI/O [, bit-PI/O] ~                              }
  {  {TUP}  {WT-number} [  {, WT-number} ] }
     {TRS}  {PT-number}    {, PT-number} ~
            {CN-number}  [  {, CN-number} ]

[ {: ON PI/O-bit-expression [, PI/O-bit-expression ~ repeated] }                                  ]
[ {: OFF PI/O-bit-expression [, PI/O-bit-expression ~ repeated] }                                 ]
[ {: Assignment-expression}                                                                       ]
[ {: PT-number (t1 [, t2], {ON bit-PI/O [, bit-PI/O] ~ [:OFF bit-PI/O [, bit-PI/O] ~] } ) }  ~   ]
[                          {OFF bit-PI/O [, bit-PI/O] ~                              }          ]
[ {: {TUP}  {WT-number} [  {, WT-number} ] }                                                      ]
[    {TRS}  {PT-number}    {, PT-number} ~                                                       ]
[           {CN-number}  [  {, CN-number} ]                                                       ]
```

Assignment expression

An assignment expression assigns the result of a logical or arithmetic calculation to a variable. A one-dimensional array is allowed for expressions. Array subscripts are allowed only for word-type variables. The usable variables and operators are shown below.

Bit-type variables

Y, G, A R, K, M E, Z, J Q, HH	=	Y, G, A R, K, M E, Z, J Q, HH X, S 0, 1	( ) & | ~ ∧	Y, G, A R, K, M E, Z, J Q, HH X, S 0, 1

Word-type variables

YW, GW AW, RW KW, MW EW, ZW JW, QW DW, FW	=	YW, GW AW, RW KW, MW EW, ZW JW, QW DW, FW XW, SW Decimal Hexadecimal	( ) & | ~ ∧ * / + -	YW, GW AW, RW KW, MW EW, ZW JW, QW DW, FW XW, SW Decimal Hexadecimal

Operands and results are assumed to be unsigned.
A multiplier and multiplicand must be both one word long.
Multipliers and multiplicands that are too long truncated to one word. The result is also one word long.
A divisor and dividend must be both one word long. A divisor and dividend that are too long are truncated to one word. The result is also one word long. If division by 0 is performed, the answer remains unchanged.
There is no answer back for the operation result state (such as normal termination or overflow). If answer back is required, use application instructions.

[Sample programs containing the assignment statement (□)]

1. □ FW000=FW001+FW002

   The current value of FW001 and that of FW002 are added and the result is assigned to FW000. Then the program proceeds to the next step.

2. □ YW000 (DW001)=HFFFF

   /FFFF is assigned to the array of YW000 having the current value of DW001 as a subscript.

ON statement

The ON statement turns on the specified PI/O output bit (Y, G, A, R, K, M, E, Z, J, Q, or HH). When multiple bits separated by commas (,) are specified, multiple outputs can be obtained for PI/O. A one-dimensional array is allowed for PI/O output bits. Array subscripts are allowed only for word-type variables.

[Sample programs containing the ON statement (□)]

1. □ ON Y000, Y00F:OFF Y001

   Y000 and Y00F are turned on, and Y001 is turned off. Then the program proceeds to the next step.

2. □ ON G000 (GW010)

   The bit separated by the value of current GW010 from G000 is turned on. Then the program proceeds to the next step.

OFF statement

The OFF statement turns off the specified PI/O output bit (Y, G, A, R, K, M, E, Z, J, Q, or HH). When multiple bits separated by commas (,) are specified, multiple outputs can be obtained for PI/O. A one-dimensional array is allowed for PI/O output bits. Array subscripts are allowed only for word-type variables.

[Sample programs containing the OFF statement (□)]

1. □ OFF Y000, Y001

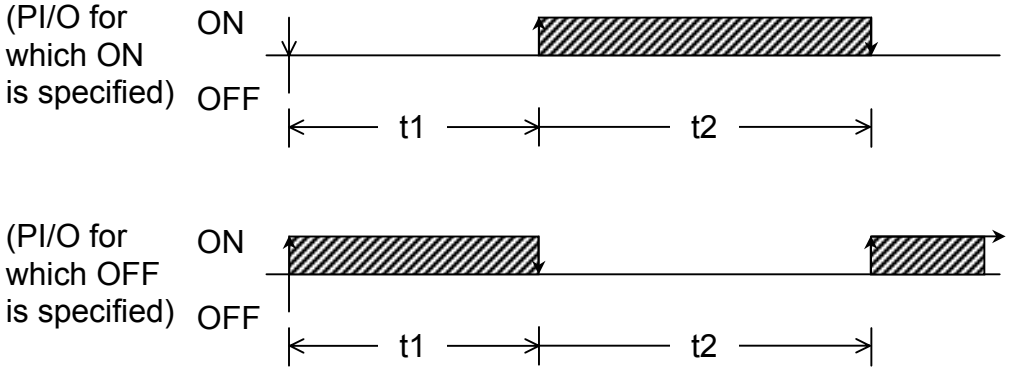
   Y000 and Y001 are turned off, and Y001 is turned off. Then the program proceeds to the next step.

2. □ OFF G000 (GW010)

   The bit separated by the value of current GW010 from G000 is turned off. Then the program proceeds to the next step.

Parallel timer

The parallel timer sends a waveform to a desired PI/O. t1 is a rising time and t2 is a falling time. When t1 is 0, PI/O for which ON is specified is just turned off after t2 elapses and PI/O for which OFF is specified is just turned on after t2 elapses. When t2 is 0 or is omitted, PI/O for which ON is specified is just turned on after t1 elapses and PI/O for which OFF is specified is just turned off after t2 elapses. After an instruction for waveform output is issued, the program proceeds to the next step soon.
Parallel timers can be specified in the range from PT000 to PT255. In t1 and t2 each, a time can be specified in 100 ms increments in the range from 0 to 32767. If you set the range of -32768 through -1, the program runs as if 32768 through 65535 were set.
If the specified timer is already in use at timer activation, the specified PI/O (by default, HH1F9) is turned on and the program waits until the timer is released.
Multiple PI/Os separated by commas (,) can be coded. Multiple statements separated by colons (:) can be coded. A one-dimensional array is also allowed. Usable bit PI/Os are Y, G, A, R, K, M, E, Z, J, Q and HH.



[Sample programs containing parallel timers (⏀)]

1. ⏀ PT000 (10, 10, ON Y000 : OFF Y001)

	When this step is passed, the program proceeds to the next step soon.	1 second later	2 seconds later
Y000	?→OFF	→ON	→OFF
Y001	?→ON	→OFF	→ON

2. ⏀ PT010 (20, ON G000 : OFF G001)

	When this step is passed, the program proceeds to the next step soon.	2 seconds later	
G000	?→OFF	→ON	→
G001	?→ON	→OFF	→

3. ⏀ PT255 (0, 30, ON J100 : OFF J101)

	When this step is passed, the program proceeds to the next step soon.	3 seconds later	
J100	?→ON	→OFF	→
J101	?→OFF	→ON	→

TUP (Timer Up)

TUP forcibly times out the timers that are performing measurement. When a wait timer is performing measurement, the timer value is reset to the specified value. As a result, the wait state is released and the program waiting for a forcible time-out proceeds to the next step. For a parallel timer, the timer value is set to t2 (t1 when t2 is omitted). As a result, the timer provides PI/O output earlier than the specified time. For a loop counter, the timer value is set to the end value. As a result, the program exits at the next loop check.

[Sample programs containing TUP (⏀)]

1. ⏀ TUP WT001, WT002, PT001, CN001

Wait timers 1 and 2, parallel timer 1, and counter 1 are forcibly timed out.

TRS (Timer Reset)

TRS resets the timers that are performing measurement. For wait timers and loop counters, TRS provides the same effect as TUP. For parallel timers, however, TRS resets t1 and t2. The state of the specified PI/O is the same as when a timer reset is indicated.

[Sample programs containing TRS (⊏)]

1. ⊏ TRS WT001, WT002, PT001, CN001

Wait timers 1 and 2, parallel timer 1, and counter 1 are reset.

4.5 Control Box

Control Box activates (reactivates), stops or resets other processes, or clears PI/O.

[Syntax]

■ {ACT Pxxx {-Pxxx] [, step-number] [, MRST] [{, TUP}] } }
{, TRS}
{, TASK, factor-number }
{RST Pxxx { [-Pxxx] [, TUP] } }
{ [, TASK] }
{STP Pxxx [-Pxxx]], TCNT] }
{CLR Pxxx [-Pxxx] }

ACT

	Item	Description
1	Function outline	ACT activates the specified processes. Specifiable processes are P0 to P255. A range of processes can be specified with a hyphen (-). When a step number is omitted, execution starts with step 1. The specified step may not be a main route. After activation, the program proceeds to the next step soon.
2	Behavior of the activated process	When Process End finishes execution, the activated process is executed again from the process start point at the next scan. (This is also true when a step is specified.)
3	Activating a process being executed	The ACT bit that represents the Control Box result is turned on. Then the program proceeds to the next step (by default, HH1FF).
4	Activating a nonexisting process	
5	Activating a stopped process	The stopped process is activated and execution is resumed.
6	Activating a reset process	The reset process is activated and execution is resumed from the process start point.
7	Instruction for timer states	When Process End or Escape is executed with the ,TUP option specified or the executable state is entered, the parallel timers occupied by the local process are forcibly timeout. When Process End or Escape is executed with the ,TRS option specified or the executable state is entered, the parallel timers occupied by the local process are reset.
8	Activation with master reset specified	When Process End or Escape is executed with the ,MRST option specified or the executable state is entered, bit-type PI/Os turned on by the local process are cleared to 0. (ON statement, parallel timer)
9	Activating a CPMS task	Specify the ,TASK,factor-number options and specify a CPMS task (1 to 127) with Pxxx. Then issue the RLEAS and QUEUE macros.

RST

	Item	Description
1	Function outline	RST resets the specified processes. Specifiable processes are P0 to P255. A range of processes can be specified with a hyphen (-). After Control Box with RST specified is issued, the program proceeds to the next step soon.
2	Behavior of the reset process	Execution of the specified process is canceled. The process enters the reset state and waits to be reexecuted with Process Start. (When the process is activated by another process with ACT or the ACT condition is satisfied to start the local process, the process is reexecuted.)
3	Instruction for timer states	The parallel timers occupied by the process with the ,TUP option specified are forcibly timed out. When the ,TUP option is not specified, the timers are reset. The option is valid only for the specified process. The called processes are not affected.
4	PI/O for the reset process	When the process has been activated with master reset specified, bit-type PI/Os turned on or off by the local process are cleared to 0.
5	Resetting a stopped process	Execution of the specified process is canceled. The process enters the reset state and waits to be reexecuted with Process Start. (When the process is activated by another process with ACT or the ACT condition is satisfied to start the local process, the process is reexecuted.)
6	Resetting a nonexisting process	The RST bit that represents the Control Box result is turned on. Then the program proceeds to the next step (by default, HH1FD).
7	Resetting the local process	Specify the local process number with the parameter.
8	Stopping a CPMS task	Specify the ,TASK option and specify a CPMS task with Pxxx. Then issue the ABORT macro.

STP

	Item	Description
1	Function outline	RST stops the specified processes. Specifiable processes are P0 to P255. A range of processes can be specified with a hyphen (-). After Control Box with STP specified is issued, the program proceeds to the next step soon.
2	Behavior of the stopped process	Execution of the specified process is stopped. The process enters the stop state and waits to be reexecuted at the current execution point.
3	Conditions for reexecution	The process is reexecuted when it is activated by another process with ACT or the ACT condition is satisfied to start the local process, the process is reexecuted.
4	Instruction for timer states	The parallel timers occupied by the process with the ,TCNT option specified continue measurement. When the ,TCNT option is not specified, the timers are stopped. The option is valid for all processes linked by the specified process.
5	PI/O for the stopped process	When the process has been activated with master reset specified, bit-type PI/Os turned on or off by the local process are cleared to 0.
6	Stopping a nonexisting process	The STP bit that represents the Control Box result is turned on. Then the program proceeds to the next step (by default, HH1FE).
7	Resetting a reset process	
8	Stopping the local process	Specify the local process number with the parameter.

CLR

	Item	Description
1	Function outline	Bit-type PI/Os turned on or off by the specified processes are cleared to 0. Specifiable processes are P0 to P255. After Control Box with CLR specified is issued, the program proceeds to the next step soon. This function is valid only when the specified processes are stopped or reset. PI/Os used by other processes are cleared without checking their usage states. A range of processes can be specified with a hyphen (-).
2	Clearing a nonexisting process	The CLR bit that represents the Control Box result is turned on. Then the program proceeds to the next step (by default, HH1FC).
3	Clearing a process being executed	
4	Clearing a process not activated	

[Sample programs containing Control Box (■)]

1. ■ ACT P1-P5, MRST

   Processes 1 to 5 are activated in master reset mode, starting from step 1. Then the program proceeds to the next step. When Process End or Escape is executed or the process enters the executable state, the parallel timers continue measurement.

2. ■ ACT P100, 5, TUP

   Process 100 is activated in zone mode, starting from step 5. Then the program proceeds to the next step. When Process End or Escape is executed or the process enters the executable state, the parallel timers are forcibly timed out.

3. ■ ACT P80, TASK, 3

   When the RLEAS macro is issued to CPMS task 80 and the QUEUE macro is issued with factor 3, the program proceeds to the next step.

4. ■ RST P10

   Process 10 is reset then the program proceeds to the next step. The parallel timers are reset, which were performing measurement when Control Box with RST specified was issued.

5. ■ RST P11, TUP

   Process 11 is reset then the program proceeds to the next step. The parallel timers are forcibly timed out, which were performing measurement when Control Box with RST specified was issued.

6. ■ RST P12, TASK

   After the ABORT macro is issued to CPMS task 12, the program proceeds to the next step.

7. ■ STP P50

   Process 50 is stopped then the program proceeds to the next step. The parallel timers and/or wait timers are stopped, which were performing measurement when Control Box with STP specified was issued.

8. ■ STP P51, TCNT

   Process 51 is stopped then the program proceeds to the next step. The parallel timers and/or wait timers continue measurement without being stopped, which were performing measurement when Control Box with STP specified was issued.

9. ■ CLR P40

   Bit-type PI/Os used by process 40 are cleared to 0 then the program proceeds to the next step.

4.6 Repeat Start and Repeat End

Repeat Start and Repeat Stop bracket the steps to be executed repeatedly. If Repeat Start steps and Repeat Stop steps are not paired correctly within the same loop, a syntax error is detected. An increment is added to the initial value after each repetition. Repetition continues until the result of addition exceeds the end value. If the initial value is greater than the end value, the program proceeds to the next step without executing the steps between Repeat Start and Repeat Stop.
When an increment is omitted, 1 is assumed. When 0 is specified as an increment, the program enters an infinite loop.
The setting range of the initial value, the end value, and increment is 0 through 32767. If you set the range of -32768 through -1, the program runs as if 32768 through 65535 were set.

[Syntax]

CNxxx (initial-value, end-value {, increment})
(xxx: Decimal number from 000 to 127)

No syntax

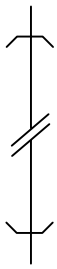
[Sample programs containing Repeat Start ( ) and Repeat End ( )]

1.

CN000 (1, 10)

The program executes the steps between Repeat Start and Repeat End repeatedly 10 times, then proceeds to the next step after Repeat End. Repeat Start is executed immediately after Repeat End is executed.

2.



CN127 (1, 5, 2)

The program executes the steps between Repeat Start and Repeat End repeatedly three times, then proceeds to the next step after Repeat End.

3.

CN001 (FW000, FW001, FW002)

The values of FW000, FW001, and FW002 when Repeat Start is passed first are the initial value, end value and increment, respectively.

4.7 If

If judges whether the specified condition expression is true or false and performs processing accordingly. When the condition expression is satisfied (true), If executes the steps between the comma (,) and semicolon (;). When the condition expression is not satisfied (false), If executes the steps after the semicolon. When the condition expression is not satisfied but there are no steps after the semicolon, the program proceeds to the next step. When a label is specified after the comma or semicolon, the program branches to the label.

[Syntax]

◇ condition-expression { destination-label (Bxxx) }
{ free-label }
{ON/OFF-statement} [{: ON/OFF-statement}]
{assignment-statement} [{: assignment-statement}]
{ACT-statement} [{: ACT-statement}]
{STP-statement} [{: STP-statement}]
{RST-statement} [{: RST-statement}] ~
{CLR-statement} [{: CLR-statement}]
{TUP-statement} [{: TUP-statement}]
{TRS-statement} [{: TRS-statement}]
{PT-statement} [{: PT-statement}]

[{; destination-label (Bxxx) }
{; free-label }
{; ON/OFF-statement} [{: ON/OFF-statement}]
{; assignment-statement} [{: assignment-statement}]
{; ACT-statement} [{: ACT-statement}]
{; STP-statement} [{: STP-statement}]
{; RST-statement} [{: RST-statement}] ~
{; CLR-statement} [{: CLR-statement}]
{; TUP-statement} [{: TUP-statement}]
{; TRS-statement} [{: TRS-statement}]
{; PT-statement} [{: PT-statement}]]

(xxx: Decimal number from 1 to 255)

<NOTICE>
A branch to another process cannot be made but a branch to another route can be made. In actual execution, however, the following branches may not be handled correctly:

- Branch to the inside of a loop between Loop Start and Loop End
- Branch from the inside of parallel processing
- Branch no the inside of parallel processing
- Branch to a route being executed

[Sample programs containing If (◇)]

1. ◇ X000, B1 ; LABEL

When X000 is on, the program jumps to the step having label B1. When X000 is off, the program jumps to the next step after the step having label LABEL.

2. ◇ H0 < > (YW000 & H3000), ON Q005

When the logical product of YW000 and H3000 is not 0, Q005 is turned on. If the logical product is 0, the program proceeds to the next step without doing nothing.

3. ◇ Q000, FW100=FW100+1 ; ACT P10

When Q000 is on, 1 is added to FW001. Then the program proceeds to the next step. When Q000 is off, process 10 is activated with ACT. Then the program proceeds to the next step.

4. ◇ GW000=4, STP P6 : RST P7 ; EW000=8 : ON J000

When GW000 is 4, process 6 is stopped and process 7 is reset. Then the program proceeds to the next step.
When GW000 is not 4, EW000 is set to 8 and J000 is turned on. Then the program proceeds to the next step.

5. ◇ X010, ON J000, J001, J002, J003 ; ERRLB

When X010 is on, J000, J001, J002, and J003 are turned on. Then the program proceeds to the next step. When X010 is off, the program jumps to the next step after the step having label ERRLB.

4.8 Jump

Jump causes the program to unconditionally jump to the specified label within the process. Label B1 to B255 are specifiable within one process. In HI-FLOW, free labels can be specified. (You can assign any names consisting of up to six characters to free labels. Free labels can be set in other than steps.)

[Syntax]

↳ { destination-label (Bxxx) }
{ free-label }

<NOTICE>

A branch to another process cannot be made but a branch to another route can be made. In actual execution, however, the following branches may not be handled correctly:

- Branch to the inside of a loop between Loop Start and Loop End
- Branch from the inside of parallel processing
- Branch no the inside of parallel processing
- Branch to a route being executed

[Sample programs containing Jump (↳)]

1. ↳ B1

The program jumps to the step having label B1 then starts execution from the step soon.

2. ↳ ERRBLK

The program jumps to the next step after the step having label LABEL then starts execution from the step soon.

4.9 Escape

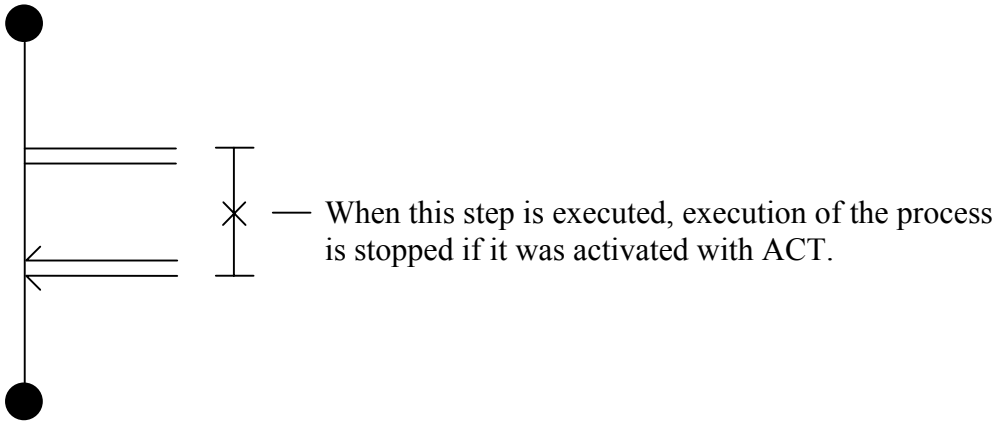
Escape forcibly terminates the local process. When the local process is a main process, Escape forcibly terminates all routes and the process enters the executable state. The processes being called, if any, are all escaped. The timers being used by the local process are handled according to the activation mode. (See the TUP and TRS options.)
When the local process is a subprocess, Escape functions in the same way as when the local process is a main process except that execution points are returned to the main process in the same scan.
When the local process was activated in master reset mode, the bit-type PI/Os to be turned on by the local process are cleared to 0. (See the ON statement and parallel timer.)

[Syntax]

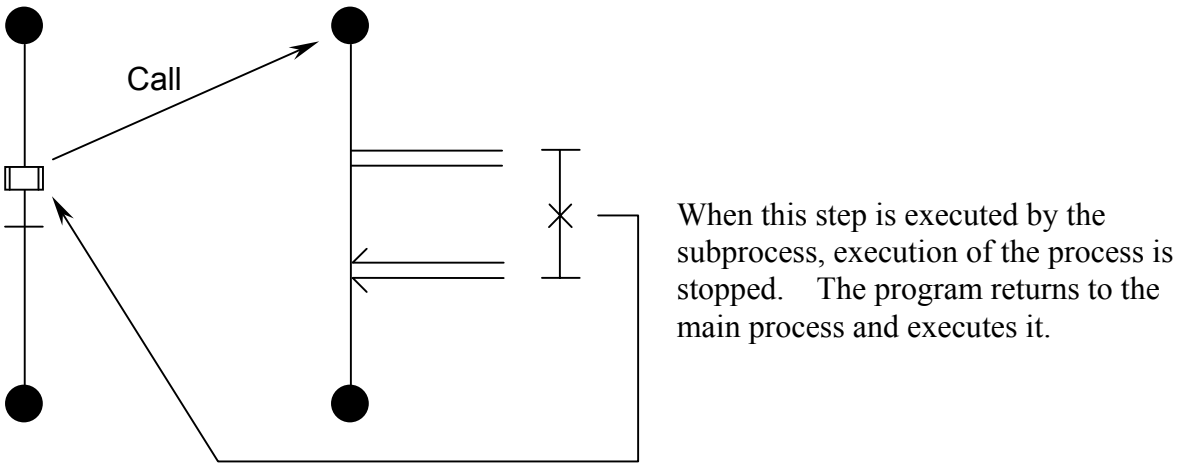
✳ No syntax

[Sample programs containing Escape (✳)]

1.



2.



4.10 Parallel Start and Parallel End

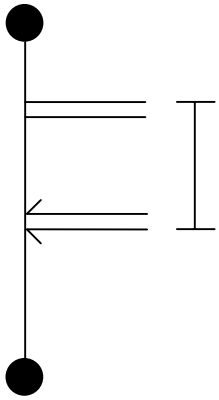
A Parallel Start and Parallel End pair represents a portion to be synchronously processed. After the synchronized subroute is activated, Parallel Start causes the program to proceed to the next step after the local route. After all joined routes are terminated, Parallel End instructs execution of the next step after the local route.
In the previous models, Parallel End monitored termination of the joined subroute (i.e., the main route was being executed), execution of the next step was delayed one scan. In this model, however, each of Parallel End and Route End checks whether it was joined last. If Parallel End or Route End was joined last, it instructs execution of the next step joining the main route. If not, it instructs termination of the local route (the main route is not always being executed). This eliminates a delay of one scan.

[Syntax]

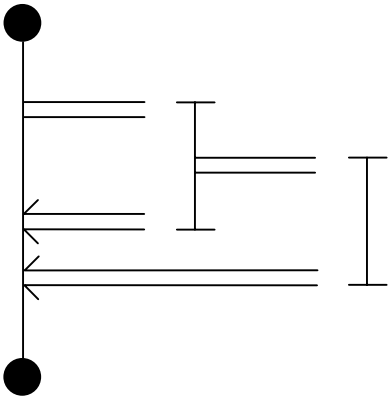
⊨ No syntax
⊭ No syntax

[Sample programs containing Parallel Start (⊨) and Parallel End (⊭)]

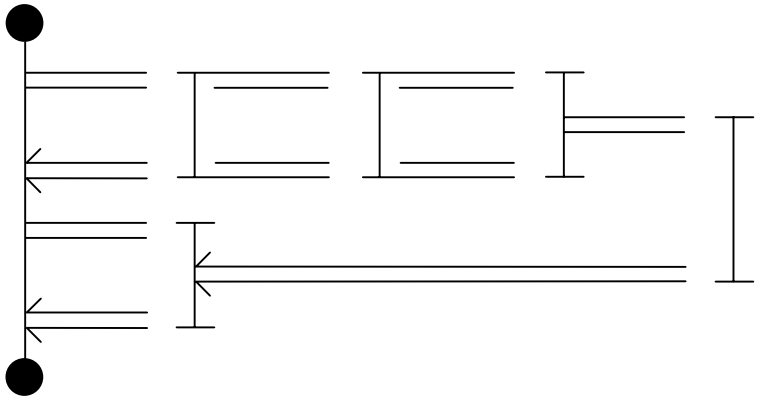
1.



2.



3.



4.11 Select, Wait in Selective Branching and Select End

A set of Select, Wait in Selective Branching and Select End represents a portion to be processed for selective branching.
After the selective branching route is activated, Select causes the program to proceed to Wait in Selective Branching in the local route. (Select and Wait in Selective Branching or Route Start and Wait in Selective Branching must be consecutive.)
When the condition expression for the local route is satisfied, Wait in Selective Branching terminates execution of the other routes. The program proceeds to the next step in the local route.
In the previous models, the main route was always being executed. (Both the main route and the selected route were being executed.) In this model, however, selection of a subroutine terminates the main route. (Only the selected route is executed.)
Condition expressions are checked sequentially from the leftmost route on the screen. When multiple conditions are met in the same scan, therefore, the leftmost route is selected.
In the previous models, Select End monitored termination of the joined subroutes (i.e., the main route was being executed), execution of the next step was delayed one scan.
In this model, however, when a subroute is selected, Route End of the route activates the main route and instructs execution of the next step joining the main route. This eliminates a delay of one scan.
Also in the previous models, Select End and Select had to be present in the same route. In this model, however, they may not need to be in the same route (they may not need to join the branch source route).

[Syntax]

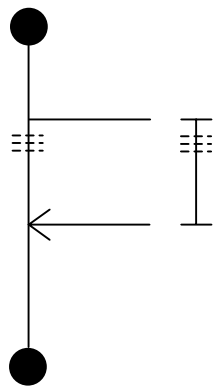
⊢ No syntax
⇶ condition-expression [, timer, output-bit]
⊭ No syntax

- Timer (usable on HI-FLOW system version 07-00 or later only)
- Output bit (usable on HI-FLOW system version 07-00 or later only)

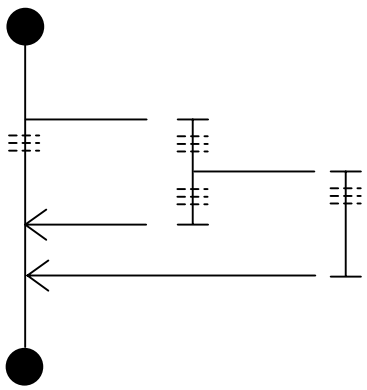
* For the timer and output bit, see “4.3 Wait.”

[Sample programs containing Select (⊢), Wait in Selective Branching (⇶) and Select End (⊭)]

1.



2.



4.12 Multi-entry

When a condition expression is set in the symbol for Select End, it is handled as Multi-entry.
When the condition expression is satisfied during process execution, execution is resumed from the step containing Multi-entry. (Even during execution of the first process, execution starts from the first process when the condition expression is satisfied.) The condition expression is checked at the beginning of scanning. This may delay execution a maximum of one scan.
Steps are checked against the condition sequentially from the one having the smallest step number. If several conditions are satisfied in the same scan, therefore, execution is resumed from the step having the smallest step number.
Multi-entry can also be set in subroutes.
When the condition is satisfied and the process is executed, all routes except those containing parallel timers, wait timers, counters, a process being called and Multi-entry are initialized but the PI/O values remain unchanged.

[Syntax]

⋞ condition-expression

<NOTICE>

- If Multi-entry is set between Loop Start and Loop End, the program may not run correctly.
- Multi-entry cannot be set in a subroute having a synchronous syntax.

[Sample programs containing Multi-entry (⋞)]

1. ⋞ X000

When X000 is on, reexecution starts from this step.

2. ⋞ GW000<H2000

When GW000 becomes smaller than H2000, reexecution starts from this step.

4.13 Call

Call makes a subroutine call for the process specified by one of P0 to P255. Execution starts from the step specified by the [,step-number] option. (When a step number is omitted, execution starts from Process Start.)
When the specified process or step is not found or the local process is called, the CALL bit that represents the Control Box result is turned on then the program proceeds to the next step.
If the specified process is already being executed, the program waits until it can call the process (until the process shifts to the executable state). However, a process which was called with ACT and is reset cab be called.
A subprocess can further call another process. Up to 16 nesting levels are allowed.
To make a master reset call, specify the [,MRST] option. When a master reset call is made, bit-type PI/Os turned on by the local process are cleared to 0 on the following timing: the called process is terminated; Escape is executed; or the executable state is entered.
When the [,TUP] option is specified, the parallel timers occupied by the local process are forcibly timed out at execution of Process End or Escape or at shift to the executable state.
When the [,TRS] option is specified, the parallel timers occupied by the local process are reset at execution of Process End or Escape or at shift to the executable state.
When the option is not specified, the parallel timers continue measurement even after the process is terminated.

[Syntax]

Pxxx [, step-number] [, MRST] { [, TUP] }
{ [, TRS] }

[Sample programs containing Call ( )]

1. P1

   Process 1 is called in zone mode, starting from step 1. The parallel timers occupied by the called process continue measurement when Process End or Escape is executed or the process executable state is entered.

2. P2, 5, MERST

   Process 2 is called in master reset mode, starting from step 5. The parallel timers occupied by the called process continue measurement when Process End or Escape is executed or the process executable state is entered.

3. P3, TUP

   Process 3 is called in zone mode, starting from step 1. The parallel timers occupied by the called process are forcibly timed out when Process End or Escape is executed or the process executable state is entered.

4.14 Function

Function supplements the operation and data processing functions that are supported by Box. For details, see the chapter explaining application instructions.

[Syntax]

○ application-instruction-name parameter [, parameter] ~

4.15 Wait with Previous State Cleared

Wait with Previous State Cleared provides the same effect as Wait before the shift conditions are met. After these conditions are met, it functions differently. When the previous step is an ON statement or process call, Wait with Precious State Cleared turns off its PI/Os before making the program proceed to the next step. When the previous step is neither an ON statement nor a process call, Wait with Previous State Cleared is the same as Wait. (The program proceeds to the next step without PI/Os being turned off.)
When starting execution from this step through a branch, note that the previous state of the branch source is not cleared.
This step is conform to the SFC standard.

[Syntax]

┼* {condition-expression}
{WTxxx (expression [, SB] [, condition-expression]) }

THIS PAGE INTENTIONALLY LEFT BLANK.

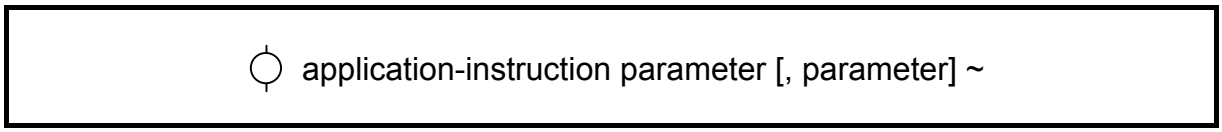
5 APPLICATION INSTRUCTIONS

5.1 Overview

HI-FLOW syntaxes support only assignment for word-length data, arithmetic operations, and logical operations as operation and data processing functions. To supplement this, HI-FLOW supports application instructions equivalent to ladder diagrams.

5.2 Usage

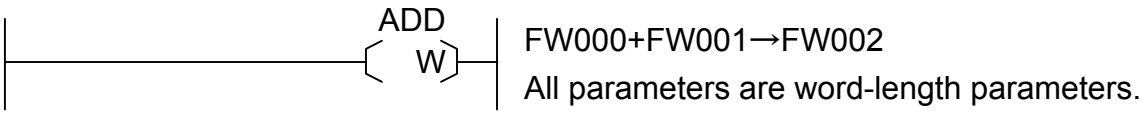
Program application instructions in the following format:



5.3 Parameters

Unlike ladder operation functions, HI-FLOW accepts difference in type between application instructions and parameters specified in them.

(Ladder)



(HI-FLOW)

For example

FW000	0001	→	After the application instruction is executed	→	0001
1	0000				0000
2	0002				0002
3	1111				0000
4	1111				0003

In general, three types of parameters are used: source (S), destination (D), and result (R). Three types of parameters, bit-type PI/O, word-type PI/O and constant, are provided. In HI-FLOW application instructions, the following four types of addressing modes can be specified for parameters:

1. Direct word-length specification: Specify parameters as they are.
2. Direct long-length specification: Enclose parameters in brackets ([ ]).
3. Indirect word-length specification: Prefix @ to the description in 1. above.
4. Indirect long-length specification: Prefix @ to the description in 2. above.

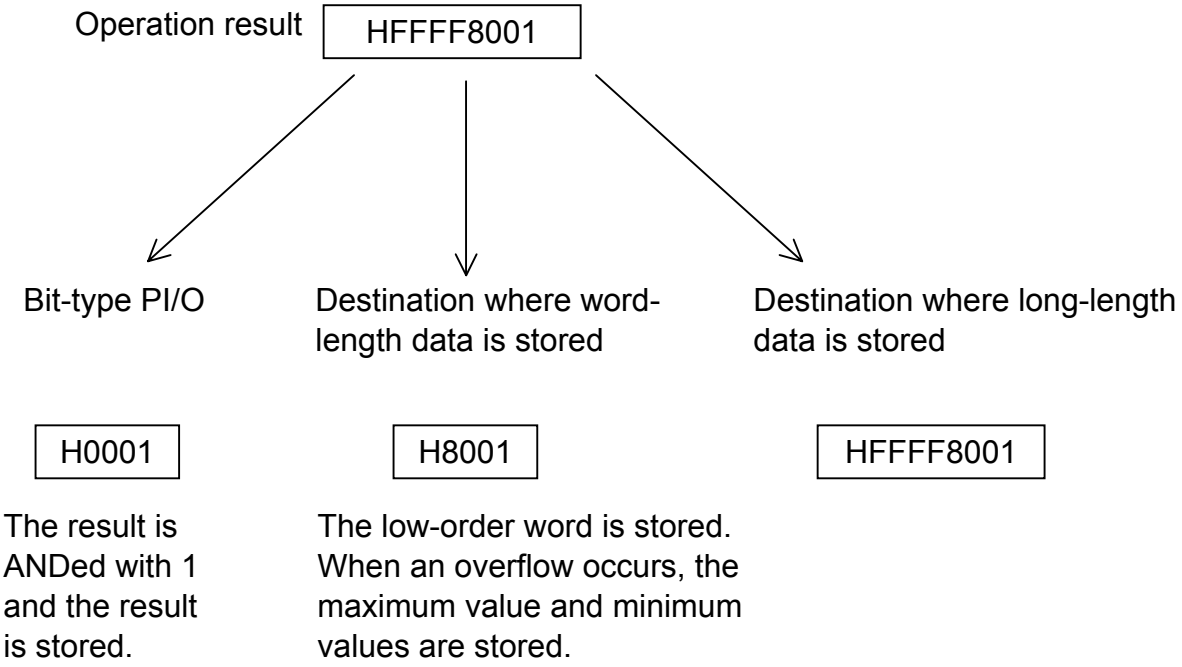
Addressing mode	Parameters		
	Bit-type PI/O	Word-type PI/O	Constant
	X000 Data 1 X001 Data 2 Data 1 and data 2 Data a Data b	FW000 Data 3 FW001 Data 4 Data 3 and data 4 Data c Data d	XXXX YYYYYYYY XXXX Data e Data f YYYYYYYY Data g Data h
1. Direct word-length	Result obtained by ANDing data 1 with data 1	Data 3	XXXX. For long-length data YYYYYYYY, only the low-order word is valid.
Examples	X000	FW000	1230 H20000000
2. Direct long-length	Result obtained by ANDing data 1 and data 2 with data 1	Data 3 and data 4	XXXX, YYYYYYYY. XXXX is handled as long-length data.
Examples	[X000]	[FW000]	[H1234] [H20000000]
3. Indirect word-length	Parameter error	Data c. When data 3 and data 4 are odd numbers, an error is detected.	For XXXX, data e is valid. For YYYYYYYY, data g is valid. When XXXX and YYYYYYYY are odd numbers, an error is detected.
Examples		@ FW000	@ HFFF0 @ H180000
4. Indirect long-length	Parameter error	Data c and data d. When data 3 and data 4 are odd numbers, an error is detected.	For XXXX, data e and data f are valid. For YYYYYYYY, data g and data h are valid. When XXXX and YYYYYYYY are odd numbers, an error is detected.
Examples		@ [FW000]	@ [HFFF0] @ [H180000]

5.4 Type Conversion for Operation

When parameter values are included for operation, they are all sign-extended to long-length data.

FW00 | 8001 | ——— During operation, this data is handled as HFFFF8001.

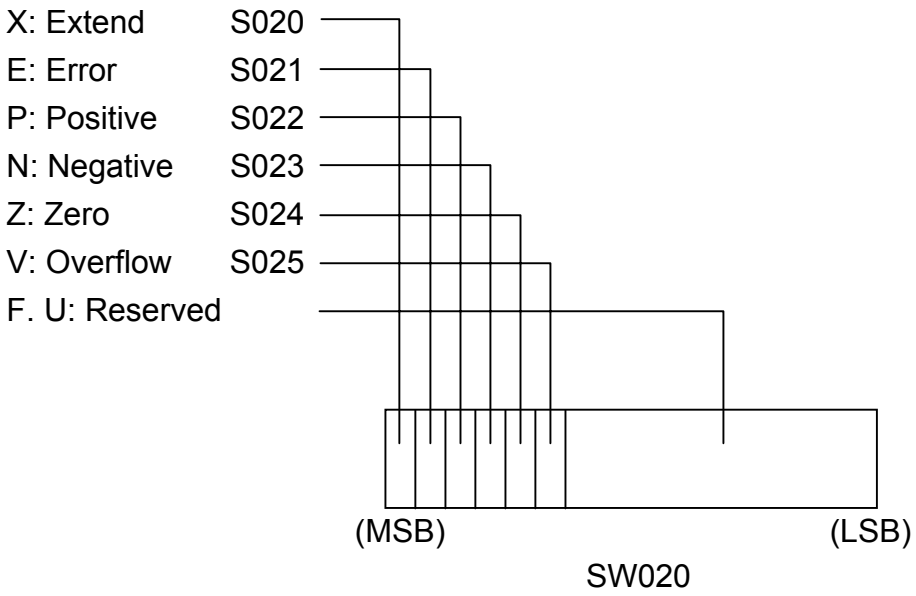
When an operation result is stored, its type is converted to conform to the data type of the destination.



5.5 System Error Flags

Flags are set in SW020, according to the execution result of the HI-FLOW application instruction.

Flags



Each flag is set according to the flag setting condition determined for each application instruction. When the following conditions hold, the pertinent flags are set for all application instructions.

Error flag: This flag is set when (1) the number of parameters used in the application instruction is invalid, (2) CPU memory is protected, (3) the address or PI/O specified by the result parameter (R) points to an address in a protected area, or (4) the specified PI/O is in error (for example, it cannot be used).

Overflow flag: This flag is set when the value of the operation result exceeds the range (word or long) specified by the result parameter (R). The limit of the size is set in the operation result.

Word-length Positive overflow: 7FFF
Negative overflow: 8000

Long-length Positive overflow: 7FFFFFFF
Negative overflow: 80000000

5.6 Explanation of Functions

This section details individual application instructions in the following format:

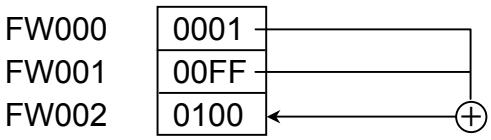
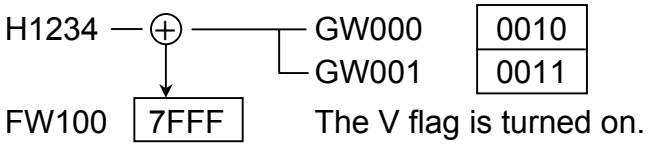
Application instruction name　Function name

ADD	ADD	
Function	Adds the contents of the source to those of the destination and stores the result in the area specified by the result parameter.	
Parameters and processing	○ ADD S, D, R S: Source D: Destination R: Result	S+D → R
Flags	The settings of the E and V flags change. The other flags are turned off.	
Processing time	0.41 ms	
Remarks		
Examples	○ ADD FW000, FW001, FW002 FW000 0001 FW001 00FF FW002 0100 ○ ADD H1234, [GW000], FW100 H1234 ⊕ GW000 0010 GW001 0011 FW100 7FFF The V flag is turned on.	
Valid parameters △: A parameter error is detected when an odd-numbered address is used.		

S, D, R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

- Processing of the application instruction is outlined.
- Processing is illustrated.
- A parameter list is shown.
- The flags whose contents change after execution of the application instruction are shown.
- The time required for processing by the S10/2α is shown.
- Notes to be followed are described.
- Main examples are shown.
- Parameters specifiable as source (S), destination (D), and result (R) parameters are shown.

○: Can be specified
△: Can be specified on conditions
×: Cannot be specified

ADD	ADD
Function	Adds the contents of the source to those of the destination and stores the result in the area specified by the result parameter.
Parameters and processing	○ ADD S, D, R　　S+D → R S: Source D: Destination R: Result
Flags	The settings of the E and V flags change.　The other flags are turned off.
Processing time	0.41 ms
Remarks	
Examples	○ ADD FW000, FW001, FW002  ○ ADD H1234, [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

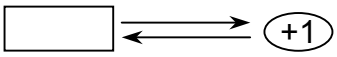
SUB	SUBTRACT
Function	Subtracts the contents of the destination from those of the source and stores the result in the area specified by the result parameter.
Parameters and processing	○ SUB S, D, R　　S-D → R S: Source D: Destination R: Result
Flags	The settings of the E and V flags change.　The other flags are turned off.
Processing time	0.41 ms
Remarks	
Examples	○ SUB FW000, FW001, FW002 FW000 0100 FW001 00FF FW002 0001 ⊖ ○ SUB H1234, [GW000], FW100 H1234 ⊖ GW000 0010 GW001 0011 FW100 8000　The V flag is turned on.

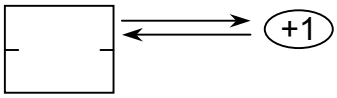
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

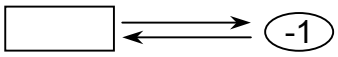
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

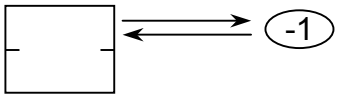
<table>
<tr><td>INC</td><td>+1 (INCREMENT)</td></tr>
<tr><td>Function</td><td>Increments the contents of the source by one.</td></tr>
<tr><td>Parameters and processing</td><td>INC S
S: Source
S+1 → S</td></tr>
<tr><td>Flags</td><td>The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.29 ms</td></tr>
<tr><td>Remarks</td><td></td></tr>
<tr><td>Examples</td><td>INC FW000
FW000
 

INC [GW000]
GW000
GW001
 

GW000 and GW001 are incremented by one, assuming that they are long-length variables.</td></tr>
<tr><td>Valid parameters

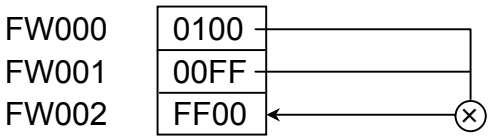
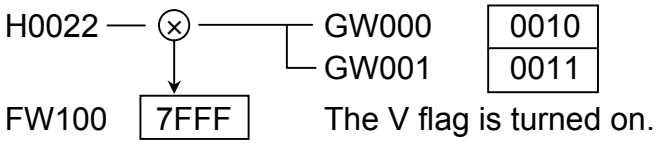
△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>DEC</td><td>-1 (DECREMENT)</td></tr>
<tr><td>Function</td><td>Decrements the contents of the source by one.</td></tr>
<tr><td>Parameters and processing</td><td>○ DEC S
S: Source
S-1 → S</td></tr>
<tr><td>Flags</td><td>The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.29 ms</td></tr>
<tr><td>Remarks</td><td></td></tr>
<tr><td>Examples</td><td>○ DEC FW000
FW000 

○ DEC [GW000]
GW000
GW001 

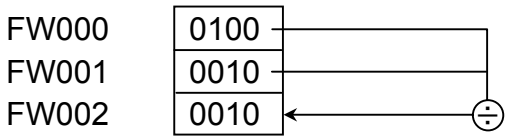
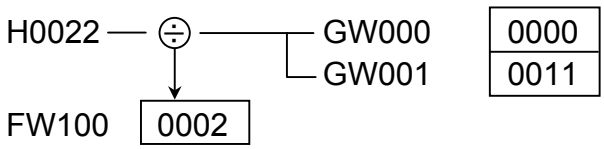
GW000 and GW001 are decremented by one, assuming that they are long-length variables.</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

MUL	MULTIPLY
Function	Multiplies the contents of the source by those of the destination and stores the result in the area specified by the result parameter.
Parameters and processing	○ MUL S, D, R　　S×D → R S: Source D: Destination R: Result
Flags	The settings of the E and V flags change.　The other flags are turned off.
Processing time	0.49 ms (when both S and D are word-length parameters) 1.35 ms (when either S or D is a long-length parameter)
Remarks	
Examples	○ MUL FW000, FW001, FW002  ○ MUL H22, [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

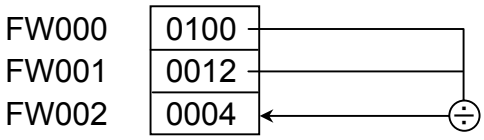
DIV	DIVIDE
Function	Divides the contents of the source by those of the destination and stores the quotient in the area specified by the result parameter.
Parameters and processing	○ DIV S, D, R S÷D → R S: Source D: Destination R: Result
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.49 ms (when both S and D are word-length parameters) 1.35 ms (when either S or D is a long-length parameter)
Remarks	When D = 0, the E flag is turned on and nothing is performed.
Examples	○ DIV FW000, FW001, FW002  ○ DIV H22, [GW000], FW100 

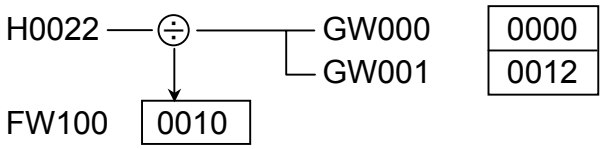
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

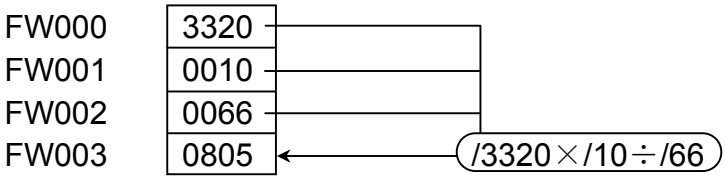
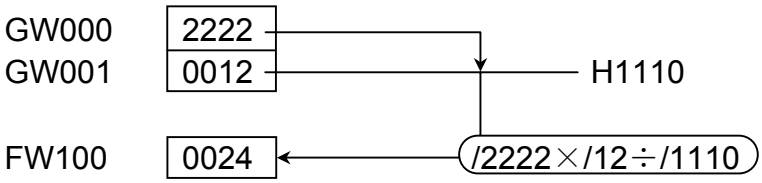
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>MOD</td><td>MOD (Remainder)</td></tr>
<tr><td>Function</td><td>Divides the contents of the source by those of the destination and stores the remainder in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ MOD S, D, R
S÷D → R
S: Source
D: Destination
R: Result</td></tr>
<tr><td>Flags</td><td>The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.50 ms (when both S and D are word-length parameters)
1.40 ms (when either S or D is a long-length parameter)</td></tr>
<tr><td>Remarks</td><td>When D = 0, the E flag is turned on and nothing is performed.
When an overflow occurs, R is set to 0.</td></tr>
<tr><td>Examples</td><td>○ MOD FW000, FW001, FW002
 

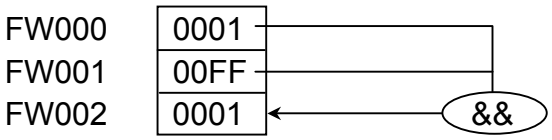
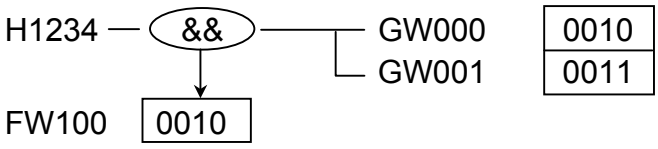
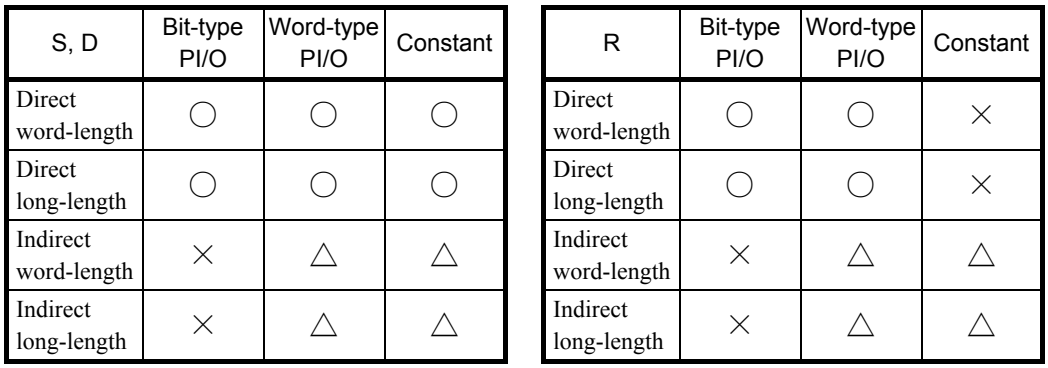
○ DIV H22, [GW000], FW100
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S, D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

SCL	SCALE CHANGE
Function	Converts the scale of the source with the contents of the destination and stores the result in the area specified by the result parameter.
Parameters and processing	○ SCL S, D1, D2, R S×D1÷D2 → R S: Source D1: Destination 1 D2: Destination 2 R: Result
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	1.38 ms
Remarks	When a multiplication overflow occurs, the overflow value is written in the area specified by the result parameter and processing is terminated. When D2=0, the E flag is turned on and nothing is performed. When an overflow occurs, R is set to 0.
Examples	○ SCL FW000, FW001, FW002, FW003  ○ SCL GW000, GW001, H1110, FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

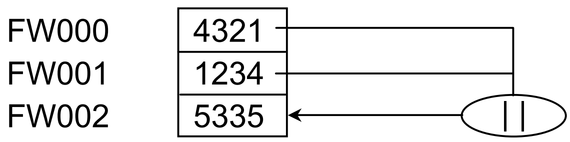
S, D1, D2	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

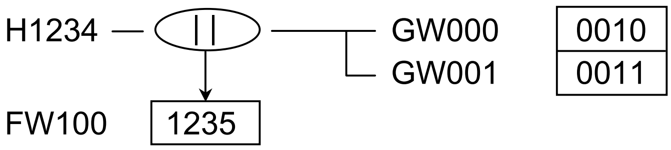
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

AND	AND
Function	ANDs the contents of the source with those of the destination and stores the logical product in the area specified by the result parameter.
Parameters and processing	○ AND S, D, R　　S && D → R S: Source D: Destination R: Result
Flags	The setting of the E flag changes.　The other flags are turned off.
Processing time	0.36 ms
Remarks	When R is a word-length parameter, the low-order word of the operation result is written.
Examples	○ AND FW000, FW001, FW002  ○ AND H1234, [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

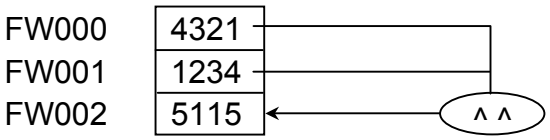
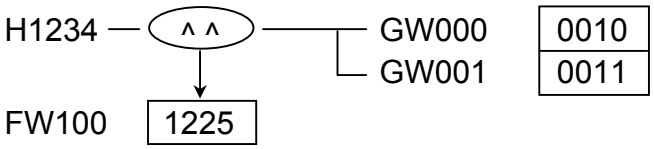
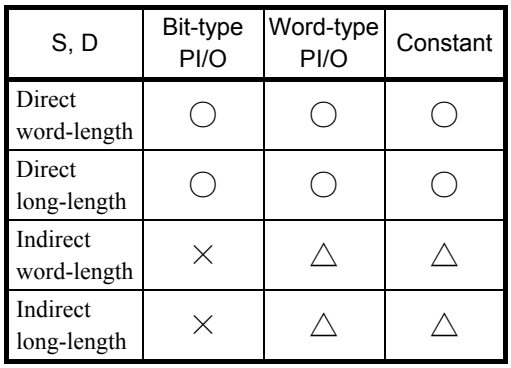
<table>
<tr><th>OR</th><th>OR</th></tr>
<tr><td>Function</td><td>ORs the contents of the source with those of the destination and stores the logical sum in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>OR S, D, R
S || D → R
S: Source
D: Destination
R: Result</td></tr>
<tr><td>Flags</td><td>The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.36 ms</td></tr>
<tr><td>Remarks</td><td>When R is a word-length parameter, the low-order word of the operation result is written.</td></tr>
<tr><td>Examples</td><td>OR FW000, FW001, FW002
 

OR H1234, [GW000], FW100
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>(see tables below)</td></tr>
</table>

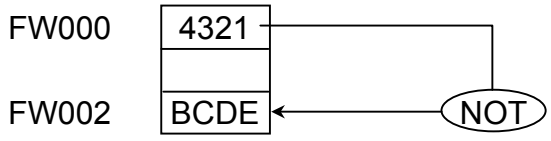
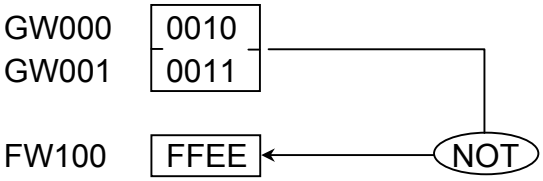
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

EOR	EXCLUSIVE OR
Function	EORs the contents of the source with those of the destination and stores the logical sum in the area specified by the result parameter.
Parameters and processing	○ EOR S, D, R S^^D → R S: Source D: Destination R: Result
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.36 ms
Remarks	When R is a word-length parameter, the low-order word of the operation result is written.
Examples	○ EOR FW000, FW001, FW002  ○ EOR H1234, [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

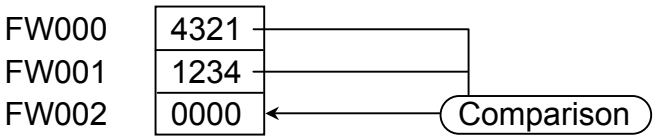
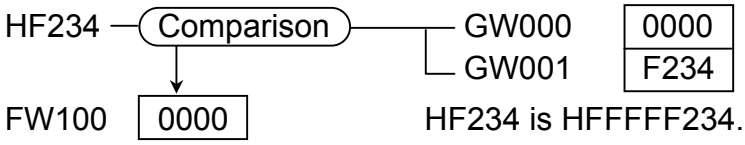
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

NOT	NOT (Negative)
Function	Inverts the contents (bits) of the source and stores the result in the area specified by the result parameter.
Parameters and processing	○ NOT S, R S: Source R: Result S (bit inversion) → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.36 ms
Remarks	
Examples	○ NOT FW000, FW002  ○ NOT [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

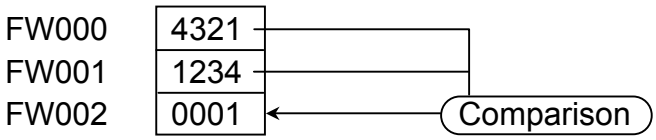
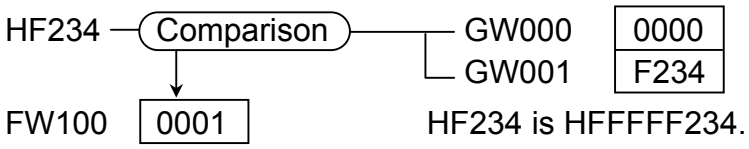
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

EQU	EQUAL
Function	Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when a match is found. Otherwise, stores 0 in the area.
Parameters and processing	○ EQU S, D, R S: Source D: Destination R: Result S = D 1 → R S ≠ D 0 → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.40 ms
Remarks	Word-length data is sign-extended to long-length data before it is compared.
Examples	○ EQU FW000, FW001, FW002  ○ EQU HF234, [GW000], FW100  HF234 is HFFFFF234.
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

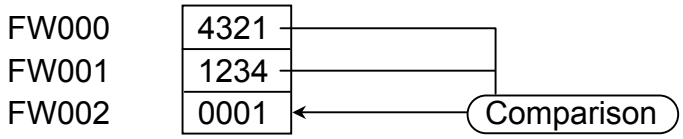
NEQ	NOT EQUAL
Function	Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when a match is not found. Otherwise, stores 0 in the area.
Parameters and processing	○ NEQ S, D, R S: Source D: Destination R: Result S ≠ D 1 → R S = D 0 → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.40 ms
Remarks	Word-length data is sign-extended to long-length data before it is compared.
Examples	○ NEQ FW000, FW001, FW002  ○ NEQ HF234, [GW000], FW100  HF234 is HFFFFF234.

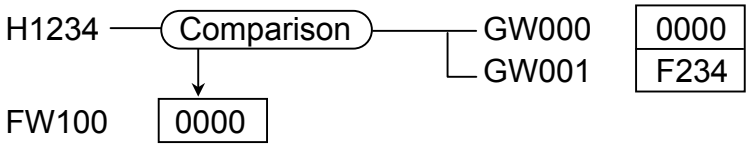
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

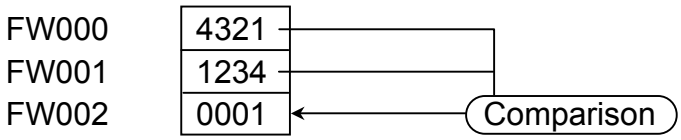
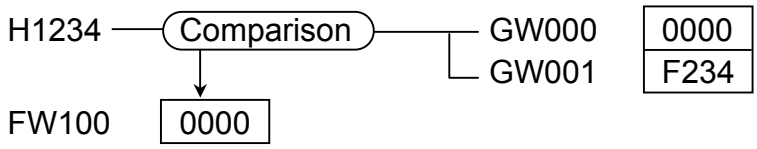
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>GT</td><td>GREATER THAN</td></tr>
<tr><td>Function</td><td>Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when the source is greater than the destination. Otherwise, stores 0 in the area.</td></tr>
<tr><td>Parameters and processing</td><td>○ GT S, D, R
S: Source
D: Destination
R: Result
S > D 1 → R
S ≦ D 0 → R</td></tr>
<tr><td>Flags</td><td>The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.40 ms</td></tr>
<tr><td>Remarks</td><td>Word-length data is sign-extended to long-length data before it is compared.</td></tr>
<tr><td>Examples</td><td>○ GT FW000, FW001, FW002
 

○ GT H1234, [GW000], FW100
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S, D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

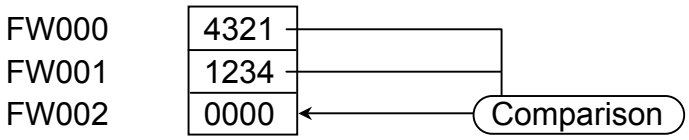
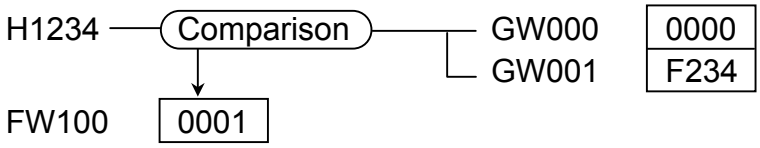
GE	GREATER OR EQUAL
Function	Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when the source is greater than or equal to the destination. Otherwise, stores 0 in the area.
Parameters and processing	○ GE S, D, R S: Source D: Destination R: Result S ≧ D 1 → R S < D 0 → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.40 ms
Remarks	Word-length data is sign-extended to long-length data before it is compared.
Examples	○ GE FW000, FW001, FW002  ○ GE H1234, [GW000], FW100 

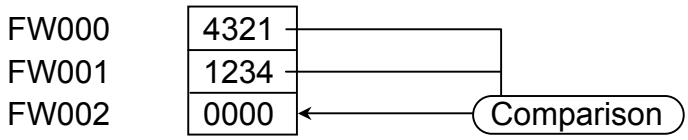
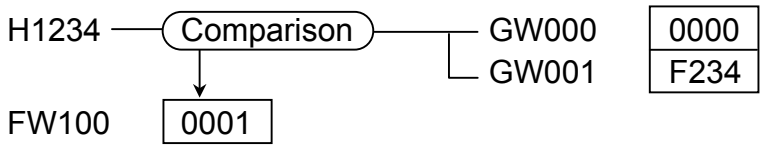
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

LT	LESS THAN
Function	Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when the source is smaller than the destination. Otherwise, stores 0 in the area.
Parameters and processing	○ LT S, D, R S: Source D: Destination R: Result S < D 1 → R S ≧ D 0 → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.40 ms
Remarks	Word-length data is sign-extended to long-length data before it is compared.
Examples	○ LT FW000, FW001, FW002  ○ LT H1234, [GW000], FW100 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	S, D: Bit-type PI/O ○, Word-type PI/O ○, Constant ○ Direct word-length ○ ○ ○ Direct long-length ○ ○ ○ Indirect word-length × △ △ Indirect long-length × △ △ R: Bit-type PI/O ○, Word-type PI/O ○, Constant × Direct word-length ○ ○ × Direct long-length ○ ○ × Indirect word-length × △ △ Indirect long-length × △ △

LE	LESS OR EQUAL
Function	Compares the contents of the source with those of the destination. Stores 1 in the area specified by the result parameter when the source is smaller than or equal to the destination. Otherwise, stores 0 in the area.
Parameters and processing	○ LE S, D, R S: Source D: Destination R: Result S ≦ D 1 → R S > D 0 → R
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.40 ms
Remarks	Word-length data is sign-extended to long-length data before it is compared.
Examples	○ LE FW000, FW001, FW002  ○ LE H1234, [GW000], FW100 

Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

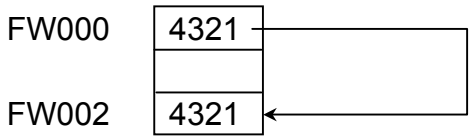
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

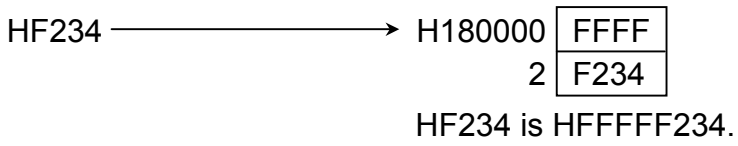
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>TST</td><td>TEST</td></tr>
<tr><td>Function</td><td>Tests the contents of the source and set the P, Z, and N flags.</td></tr>
<tr><td>Parameters and processing</td><td>TST S
S: Source
S > 0 : P = 1, Z = 0, N = 0
S = 0 : P = 0, Z = 1, N = 0
S < 0 : P = 0, Z = 0, N = 1</td></tr>
<tr><td>Flags</td><td>The settings of the E, P, Z, and N flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.17 ms</td></tr>
<tr><td>Remarks</td><td>Word-length data is sign-extended to long-length data before it is tested.</td></tr>
<tr><td>Examples</td><td>TST FW000
FW000 4321
FW020 2000 ← Test

TST [GW000]
GW000 FFFF
GW001 F234
SW020 1000 ← Test</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>MOV</td><td>MOVE</td></tr>
<tr><td>Function</td><td>Moves the contents of the source to the destination.</td></tr>
<tr><td>Parameters and processing</td><td>○ MOV S, D
S → D
S: Source
D: Destination</td></tr>
<tr><td>Flags</td><td>The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.28 ms</td></tr>
<tr><td>Remarks</td><td>When the size of data to be moved does not conform, its type is converted.</td></tr>
<tr><td>Examples</td><td>○ MOV FW000, FW002
 

○ MOV HF234, @ [H180000]
 

HF234 is HFFFFF234.</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

MOM	MOVE MULTI
Function	Moves the contents of the source to the destination n word-length or long-length data items at one time.
Parameters and processing	○ MOM S, n, D S: Source D: Destination n: Number of word-length or long-length data items transferred at one time S1 → D1 ↯ Sn → Dn
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.39+0.02×n ms
Remarks	When n is less than or equal to 0 or greater than 256, nothing is processed. When S is a constant, its value is converted to the type of D before the value is set. When S and D have different types, type conversion is performed.
Examples	○ MOM FW000, FW002 FW000 4321 FW002 4321 ○ MOM FW000, 2, @ [H180000] FW000 F234 FW001 0001 FW002 0000 FW003 FFFF H180000 FFFF 2 F234 4 0000 6 0001 HF234 is HFFFFF234. H0001 is H00000001.
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

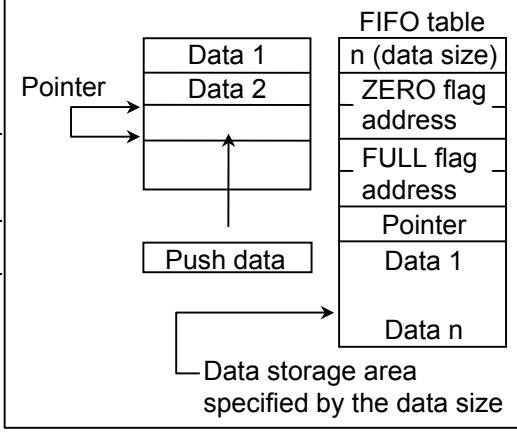
S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

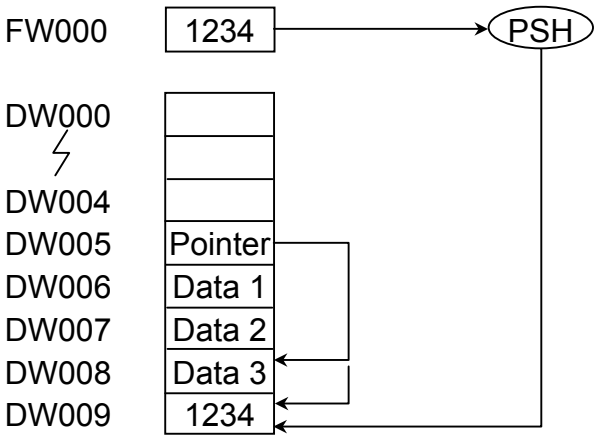
D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

EXC	EXCHANGE
Function	Exchanges the contents of the source and those of the destination with each other.
Parameters and processing	○ EXC S, D S: Source D: Destination S ⟷ D
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.31 ms
Remarks	When the size of data to be moved does not conform, its type is converted.
Examples	○ EXC FW000, FW002 FW000 1234 FW002 4321 ○ EXC @ H170000, @ [H180000] H170000 F234 ⟷ H180000 0010 2 0001 After exchange H170000 7FFF H180000 FFFF 2 F234
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

PSH	FIFO PUSH
Function	Pushes the contents of the source into the FIFO table. The FIFO table contains only word-length data.
Parameters and processing	○ PSH S, TB S: Source TB: First address of the FIFO table
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.31 ms
Remarks	When n is less than or equal to 0 or greater than 256, nothing is processed. When the value of the pointer is less than 0 or the data size is less than the value of the pointer, noting is processed. When the value of the pointer is equal to the data size, the FULL flag is turned on and nothing is processed. After the contents of the source are pushed, the contents of the pointer are incremented. When the pointer reaches n, the FULL flag is turned on. Otherwise, the ZERO and FULL flags are turned off. When TB is a constant, it is assumed to be the table address.
Examples	○ PSH FW000, DW000
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

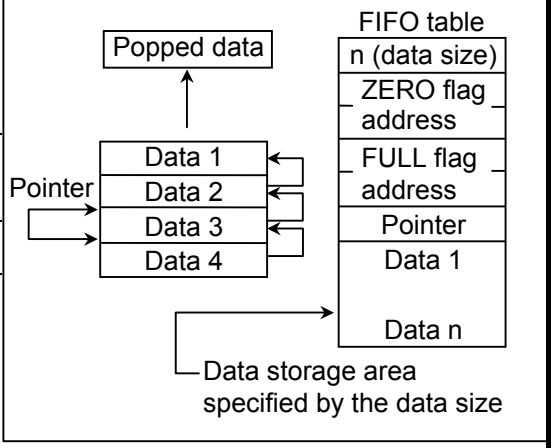


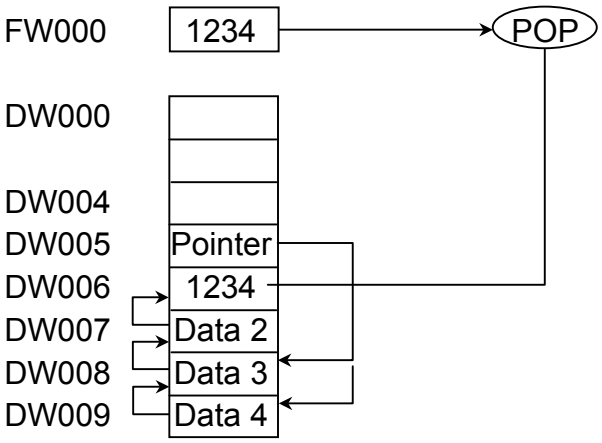


S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

TB	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	×	○	△
Direct long-length	×	○	△
Indirect word-length	×	△	△
Indirect long-length	×	△	△

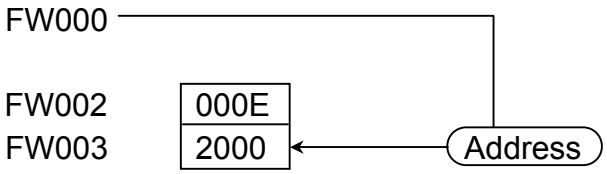
POP	FIFO POP
Function	Pops data from the FIFO table and stores the popped data in the destination. The FIFO table contains only word-length data.
Parameters and processing	○ POP TB, D D: Destination TB: First address of the FIFO table
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.32+0.01×n ms
Remarks	When n is less than or equal to 0 or greater than 256, nothing is processed. When the value of the pointer is less than 0 or the data size is less than the value of the pointer, noting is processed. When the value of the pointer is 0, the ZERO flag is turned on and nothing is processed. After data is popped, the contents of the pointer are decremented. When the pointer reaches 0, the ZERO flag is turned on. Otherwise, the ZERO and FULL flags are turned off. When TB is a constant, it is assumed to be the table address.
Examples	○ POP DW000, FW000
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

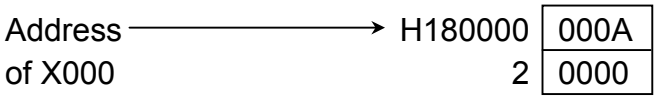




TB	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	×	○	△
Direct long-length	×	○	△
Indirect word-length	×	△	△
Indirect long-length	×	△	△

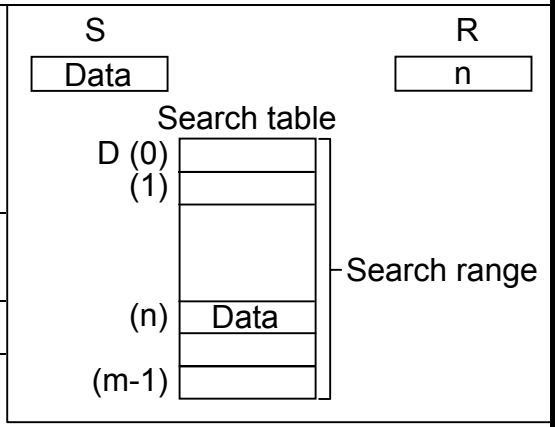
D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>AST</td><td>ADDRESS SET</td></tr>
<tr><td>Function</td><td>Transfers the address of the source to the destination. Valid only for PI/O.</td></tr>
<tr><td>Parameters and processing</td><td>○ AST S, D
Address of S → D
S: Source
D: Destination</td></tr>
<tr><td>Flags</td><td>The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.25 ms</td></tr>
<tr><td>Remarks</td><td>When word-length data is specified in D, the address is converted to word-length data.</td></tr>
<tr><td>Examples</td><td>○ AST FW000, [FW002]
 

○ AST X000, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

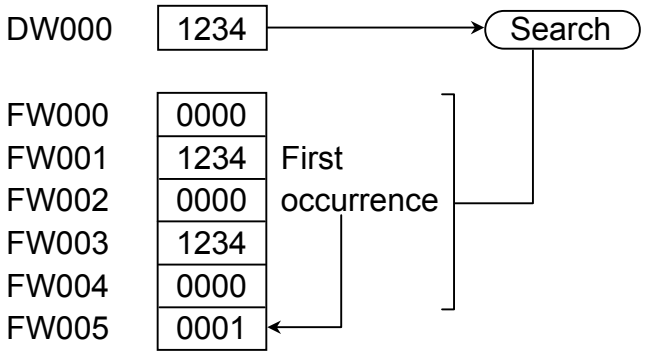
△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>×</td><td>×</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>×</td><td>×</td></tr>
</table>
<table>
<tr><th>D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

SCH	SEARCH
Function	Searches the destination for the contents of the source from the beginning of the destination before the specified area (m) is reached. Stores, in the area specified by the result parameter, the number (n) of the steps where the contents of the source are found.
Parameters and processing	○ SCH S, D, m, R S: Source D: Destination m: Number of steps to be searched R: Result
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.45+0.01×n ms
Remarks	When m is less than or equal to 0 or greater than 256, nothing is processed. A match occurs when the first occurrence of the contents of the source is encountered. If a match is not found within the search range, -1 is set in the area specified by the result parameter. When the type (long or word) of the data to be searched is different from that of data in the destination, an error is detected. The step numbers (n) in the destination start with 0.



Examples

○ SCH DW000, FW000, 5, FW005

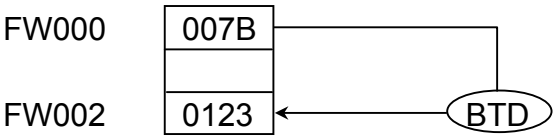


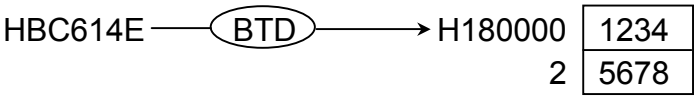
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

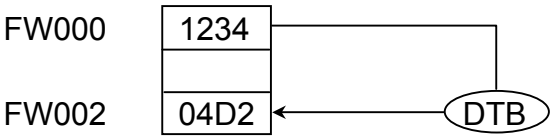
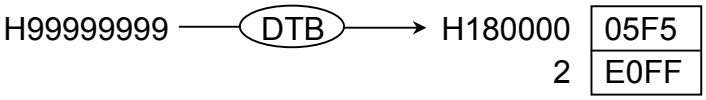
S, m	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

D, R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>BTD</td><td>BINARY → BCD CONVERSION</td></tr>
<tr><td>Function</td><td>Converts the contents of the source from the binary form to the BCD form and stores the result in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ BTD S, R
S: Source
R: Result
S (binary) → R (BCD)</td></tr>
<tr><td>Flags</td><td>The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>1.03 ms</td></tr>
<tr><td>Remarks</td><td>When S < 0, nothing is processed. In this case, the E flag is turned on and the V flag is turned off. If an overflow occurs, H9999 or H99999999 is set.</td></tr>
<tr><td>Examples</td><td>○ BTD FW000, FW002
 

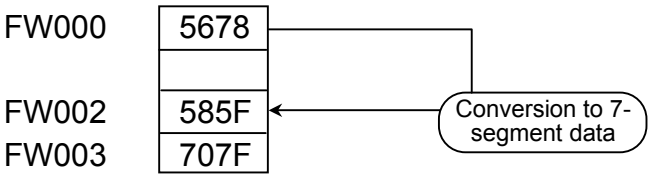
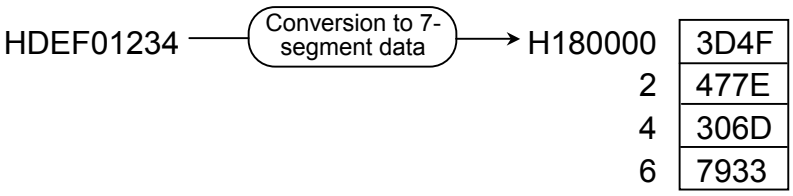
○ BTD HBC614E, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

DTB	BCD → BINARY CONVERSION
Function	Converts the contents of the source from the BCD form to the binary form and stores the result in the area specified by the result parameter.
Parameters and processing	○ DTB S, R S: Source R: Result S (BCD) → R (binary)
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.46 ms
Remarks	If any of A to F is used in S, the E flag is turned on and nothing is processed.
Examples	○ DTB FW000, FW002  ○ DTB H99999999, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

SEG	BINARY → 7-SEGMENT CONVERSION
Function	Converts the contents of the source from the binary form to 7-segment data and stores the result in the area specified by the result parameter.
Parameters and processing	○ SEG S, R S: Source R: Result S (binary) → R (7-segment data)
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.35 ms
Remarks	S is doubled in size and written in R.
Examples	○ SEG FW000, FW002  ○ SEG HDEF01234, @ [H180000]  7-segment correspondence table

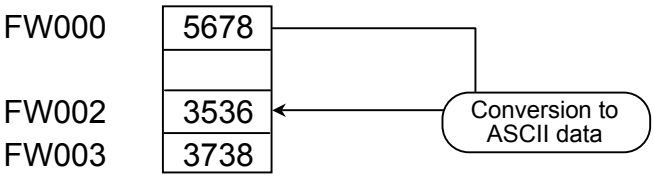
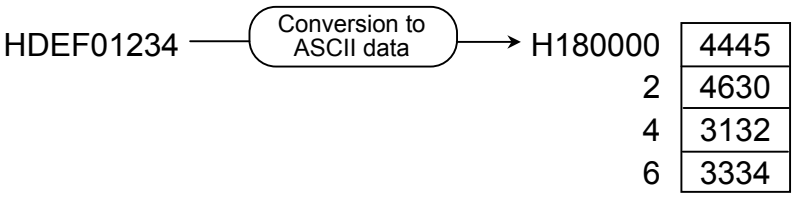
No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Data	7E	30	6D	79	33	5B	5F	70	7F	7B	77	1F	4E	3D	4F	47

Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

ASP	BINARY → ASCII CONVERSION (PACKED MODE)
Function	Converts the contents of the source from the binary form to ASCII data and stores the result in the area specified by the result parameter in packed mode.
Parameters and processing	○ ASP S, R S: Source R: Result S (binary) → R (ASCII, packed)
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.47 ms
Remarks	S is doubled in size and written in R.
Examples	○ ASP FW000, FW002  ○ ASP HDEF01234, @ [H180000]  Correspondence table between ASCII and binary data

Binary	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Data	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Valid parameters

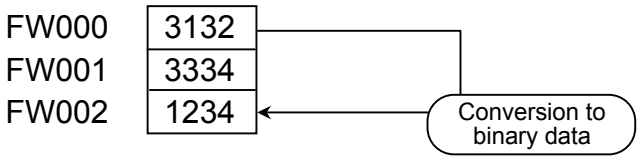
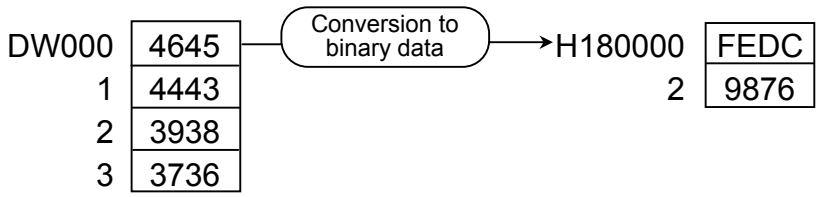
△: A parameter error is detected when an odd-numbered address is used.

S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>ASU</td><td>BINARY → ASCII CONVERSION (UNPACKED MODE)</td></tr>
<tr><td>Function</td><td>Converts the contents of the source from the binary form to ASCII data and stores the result in the area specified by the result parameter in unpacked mode.</td></tr>
<tr><td>Parameters and processing</td><td>○ ASU S, R
S (binary) → R (ASCII, unpacked)
S: Source
R: Result</td></tr>
<tr><td>Flags</td><td>The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.49 ms</td></tr>
<tr><td>Remarks</td><td>S is multiplied by four in size and written in R.</td></tr>
<tr><td>Examples</td><td>○ ASU FW000, FW002
FW001 5678
FW002 3035
FW003 3036
FW004 3037
FW005 3038
Conversion to ASCII data
○ ASU HDEF01234, @ [H180000]
HDEF01234 → Conversion to ASCII data → H180000 3044
2 3045
4 3046
6 3030
8 3031
A 3032
C 3033
E 3034
Correspondence table between ASCII and binary data
<table>
<tr><td>Binary</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr>
<tr><td>Data</td><td>30</td><td>31</td><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td></tr>
</table>
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

APB	ASCII → BINARY CONVERSION (PACKED MODE)
Function	Converts the contents of the source from ASCII data in packed mode to the binary form and stores the result in the area specified by the result parameter.
Parameters and processing	○ APB S, R S: Source R: Result S (ASCII, packed) → R (binary)
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.57 ms
Remarks	Data with the size in R multiplied by two is fetched from S and converted. If data in S contains any of H30 to H39 and H41 to H46, the E flag is turned on and nothing is processed.
Examples	○ APB FW000, FW002  ○ APB DW000, @ [H180000]  Correspondence table between ASCII and binary data

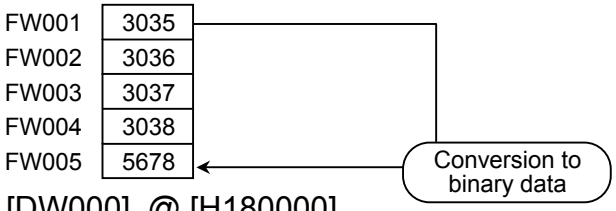
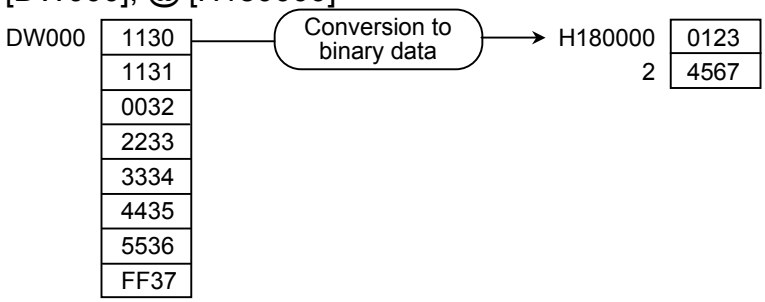
Binary	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	×	○	×
Direct long-length	×	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

AUB	ASCII → BINARY CONVERSION (UNPACKED MODE)
Function	Converts the contents of the source from ASCII data in unpacked mode to the binary form and stores the result in the area specified by the result parameter.
Parameters and processing	○ AUB S, R S: Source R: Result S (ASCII, unpacked) → R (binary)
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.57 ms
Remarks	Data with the size in R multiplied by four is fetched from S and converted. If data in S contains of any H30 to H39 and H41 to H46, the E flag is turned on and nothing is processed.
Examples	○ AUB FW001, FW002  ○ AUB [DW000], @ [H180000]  Correspondence table between ASCII and binary data

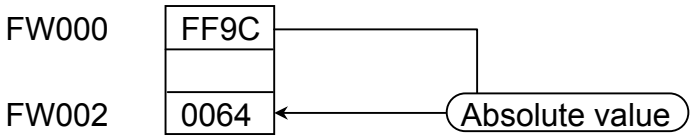
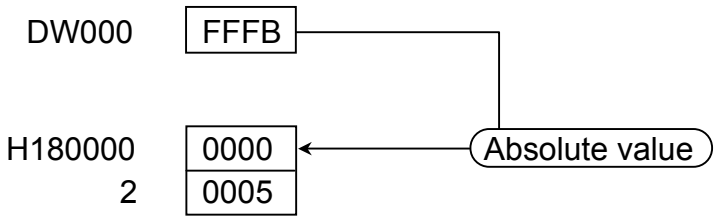
Binary	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
ASCII	30	31	32	33	34	35	36	37	38	39	41	42	43	44	45	46

Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

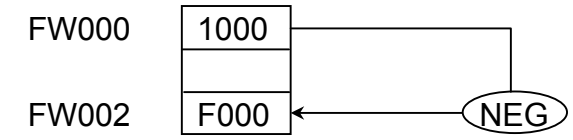
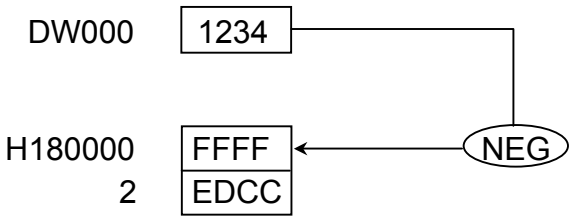
S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	×	○	×
Direct long-length	×	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

ABS	ABSOLUTE VALUE
Function	Stores the absolute value of the contents of the source in the area specified by the result parameter.
Parameters and processing	○ ABS S, R | S | → R S: Source R: Result
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.32 ms
Remarks	If an overflow occurs, H7FFFFFFF is set in the area specified by the result parameter.
Examples	○ ABS FW000, FW002  ○ ABS DW000, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

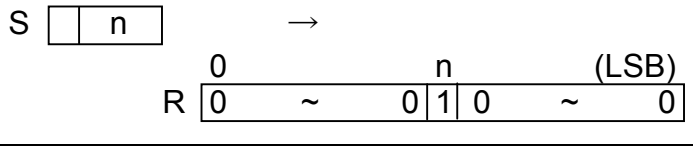
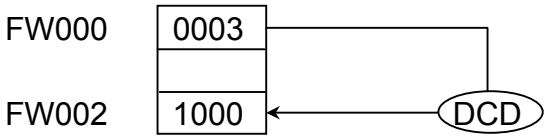
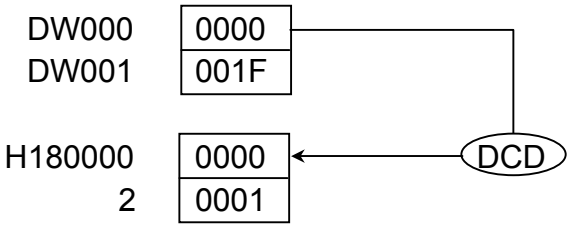
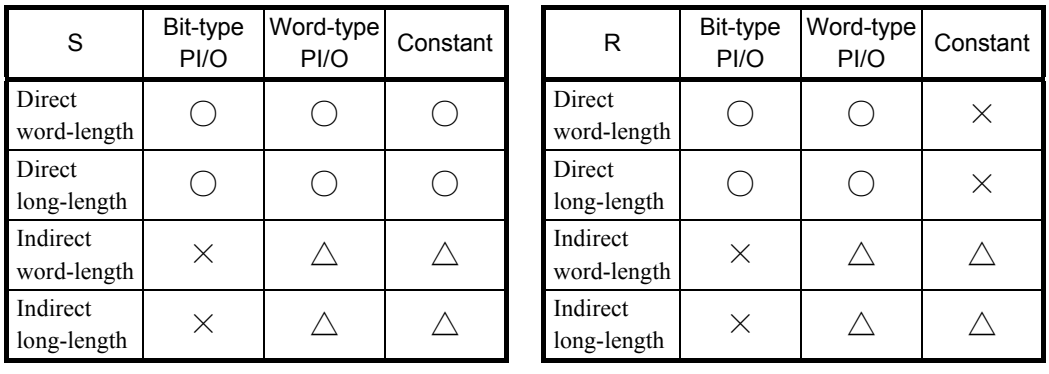
S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

NEG	SIGN SEARCH
Function	Sign-converts the contents of the source and stores the result in the area specified by the result parameter.
Parameters and processing	○ NEG S, R S: Source R: Result -S → R
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.32 ms
Remarks	If an overflow occurs, H7FFF or H7FFFFFFF is set in the area specified by the result parameter.
Examples	○ NEG FW000, FW002  ○ NEG DW000, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

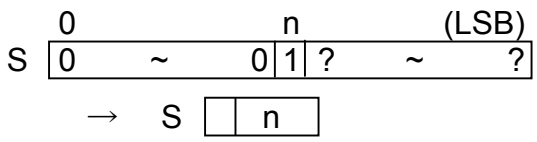
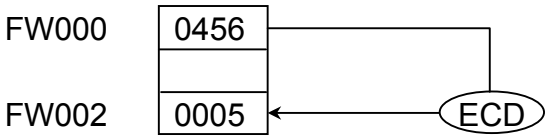
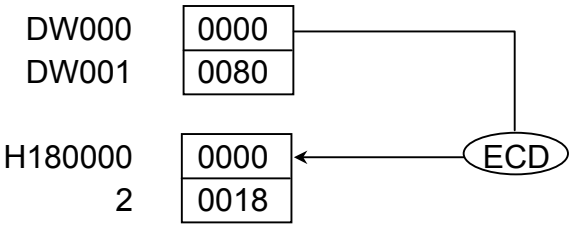
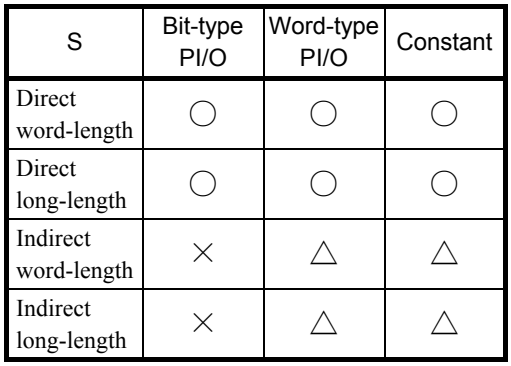
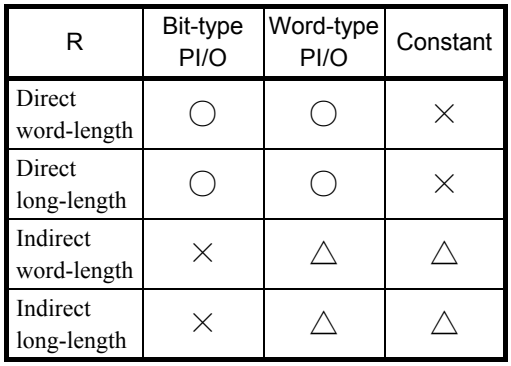
S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

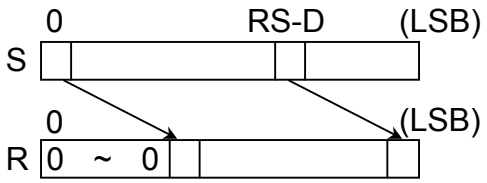
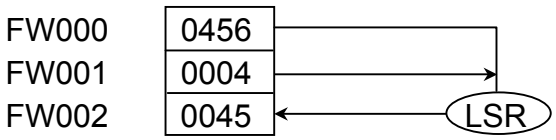
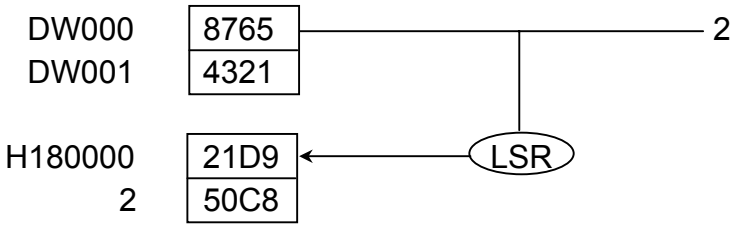
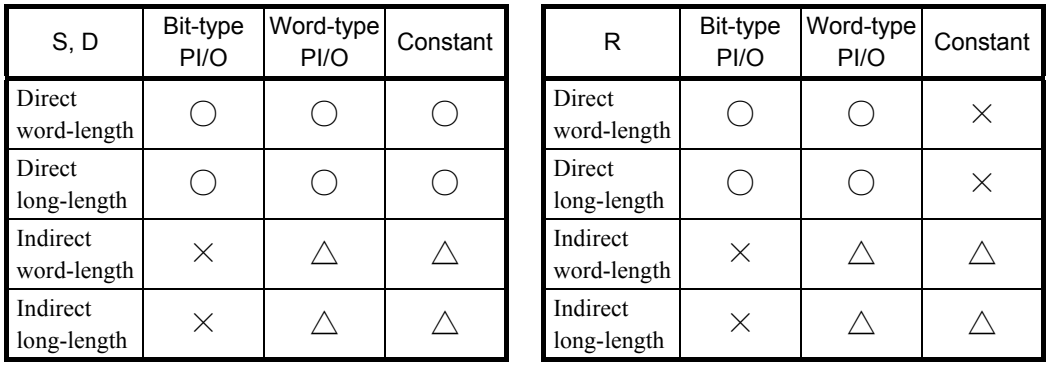
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

DCD	DECODE
Function	Decodes the contents of the source and stores the result in the area specified by the result parameter.
Parameters and processing	○ DCD S, R S: Source R: Result 
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.38 ms
Remarks	The valid bits in S depend on the data size specified in R. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ DCD FW000, FW002  ○ DCD [DW000], @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

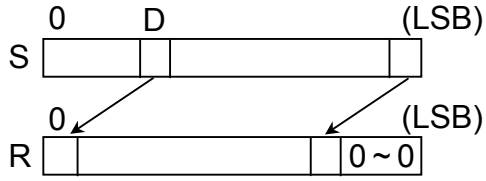
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

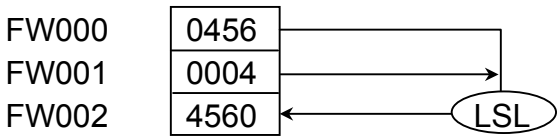
ECD	ENCODE
Function	Encodes the contents of the source and stores the result in the area specified by the result parameter.
Parameters and processing	○ ECD S, R S: Source R: Result 
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.38+0.01×n ms
Remarks	When S=0, nothing is processed. Only one bit is decoded. It is the first bit set to 1, when it is searched from the most significant bit (MSB).
Examples	○ ECD FW000, FW002  ○ ECD [DW000], @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	 

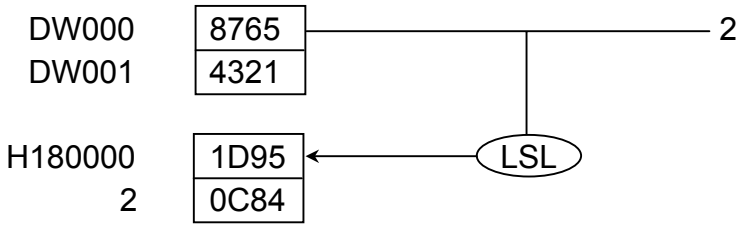
LSR	LOGICAL SHIFT RIGHT
Function	Shifts the contents of the source right by the contents of the destination, and stores the result in the area specified by the result parameter.
Parameters and processing	○ LSR S, D, R S: Source R: Result D: Destination  The length of RS depends on the data type; 15 bits for word-length or 31 bits for long-length.
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.37 ms
Remarks	The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ LSR FW000, FW001, FW002  ○ LSR [DW000], 2, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

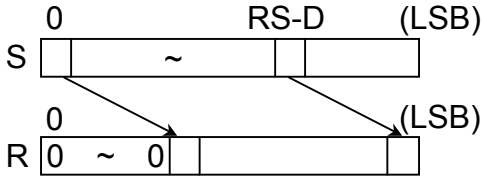
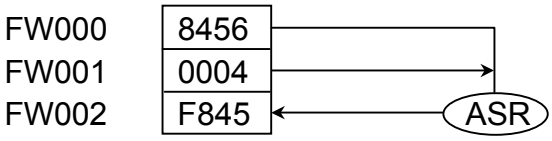
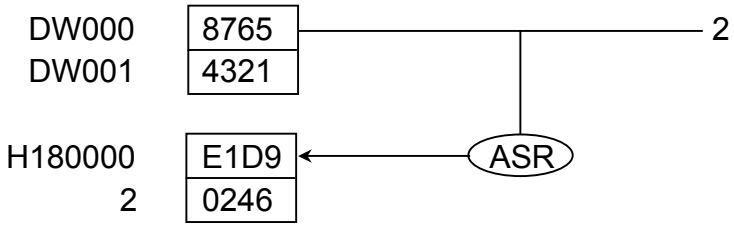
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>LSL</td><td colspan="2">LOGICAL SHIFT LEFT</td></tr>
<tr><td>Function</td><td colspan="2">Shifts the contents of the source left by the contents of the destination, and stores the result in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ LSL S, D, R
S: Source
R: Result
D: Destination</td><td> 

The length of LSB depends on the data type; 15 bits for word-length or 31 bits for long-length.</td></tr>
<tr><td>Flags</td><td colspan="2">The setting of the E flag changes. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td colspan="2">0.37 ms</td></tr>
<tr><td>Remarks</td><td colspan="2">The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.</td></tr>
<tr><td>Examples</td><td colspan="2">○ LSL FW000, FW001, FW002
 

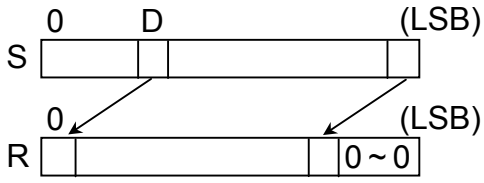
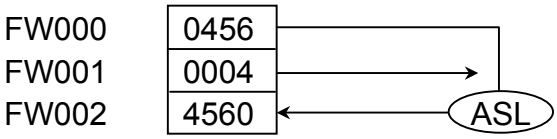
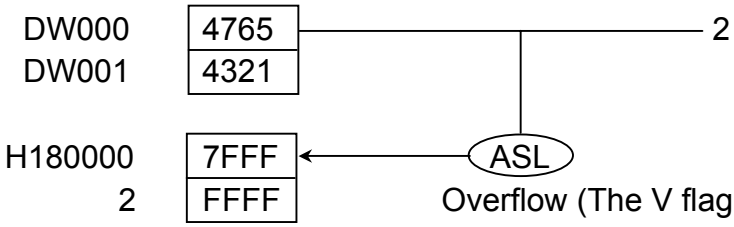
○ LSL [DW000], 2, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td colspan="2">
<table>
<tr><th>S, D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

ASR	ARITHMETIC SHIFT RIGHT
Function	Shifts the contents of the source right by the contents of the destination (the sign bit is retained), and stores the result in the area specified by the result parameter.
Parameters and processing	○ ASR S, D, R S: Source R: Result D: Destination  The length of RS depends on the data type; 15 bits for word-length or 31 bits for long-length.
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.43 ms
Remarks	When R is of the word-length type, the low-order word is set. The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ ASR FW000, FW001, FW002  ○ ASR [DW000], 2, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

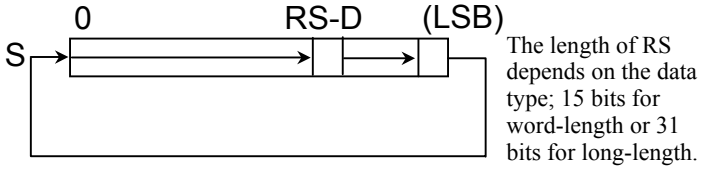
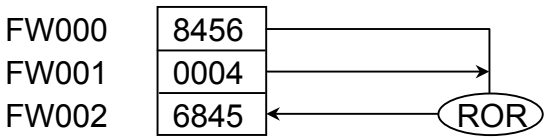
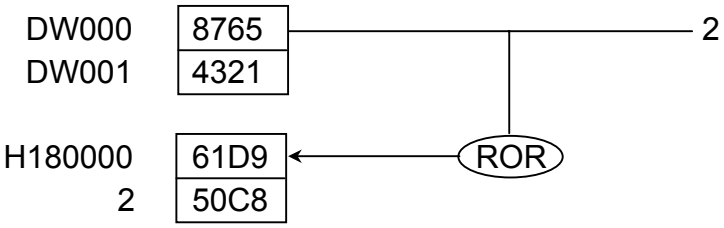
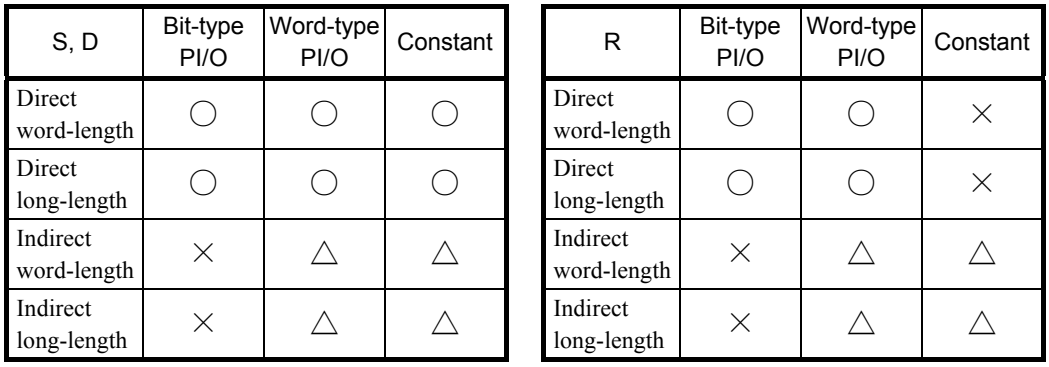
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

ASL	ARITHMETIC SHIFT LEFT
Function	Shifts the contents of the source left by the contents of the destination, and stores the result in the area specified by the result parameter. Sets the full-scale value if an overflow occurs.
Parameters and processing	○ ASL S, D, R S: Source R: Result D: Destination  The length of LSB depends on the data type; 15 bits for word-length or 31 bits for long-length.
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.41 ms
Remarks	The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ ASL FW000, FW001, FW002  ○ ASL [DW000], 2, @ [H180000]  Overflow (The V flag is turned on.)
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

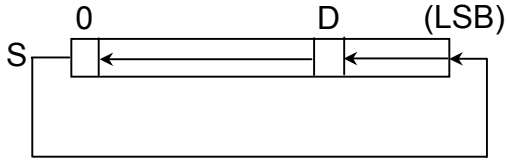
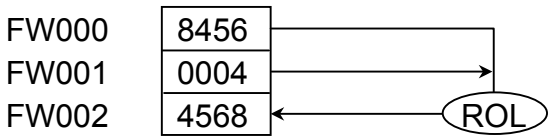
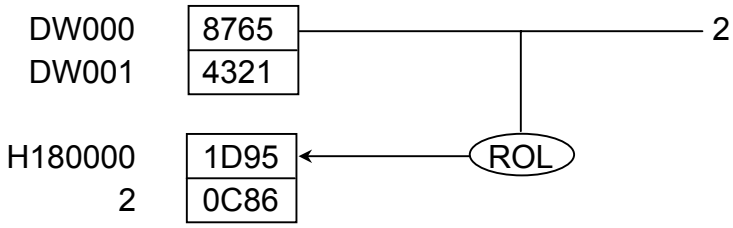
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

ROR	ROTATE RIGHT
Function	Rotates the contents of the source right by the contents of the destination, and stores the result in the area specified by the result parameter.
Parameters and processing	○ ROR S, D, R S: Source R: Result D: Destination 
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.37 ms
Remarks	The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ ROR FW000, FW001, FW002  ○ ROR [DW000], 2, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

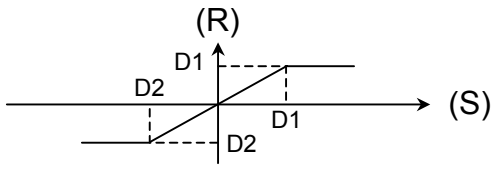
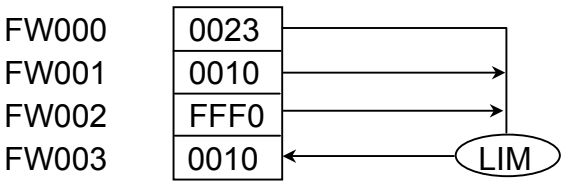
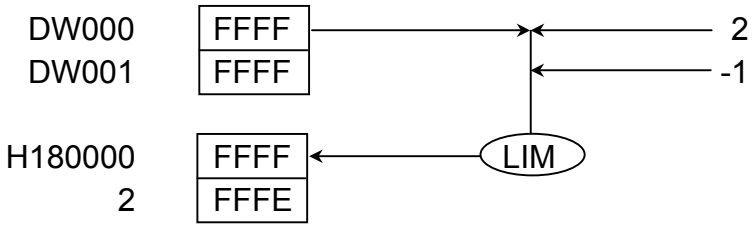
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

ROL	ROTATE LEFT
Function	Rotates the contents of the source left by the contents of the destination, and stores the result in the area specified by the result parameter.
Parameters and processing	○ ROL S, D, R S: Source R: Result D: Destination  The length of RS depends on the data type; 15 bits for word-length or 31 bits for long-length.
Flags	The setting of the E flag changes. The other flags are turned off.
Processing time	0.37 ms
Remarks	The valid bits in D depend on the data type in S. When word-length is specified, the low-order four bits are valid. When long-length is specified, the low-order five bits are valid.
Examples	○ ROL FW000, FW001, FW002  ○ ROL [DW000], 2, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

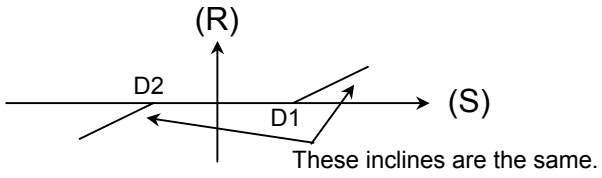
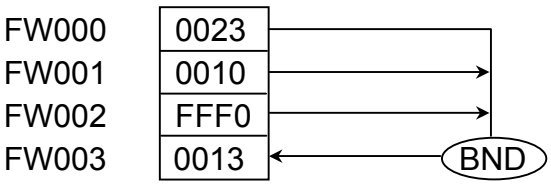
S, D	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

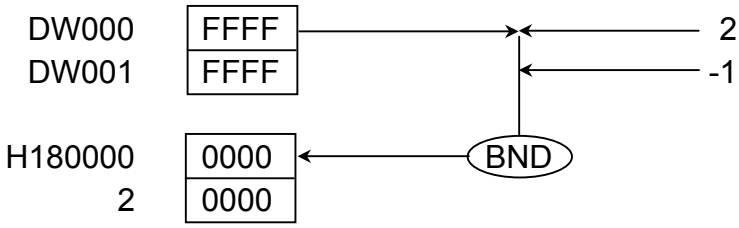
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

LIM	LIMITER
Function	Compares the contents of the source with the values of the specified boundaries (contents of destinations D1 and D2), and stores the result in the area specified by the result parameter.
Parameters and processing	○ LIM S, D1, D2, R S: Source R: Result D1, D2: Destination 
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.52 ms
Remarks	When D1 < D2, the E flag is turned on.
Examples	○ LIM FW000, FW001, FW002, FW003  ○ LIM [DW000], 2, -1, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

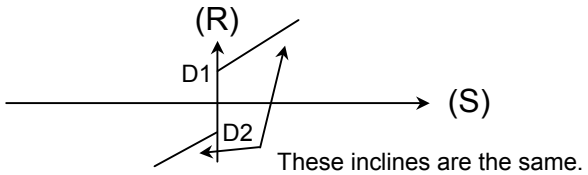
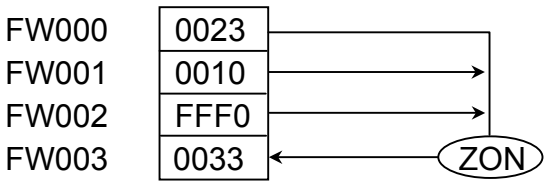
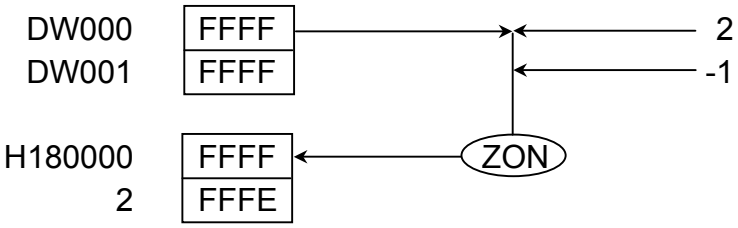
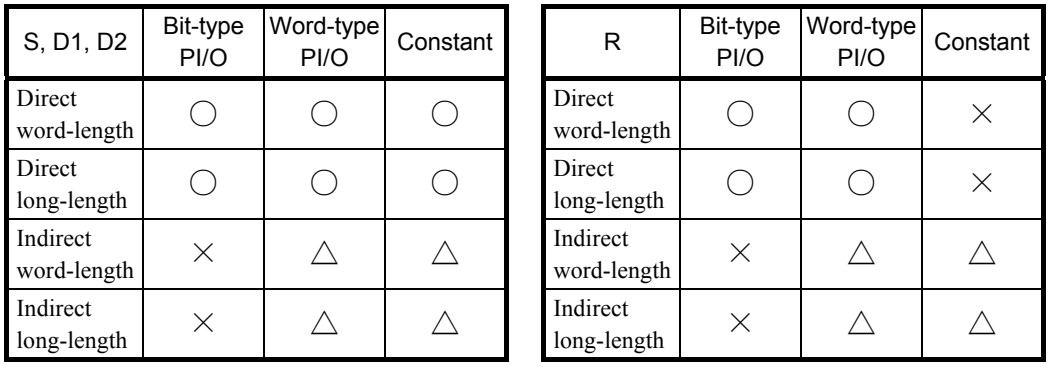
S, D1, D2	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>BND</td><td>DEAD BAND</td></tr>
<tr><td>Function</td><td>Compares the contents of the source with the values of the specified boundaries (contents of destinations D1 and D2), and stores the contents within the boundaries as a dead area (data 0) in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ BND S, D1, D2, R
S: Source
R: Result
D1, D2: Destination
 
</td></tr>
<tr><td>Flags</td><td>The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td>0.52 ms</td></tr>
<tr><td>Remarks</td><td>When D1 < D2, the E flag is turned on.</td></tr>
<tr><td>Examples</td><td>○ BND FW000, FW001, FW002, FW003
 

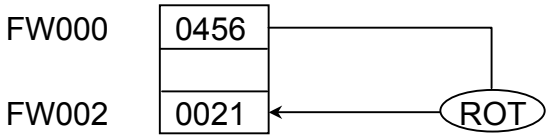
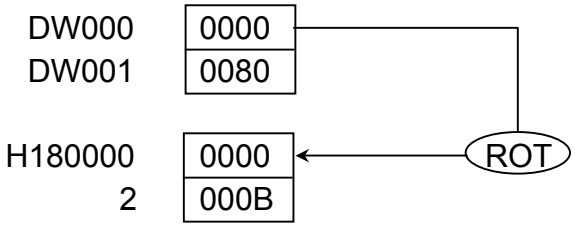
○ BND [DW000], 2, -1, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td>
<table>
<tr><th>S, D1, D2</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

ZON	DEAD ZONE
Function	Adds a bias (contents of destination D1 or D2) to the contents of the source according to its sign (positive or negative), and stores result in the area specified by the result parameter.
Parameters and processing	○ ZON S, D1, D2, R S: Source R: Result D1, D2: Destination 
Flags	The settings of the E and V flags change. The other flags are turned off.
Processing time	0.52 ms
Remarks	When D1 < D2, the E flag is turned on.
Examples	○ ZON FW000, FW001, FW002, FW003  ○ ZON [DW000], 2, -1, @ [H180000] 
Valid parameters △: A parameter error is detected when an odd-numbered address is used.	

S, D1, D2	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

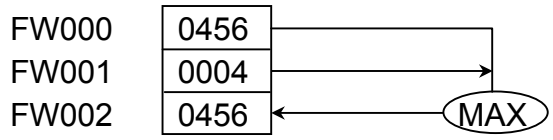
ROT	ROOT	
Function	Stores the integer part of the root obtained from the contents of the source in the area specified by the result parameter.	
Parameters and processing	○ ROT S, R S: Source R: Result	When S ≧ 0: Root of S → R When S < 0: 0 → R
Flags	The settings of the E and V flags change. The other flags are turned off.	
Processing time	0.77 ms	
Remarks		
Examples	○ ROT FW000, FW002  ○ ROT [DW000], @ [H180000]  	

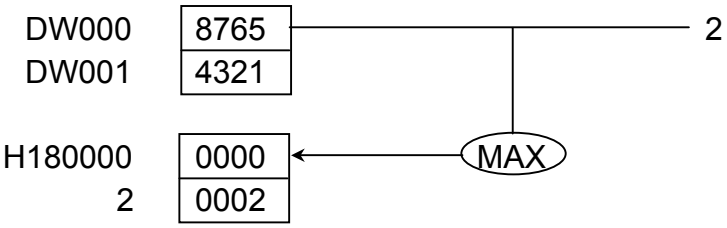
Valid parameters

△: A parameter error is detected when an odd-numbered address is used.

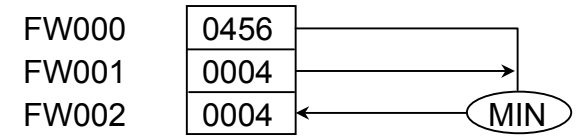
S	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	○
Direct long-length	○	○	○
Indirect word-length	×	△	△
Indirect long-length	×	△	△

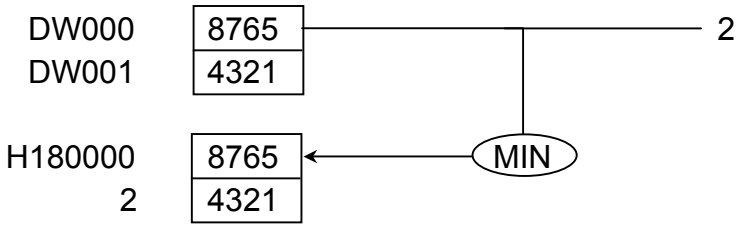
R	Bit-type PI/O	Word-type PI/O	Constant
Direct word-length	○	○	×
Direct long-length	○	○	×
Indirect word-length	×	△	△
Indirect long-length	×	△	△

<table>
<tr><td>MAX</td><td colspan="2">MAXIMUM</td></tr>
<tr><td>Function</td><td colspan="2">Compares the contents of the source with those of the destination, and stores the larger value in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ MAX S, D, R
S: Source
R: Result
D: Destination</td><td>When S ≧ D: S → R
When S < D: D → R</td></tr>
<tr><td>Flags</td><td colspan="2">The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td colspan="2">0.42 ms</td></tr>
<tr><td>Remarks</td><td colspan="2"></td></tr>
<tr><td>Examples</td><td colspan="2">○ MAX FW000, FW001, FW002
 

○ MAX [DW000], 2, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

△: A parameter error is detected when an odd-numbered address is used.</td><td colspan="2">
<table>
<tr><th>S, D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>MIN</td><td colspan="2">MINIMUM</td></tr>
<tr><td>Function</td><td colspan="2">Compares the contents of the source with those of the destination, and stores the smaller value in the area specified by the result parameter.</td></tr>
<tr><td>Parameters and processing</td><td>○ MIN S, D, R
S: Source
R: Result
D: Destination</td><td>When S ≦ D: S → R
When S > D: D → R</td></tr>
<tr><td>Flags</td><td colspan="2">The settings of the E and V flags change. The other flags are turned off.</td></tr>
<tr><td>Processing time</td><td colspan="2">0.42 ms</td></tr>
<tr><td>Remarks</td><td colspan="2"></td></tr>
<tr><td>Examples</td><td colspan="2">○ MIN FW000, FW001, FW002
 

○ MIN [DW000], 2, @ [H180000]
 
</td></tr>
<tr><td>Valid parameters

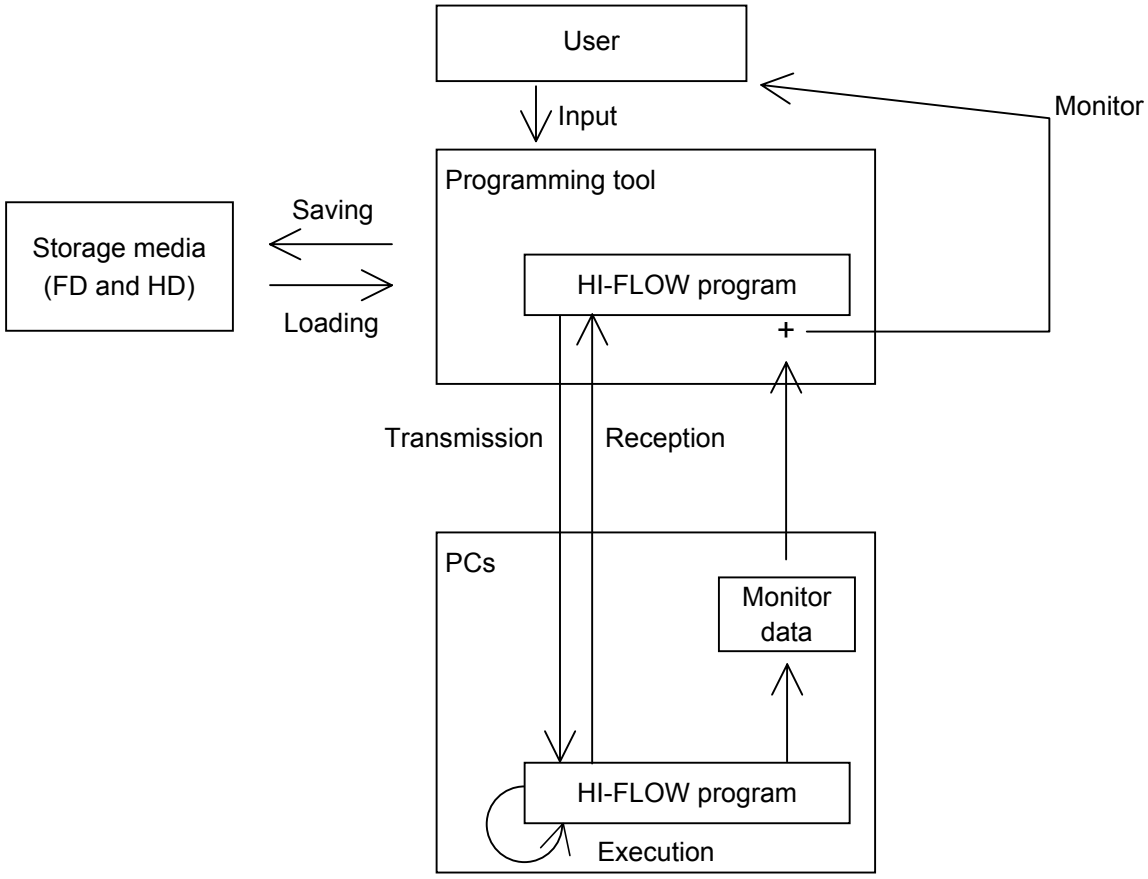
△: A parameter error is detected when an odd-numbered address is used.</td><td colspan="2">
<table>
<tr><th>S, D</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>○</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
<table>
<tr><th>R</th><th>Bit-type PI/O</th><th>Word-type PI/O</th><th>Constant</th></tr>
<tr><td>Direct word-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Direct long-length</td><td>○</td><td>○</td><td>×</td></tr>
<tr><td>Indirect word-length</td><td>×</td><td>△</td><td>△</td></tr>
<tr><td>Indirect long-length</td><td>×</td><td>△</td><td>△</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>CLR</td><td>CLEAR</td></tr>
<tr><td>Function</td><td>Clears a specified I/O area. TCLR, UCLR, and CCLR also clear the respective measured value areas.</td></tr>
<tr><td>Parameters and processing</td><td>Name S
Name: CLR application instruction name
S: Source (Top of the specified I/O area.)</td></tr>
<tr><td>Flags</td><td>All flags are set to 0.</td></tr>
<tr><td>Processing time</td><td>See the table below.</td></tr>
<tr><td>Remarks</td><td></td></tr>
<tr><td>Explanation</td><td>
<table>
<tr><th>Name</th><th>Function</th><th>Processing time</th></tr>
<tr><td>XCLR</td><td>Clears X000 to XFFF.</td><td>0.92 ms</td></tr>
<tr><td>YCLR</td><td>Clears Y000 to YFFF.</td><td>0.92 ms</td></tr>
<tr><td>GCLR</td><td>Clears G000 to GFFF.</td><td>0.92 ms</td></tr>
<tr><td>RCLR</td><td>Clears R000 to RFFF.</td><td>0.92 ms</td></tr>
<tr><td>KCLR</td><td>Clears K000 to KFFF.</td><td>0.92 ms</td></tr>
<tr><td>TCLR</td><td>Clears T000 to T3FF and also clears the measured area of T.</td><td>2.46 ms</td></tr>
<tr><td>UCLR</td><td>Clears U000 to U3FF and also clears the measured area of U.</td><td>1.70 ms</td></tr>
<tr><td>CCLR</td><td>Clears C000 to C3FF and also clears the measured area of C.</td><td>1.70 ms</td></tr>
<tr><td>VCLR</td><td>Clears V000 to VFFF.</td><td>0.92 ms</td></tr>
<tr><td>ECLR</td><td>Clears E000 to EFFF.</td><td>0.92 ms.</td></tr>
<tr><td>FCLR</td><td>Clears S020 to S027.</td><td>0.15 ms</td></tr>
<tr><td>JCLR</td><td>Clears J000 to JFFF.</td><td>0.92 ms</td></tr>
<tr><td>QCLR</td><td>Clears Q000 to QFFF.</td><td>0.92 ms</td></tr>
<tr><td>HHCLR</td><td>Clears HH000 to HH1FF.</td><td>1.70 ms</td></tr>
</table>
</td></tr>
<tr><td>Examples</td><td>XCLR X000
HHCLR HH000</td></tr>
</table>

SUPPLEMENT

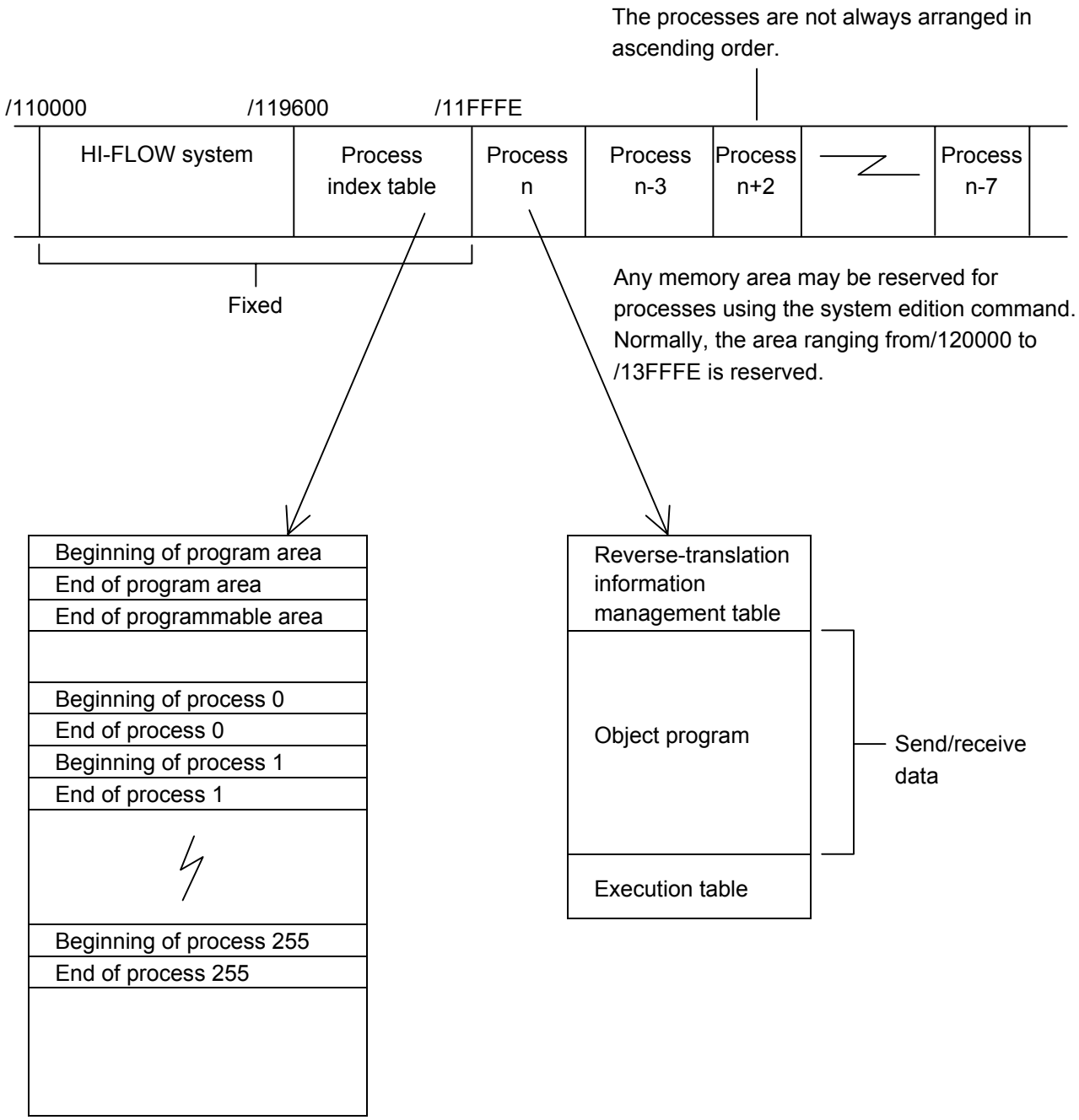
Supplement 1　Work Flow Based on HI-FLOW Program

A HI-FLOW program is created using a programming tool such as a personal computer and executed by the PCs.　When the execution result is to be monitored, minimum necessary data is received from the PCs, and it is combined with the program in the tool and output.　This aims at increasing the monitor speed by decreasing the amount of communication.
The storage media such as FD and HD are also used to save and load the created program.



Supplement 2 PCs Memory

The HI-FLOW program to be executed on the PCs are placed in the area shown below. The program is actually stored in the memory of the PC, as shown in the memory map below.



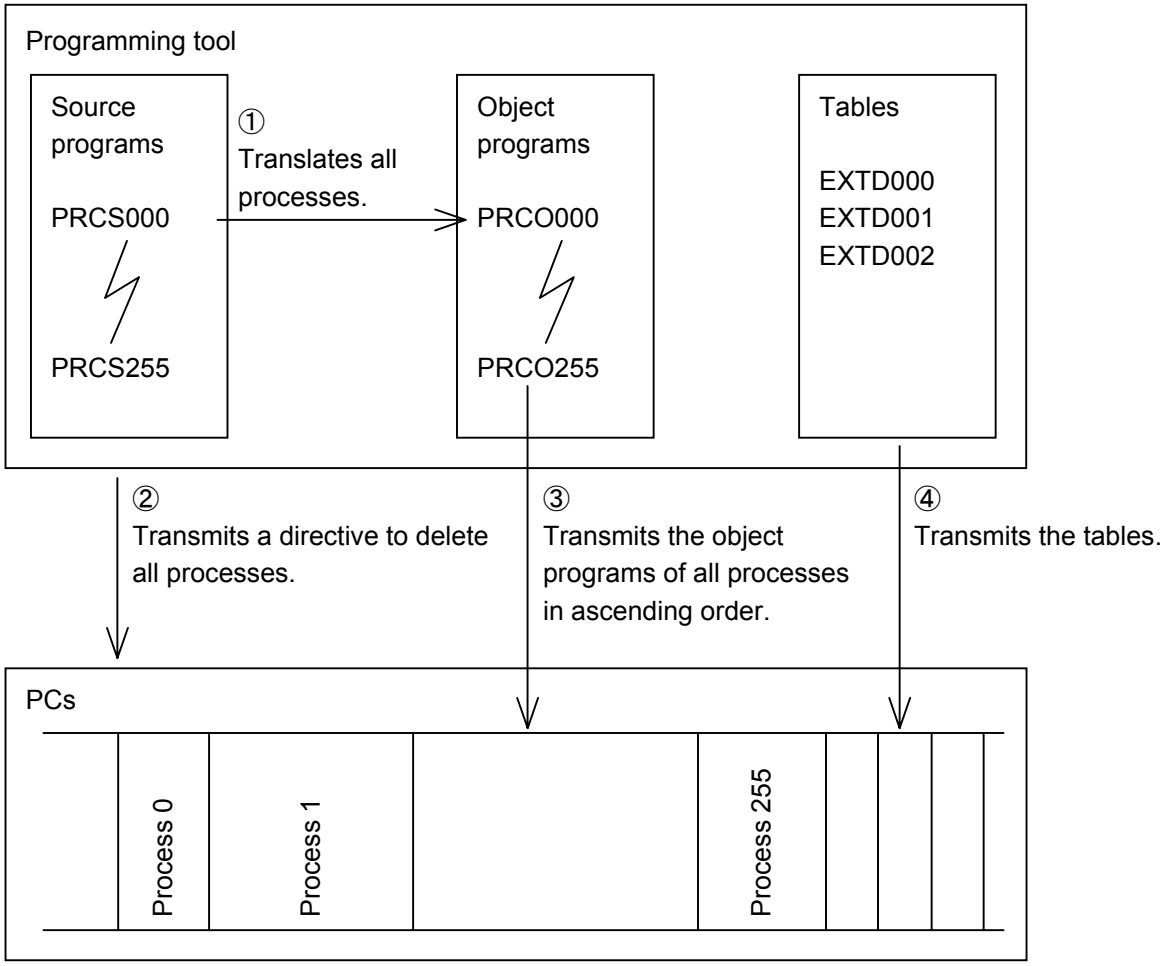
Supplement 3 Online Mode

"Offline" means that the text to be edited is made into a program of the programming tool regardless of the contents of PCs memory.

"Online" means that the text to be edited or monitored is made into a program for the PCs. When the PCs are placed under monitoring, not all necessary data has been read from the PCs since it takes a long time to do so by communication. For this reason, it is necessary that the program of the tool matches that of the PCs. This matching is enabled by transmission or reception.

A HI-FLOW program is self-contained in a process. Therefore, if only one process contains a matching program, the process can be subjected to editing or monitoring. Transmission and reception of all processes or just one process are available for saving time and many other purposes.

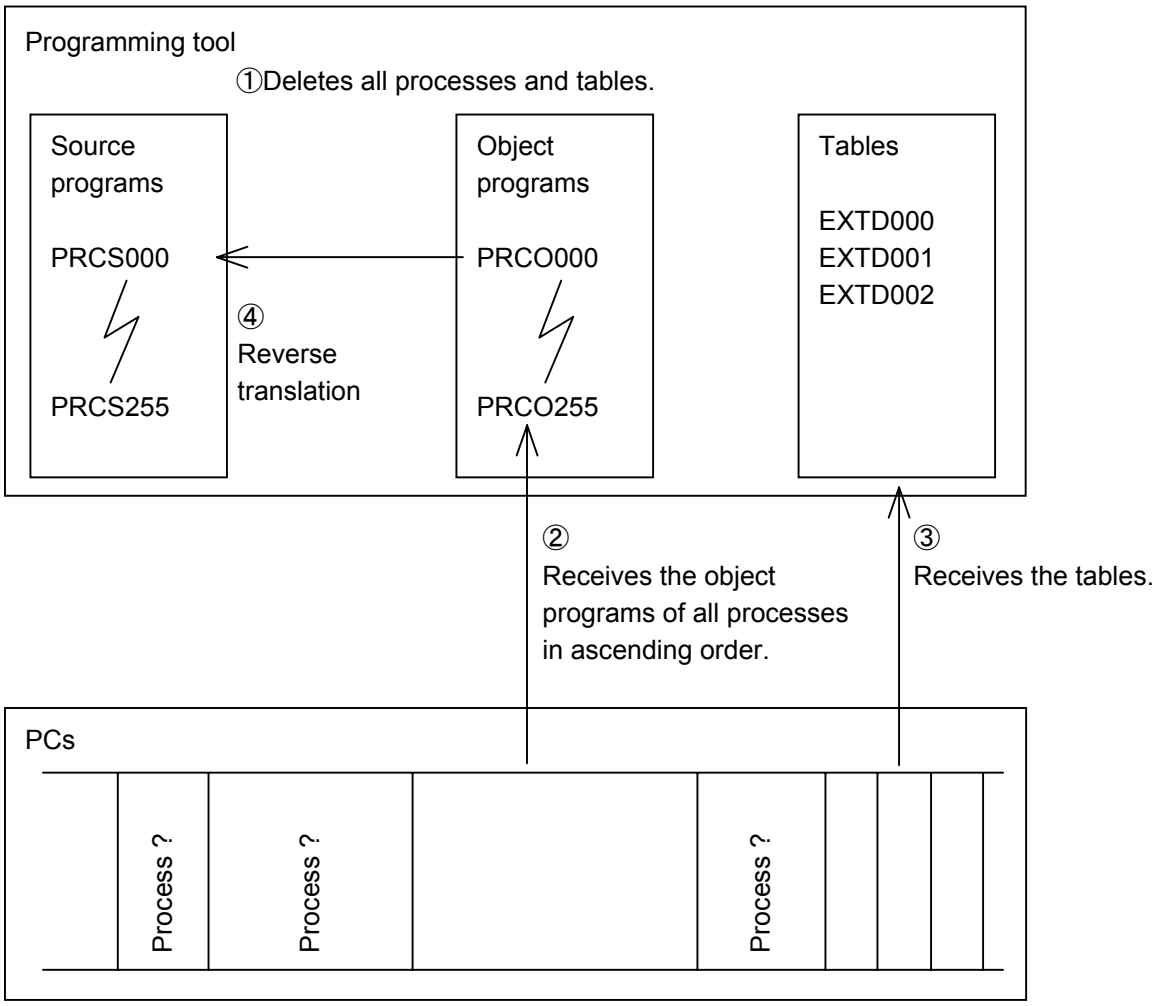
(1) All-processes transmission
The following shows the data flow when transmitting all HI-FLOW programs on the tool to the PCs at one time.



After all the processes have been transmitted, the processes and tables in memory are arranged in ascending order.

(2) All-processes reception

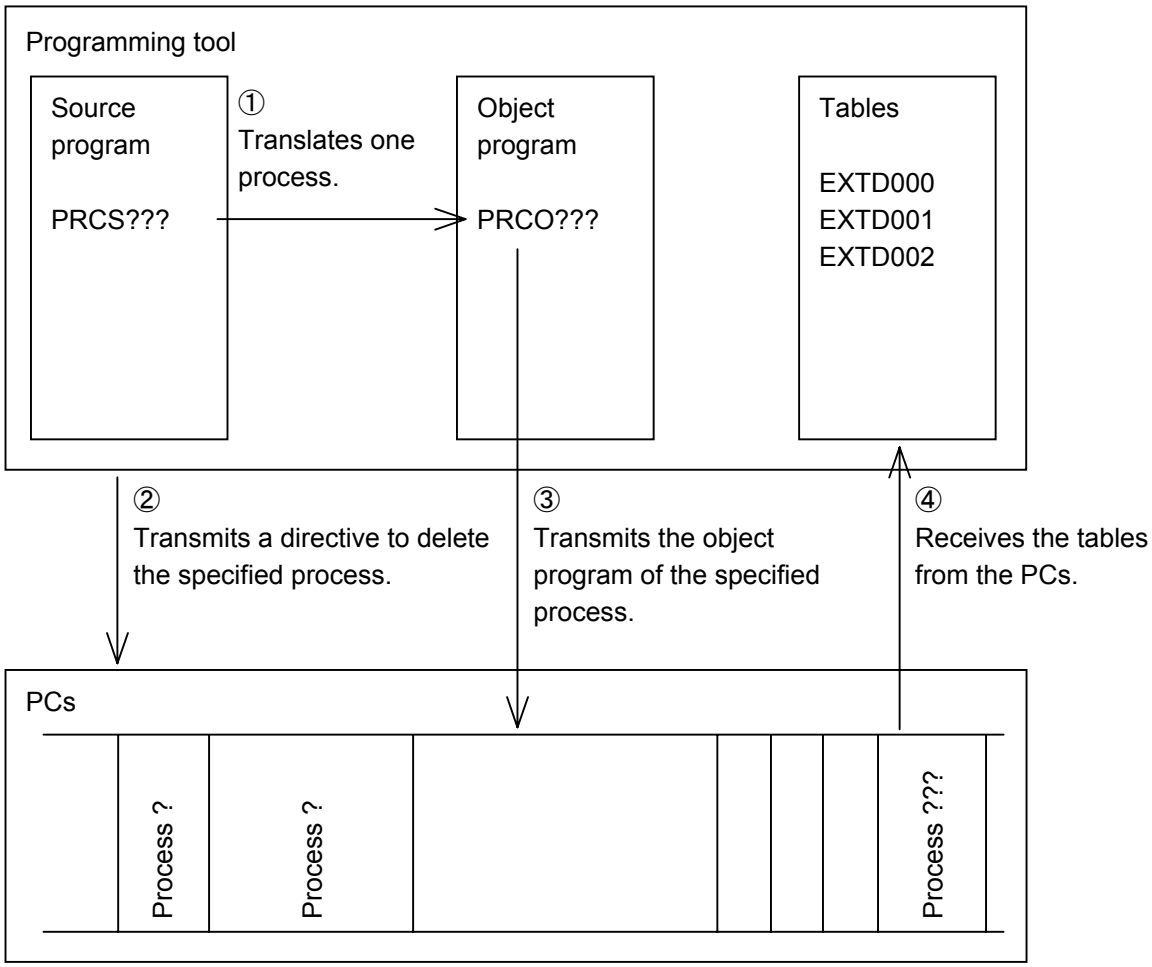
The following shows the data flow when receiving all HI-FLOW programs from the PCs with the tool.



On reception, the processes and tables in the memory are not always arranged in ascending order.

(3) One-process transmission

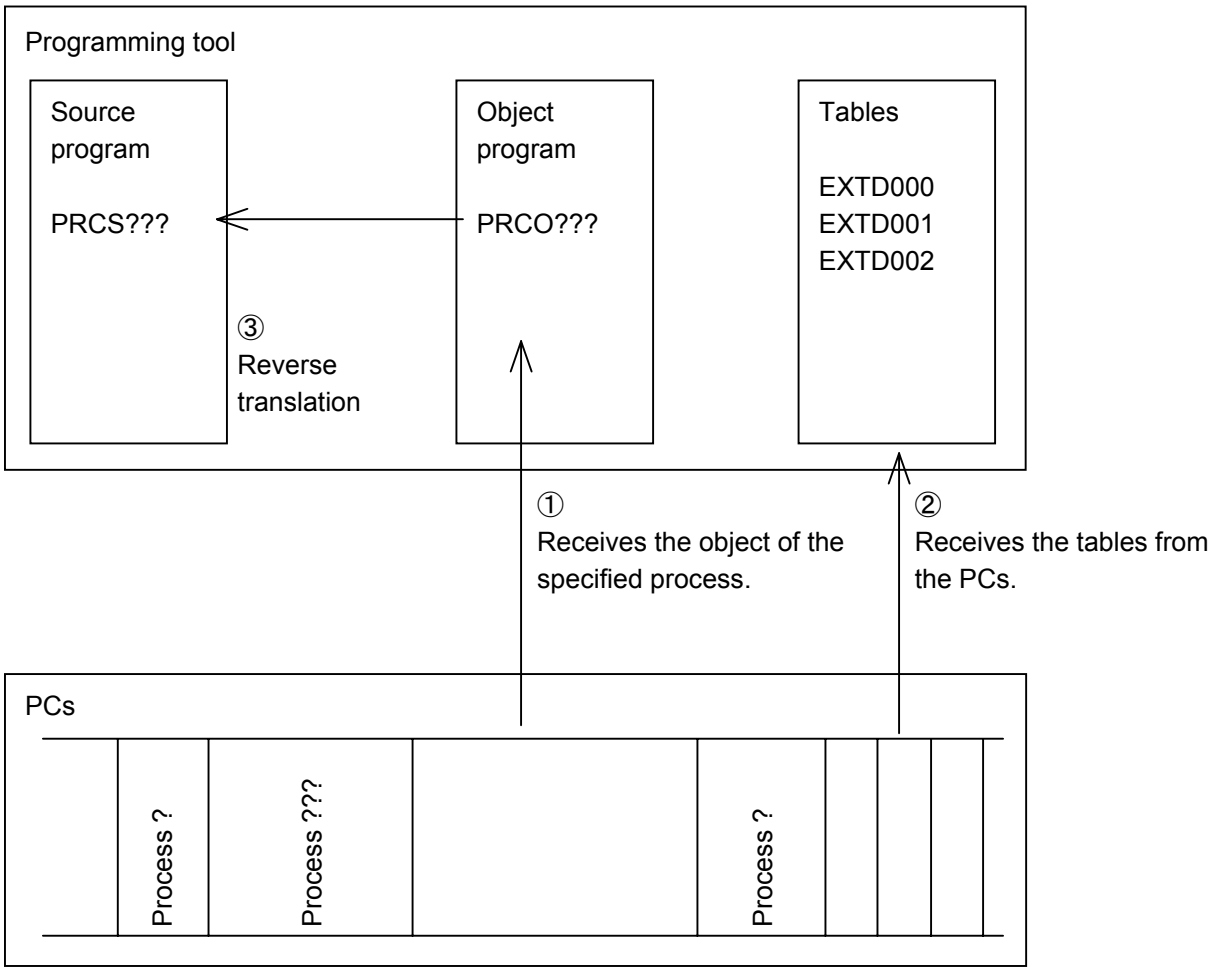
The following shows the data flow when transmitting one specific HI-FLOW process on the tool to the PCs.



After the process has been transmitted, it is placed after all other processes in the memory.

(4) One-process reception

The following shows the data flow when receiving one specific HI-FLOW process on the PCs with the tool.



On reception, the processes and tables on the memory are not always arranged in ascending order.

Supplement 4 Check for Progress

HI-FLOW indicates the current position on the user program with the monitor cursor. The HI-FLOW system in the PCs controls the progress of the current position. This section describes how the user program transferred to the PCs is checked for progress by the PCs.

Item	Description
Basic rules	・ The progress is checked at each by the PCs. ・ Processes activated with ACT are checked sequentially in ascending order of process numbers. ・ In the same process, routes are checked sequentially in ascending order of route numbers. (The route numbers on the screen are increased from left to right and from top to bottom.) ・ In the same route, steps are checked sequentially in ascending order of step numbers. ・ Upon completion of checking a step, the next step is checked for progress. If the program cannot proceed to the next step, the route having the next route number is checked for process. At the next scan, progress check for this process and route starts with this step.
Called process	・ The called process is checked for progress after progress check for the calling process and calling route is done. When progress check for the called process is completed but process execution is not completed, the next route after the calling process and calling route is checked. When process execution is completed, the next step after the calling route is checked.
Process control	・ The process called with ACT is checked for progress at the time of activation. When processes have smaller process numbers than the process number of the process called with ACT, these processes are checked for progress at the next scan. When the process numbers are larger, these processes are checked at the same scan as for the process. ・ RST, STP, and CLR in Control Box and Process Start are processed at they are specified.
Around-the-clock monitoring	・ The condition to start a process (ACT, STP, RST, or CLR) and multi-entry condition are checked before the process is checked for progress. When the condition is satisfied, the processing is performed before process progress check. ・ Y-output interlock conditions are checked at each scan and Y-output is turned on or off before the first process is checked for progress.

Item	Description
Branch	・ After a branch step (If or Jump) is executed, the destination step is checked for progress. Therefore, the route being executed may not be checked for progress for one scan or it may be checked twice. Also, a closed loop without progress conditions may be executed infinitely.
Repetition	・ After Repeat End, Repeat Start is checked for progress. Therefore, repetition without progress conditions may result in an infinite loop.
Forcible termination	・ When Escape is executed, the next process is checked for progress. When the process is a called process, the next step after the calling process and calling route is checked for progress.
Synchronization	・ After execution of Parallel Start, the next step is checked for progress. ・ Parallel End or Route End checks the next step after Parallel End for the joined route when all synchronous routes are terminated. When some synchronous routes are not terminated, Parallel End or Route End stops the local route and checks the next route for progress.
Selection	・ After execution of Select, the next step is checked for progress. ・ When the condition for Wait in Selective Branching is satisfied, the other selected routes are stopped and the next step is checked for progress. When the condition is not satisfied, the next route is checked. ・ Select End or Route End checks the next step after the Select End for the joined route. If the joined route is stopped, Select End or Route End activates it.
Wait for condition	・ When the condition is satisfied, Wait checks the next step for progress. When the condition is not satisfied, Wait checks the route having the next number. At the next scan, progress check for this process and route starts with this step. ・ Wait in Selective Branching checks whether the previous step is an ON statement before proceeding to the next step. If so, Wait in Selective Branching clears to 0 the ON statement.
Symbol with no delay	・ This symbol enables the program to proceed to the next step with no delay in any case. The following steps proceed to the appropriate step with no delay: Process Start, Route Start, Parallel Start, Select, Multi-entry, Box , Control Box, Function, Process End at termination of a called process, Route End and Parallel End at synchronization end, Route End and Select End at selective joining, and branch steps (Repeat Step, Repeat End, If, and Jump).

Supplement 5 Relationships between a HI-FLOW Program and the CPU Load

A HI-FLOW program runs on a PCs as part of the operating system. As the amount of the HI-FLOW program increases, therefore, the load of the operating system in the PCs increases. As a result, the following problems arise:

- More sequence cycle time than the specified value is required.
- The states of LEDs on the PCs cannot change.
- When the PCs is reset, LEDs do not light up.

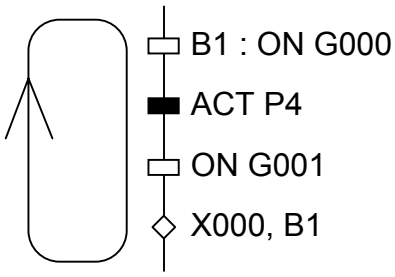
If the load is further increased, the entire system dose not rum correctly, for example, the sequence cycle stops. How to create an efficient HI-FLOW program is explained as well as a measure to judge the load.

How to create an efficient HI-FLOW program

1. The load of the HI-FLOW program is determined by the number of steps being executed. It is not affected by the vertical (route) length of the HI-FLOW program. Therefore, the load of a program consisting of too many processes and routes is high.

2. Do not make unnecessary loops.

   Do not make loops that are not required or have no stop points.



The program on the left continues executing steps in the loop until X000 is turned off, resulting in a high load.

3. Use timers so that their numbers are specified consecutively in ascending order. Wait timers, parallel timers, and counters having lower numbers have lower load.

4. In the same route, use wait timers having the same number.

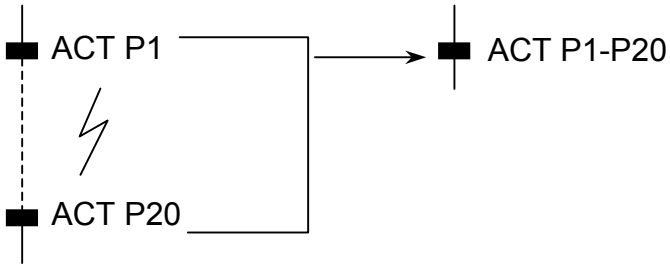
   In the same route, multiple timers are never used at the same time. Use wait timers having the same number. Do not use wait timers having a larger number as far as possible.

5. Use a minimum number of called processes.

   A program divided into subroutines is more understandable. During execution, however, the load is higher compared when called processes are not used. Carefully determine a program structure.

6. Do not use Control Box steps consecutively, if possible.

   The execution load of the Control Box step is very high. Do not use Control Box steps consecutively as far as possible. If it is unavoidable to use them in such a way, use consecutive specification of processes in an efficient way.



7. Set a minimum number of system control bits.

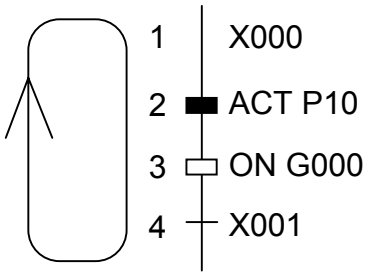
   System control bits (see utilities in the operation manual) must be checked in each sequence cycle or during execution of each step, increasing the load. Set a minimum number of system control bits.

8. Use a minimum number of Multi-entry steps.

   Multi-entry steps must be checked in each sequence cycle. When too many Multi-entry steps are used, the load becomes high. Use a minimum number of Multi-entry steps.

9. Take care so that the Multi-entry step does not enter an in-loop.

   The Multi-entry step checks condition expressions in each cycle. If a condition is satisfied, Multi-entry starts execution from the step. If consecutive conditions rather than an edge condition are satisfied, Multi-entry enters an infinite loop. Be sure to edge-trigger for conditions Multi-entry.



If X001 is not satisfied and X000 is left on in the program on the left, steps 1 to 4 are executed in each sequence cycle. To prevent this, set an edge condition for X000.

10. Set STP or RST in the Process Start step a minimum number of times.

    The Process Start step with STP or RST specified checks conditions in each sequence cycle, increasing the load significantly. Set STP or RST a minimum number of times.

11. Take care when setting CLR in the Process Start step.

    The Process Start step with CLR specified clears PI/O each time a condition is satisfied, increasing the load. (The Process Start step with RST, STP, or ACT specified does not check further conditions once a condition is satisfied.) Create conditions to be checked by the Process Start step with CLR specified.

12. Do not use consecutive application instructions if possible.

    Application instructions perform operation continuously. If they are written consecutively, the sequence cycle may be extended. When writing application instructions, take care.

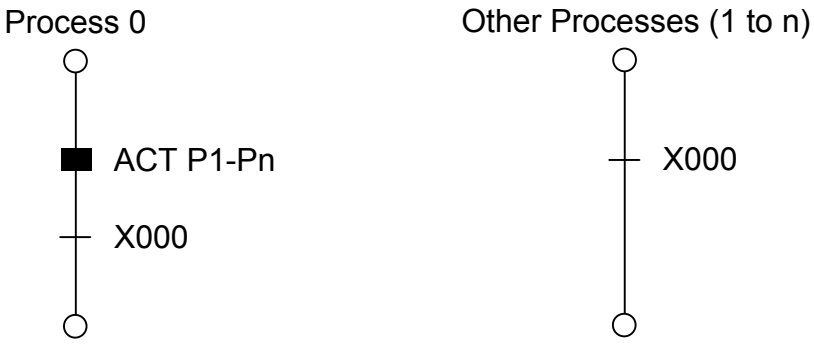
13. Do not use complex condition expressions if possible.

    When complex condition expressions are used in a HI-FLOW program, analysis of them takes more time than a ladder program. When using complex condition expressions, write them in a ladder program then pass them to the HI-FLOW program.

Correlation between a HI-FLOW program and the CPU load

The following tables gives measures of times of sequence cycle during which processes can be executed on the S10/2α, S10/2αE, and S10/2αH(f) for individual process counts. The conditions below are assumed:

(1) The following processes are used. Conditions are not satisfied. Other programs including a ladder program are not used.



(2) All system control bits are invalid.

(3) When the PCs are turned on with the CPU key switch set to RUN, the actual sequence cycle time is measured. (The sequence cycle accumulation counter in SW140 is measured for a certain time.)

[S10/2α] : These process can be executed in the specified time.
: When these processes are set, the sequence cycle stops.

Specified sequence cycle time (ms)	Number of processes to be executed (n)						
	10	30	50	100	150	200	256
10							
20							
30							
50							
75							
100							
125							
150							
175							
200							

[S10/2αE] ▒▒ : These processes can be executed in the specified time.

Specified sequence cycle time (ms)	Number of processes to be executed (n)						
	10	30	50	100	150	200	256
10	▒▒	▒▒					
20	▒▒	▒▒	▒▒	▒▒			
30	▒▒	▒▒	▒▒	▒▒	▒▒		
50	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
75	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
100	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
125	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
150	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
175	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
200	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒

[S10/2αH(f)] ▒▒ : These processes can be executed in the specified time.

Specified sequence cycle time (ms)	Number of processes to be executed (n)						
	10	30	50	100	150	200	256
10	▒▒	▒▒	▒▒				
20	▒▒	▒▒	▒▒	▒▒	▒▒		
30	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	
50	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
75	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
100	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
125	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
150	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
175	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
200	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒

When actually designing the system, set the sequence cycle time based on the above table, taking the safety factor into account. Considering the ladder operation functions and the existence of C mode programs, about halt of the sequence cycle time in which program execution is possible on the PCs is appropriate.

INDEX