

Mitsubishi Programmable Controller

MELSEC iQ-R
series

MELSEC iQ-R Programming Manual
(Program Design)

SAFETY PRECAUTIONS

(Read these precautions before using this product.)

Before using MELSEC iQ-R series programmable controllers, please read the manuals for the product and the relevant manuals introduced in those manuals carefully, and pay full attention to safety to handle the product correctly.

Make sure that the end users read this manual and then keep the manual in a safe place for future reference.

CONDITIONS OF USE FOR THE PRODUCT

(1) Mitsubishi programmable controller ("the PRODUCT") shall be used in conditions;

- i) where any problem, fault or failure occurring in the PRODUCT, if any, shall not lead to any major or serious accident; and
- ii) where the backup and fail-safe function are systematically or automatically provided outside of the PRODUCT for the case of any problem, fault or failure occurring in the PRODUCT.

(2) The PRODUCT has been designed and manufactured for the purpose of being used in general industries.

MITSUBISHI SHALL HAVE NO RESPONSIBILITY OR LIABILITY (INCLUDING, BUT NOT LIMITED TO ANY AND ALL RESPONSIBILITY OR LIABILITY BASED ON CONTRACT, WARRANTY, TORT, PRODUCT LIABILITY) FOR ANY INJURY OR DEATH TO PERSONS OR LOSS OR DAMAGE TO PROPERTY CAUSED BY the PRODUCT THAT ARE OPERATED OR USED IN APPLICATION NOT INTENDED OR EXCLUDED BY INSTRUCTIONS, PRECAUTIONS, OR WARNING CONTAINED IN MITSUBISHI'S USER, INSTRUCTION AND/OR SAFETY MANUALS, TECHNICAL BULLETINS AND GUIDELINES FOR the PRODUCT.

("Prohibited Application")

Prohibited Applications include, but not limited to, the use of the PRODUCT in;

- Nuclear Power Plants and any other power plants operated by Power companies, and/or any other cases in which the public could be affected if any problem or fault occurs in the PRODUCT.
- Railway companies or Public service purposes, and/or any other cases in which establishment of a special quality assurance system is required by the Purchaser or End User.
- Aircraft or Aerospace, Medical applications, Train equipment, transport equipment such as Elevator and Escalator, Incineration and Fuel devices, Vehicles, Manned transportation, Equipment for Recreation and Amusement, and Safety devices, handling of Nuclear or Hazardous Materials or Chemicals, Mining and Drilling, and/or other applications where there is a significant risk of injury to the public or property.

Notwithstanding the above, restrictions Mitsubishi may in its sole discretion, authorize use of the PRODUCT in one or more of the Prohibited Applications, provided that the usage of the PRODUCT is limited only for the specific applications agreed to by Mitsubishi and provided further that no special quality assurance or fail-safe, redundant or other safety features which exceed the general specifications of the PRODUCTS are required. For details, please contact the Mitsubishi representative in your region.

- For Safety CPUs

- (1) Although MELCO has obtained the certification for Product's compliance to the international safety standards IEC61508, EN954-1/ISO13849-1 from TUV Rheinland, this fact does not guarantee that Product will be free from any malfunction or failure. The user of this Product shall comply with any and all applicable safety standard, regulation or law and take appropriate safety measures for the system in which the Product is installed or used and shall take the second or third safety measures other than the Product. MELCO is not liable for damages that could have been prevented by compliance with any applicable safety standard, regulation or law.
- (2) MELCO prohibits the use of Products with or in any application involving, and MELCO shall not be liable for a default, a liability for defect warranty, a quality assurance, negligence or other tort and a product liability in these applications.
 - (a) power plants,
 - (b) trains, railway systems, airplanes, airline operations, other transportation systems,
 - (c) hospitals, medical care, dialysis and life support facilities or equipment,
 - (d) amusement equipments,
 - (e) incineration and fuel devices,
 - (f) handling of nuclear or hazardous materials or chemicals,
 - (g) mining and drilling,
 - (h) and other applications where the level of risk to human life, health or property are elevated.

INTRODUCTION

Thank you for purchasing the Mitsubishi MELSEC iQ-R series programmable controllers.

This manual describes the program structures and data required for programming.

Before using this product, please read this manual and the relevant manuals carefully and develop familiarity with the functions and performance of the MELSEC iQ-R series programmable controller to handle the product correctly.

When applying the program examples provided in this manual to an actual system, ensure the applicability and confirm that it will not cause system control problems.

Please make sure that the end users read this manual.



Most of the information in this manual is described using labels. Devices can be used in the same way as labels.

MEMO

CONTENTS

SAFETY PRECAUTIONS	1
CONDITIONS OF USE FOR THE PRODUCT	1
INTRODUCTION	3
RELEVANT MANUALS	7
TERMS	8
CHAPTER 1 OVERVIEW	9
CHAPTER 2 PROGRAM CONFIGURATION	11
CHAPTER 3 PROGRAM ORGANIZATION UNITS	13
3.1 Program Blocks	14
3.2 Functions (FUN)	15
3.3 Function Blocks (FB)	20
3.4 Precautions	30
3.5 When a Safety Program Is Used	35
Safety functions (Safety FUN)	35
Safety function blocks (Safety FB)	36
CHAPTER 4 LABELS	37
4.1 Label Types	37
4.2 Classes	38
4.3 Data Types	39
4.4 Arrays	42
4.5 Structures	45
4.6 Constants	47
4.7 Precautions	48
4.8 When a Safety Program Is Used	50
Safety label types	50
Classes	51
Data types	52
Structures	52
CHAPTER 5 LADDER DIAGRAM	53
5.1 Configuration	53
Ladder symbols	53
Program execution order	54
5.2 Inline ST	55
5.3 Statements and Notes	57
CHAPTER 6 STRUCTURED TEXT LANGUAGE	58
6.1 Configuration	59
Delimiters	60
Operators	60
Control statements	61
Constants	70
Labels and devices	71
Comments	73

CHAPTER 7	FBD/LD	74
7.1	Configuration	74
	Program elements	75
	Constant	82
	Labels and devices	82
7.2	Program Execution Order	84
	Execution order of program elements	84
CHAPTER 8	SFC PROGRAM	86
8.1	Specifications	89
8.2	Structure	90
	Block	91
	Step	92
	Action	103
	Transition	107
8.3	SFC Control Instructions	118
8.4	SFC Information Devices	120
8.5	SFC Setting	128
	CPU parameter	128
	SFC block setting	134
8.6	SFC Program Execution Order	135
	Whole program processing	135
	SFC program processing sequence	137
8.7	SFC Program Execution	140
	Starting and stopping the SFC program	140
	Starting and ending a block	141
	Pausing and restarting a block	142
	Activating and deactivating a step	143
	Behavior when an active step is activated	144
	Operation when a program is modified	145
	Checking SFC program operation	146
INDEX		148
	REVISIONS	150
	WARRANTY	151
	TRADEMARKS	152

RELEVANT MANUALS

Manual name [manual number]	Description	Available form
MELSEC iQ-R Programming Manual (Program Design) [SH-081265ENG] (this manual)	Program specifications (ladder, ST, FBD/LD, and SFC programs) and labels	e-Manual PDF
MELSEC iQ-R Programming Manual (Instructions, Standard Functions/ Function Blocks) [SH-081266ENG]	Instructions for the CPU module, dedicated instructions for the intelligent function modules, and standard functions/function blocks	e-Manual PDF
GX Works3 Operating Manual [SH-081215ENG]	System configuration, parameter settings, and online operations of GX Works3	e-Manual PDF



e-Manual refers to the Mitsubishi FA electronic book manuals that can be browsed using a dedicated tool.

e-Manual has the following features:

- Required information can be cross-searched in multiple manuals.
- Other manuals can be accessed from the links in the manual.
- The hardware specifications of each part can be found from the product figures.
- Pages that users often browse can be bookmarked.

TERMS

Unless otherwise specified, this manual uses the following terms.

Term	Description
Buffer memory	Memory in an intelligent function module for storing data such as setting values and monitored values. Buffer memory in a CPU module stores setting values and monitored values of the Ethernet function and data used for data communications among the CPU modules in a multiple CPU system.
CPU module	A generic term for the MELSEC iQ-R series CPU modules
Device	A device (X, Y, M, D, or others) in a CPU module
Engineering tool	The product name of the software package for the MELSEC programmable controllers
GX Works3	The product name of the software package, SWnDNC-GXW3, for the MELSEC programmable controllers (The 'n' represents a version.)
I/O module	A generic term for the input module, output module, I/O combined module, and interrupt module
Intelligent function module	A module that has functions other than input and output, such as an A/D converter module and D/A converter module
Label	A label that represents a device in a given character string
Module label	A label that represents one of memory areas (I/O signals and buffer memory areas) specific to each module in a given character string. For the module used, the engineering tool automatically generates this label, which can be used as a global label.
Multiple CPU system	A system where two to four CPU modules separately control I/O modules and intelligent function modules
Network module	A generic term for the following modules: <ul style="list-style-type: none"> • Ethernet interface module • CC-Link IE Controller Network module • Module on CC-Link IE Field Network • MELSECNET/H network module • MELSECNET/10 network module • RnENCPU (network part)
Operand	A generic term for the devices, such as source data (s), destination data (d), number of devices (n), and others, used as parts to configure instructions and functions
POU	A unit that configures a program. Units are categorized and provided in accordance with functions. Use of POU's enables dividing the lower-layer processing in a hierarchical program into some units in accordance with processing or functions, and creating programs for each unit.
Predefined protocol support function	A function of GX Works3. This function sets protocols appropriate to each external device and reads/writes protocol setting data.
Standard/safety shared label	A label that can be used in both standard programs and safety programs. This label is used to pass data between safety programs and standard programs.

The following terms are used to explain a safety programmable controller system using the Safety CPU.

Term	Description
Safety control	Machine control by safety programs and safety data communications. When an error occurs, the machine in operation is securely stopped.
Safety communications	Communication service that performs send/receive processing in the safety layer of the safety communication protocol
Safety device	A device that can be used in safety programs
Safety program	A program that performs safety control
Standard CPU	A generic term for MELSEC iQ-R series CPU modules (other than Safety CPU) that perform standard control (This term is used to distinguish from the Safety CPU.)
Standard control	Machine control by standard programs and standard data communications. Programmable controllers other than the safety programmable controller perform only standard control. (This term is used to distinguish from safety control.)
Standard communications	Communications other than safety communications, such as cyclic transmission and transient transmission of CC-Link IE Field Network
Standard device	A device (X, Y, M, D, or others) in a CPU module. (Safety devices are excluded.) This device can be used only in standard programs. (This term is used to distinguish from a safety device.)
Standard program	A program that performs sequence control. (Safety programs are excluded.) (This term is used to distinguish from a safety program.)

1 OVERVIEW

This manual describes program configurations, contents, and coding methods required for programming.
For information on creating, editing, and monitoring programs using an engineering tool, refer to the following.

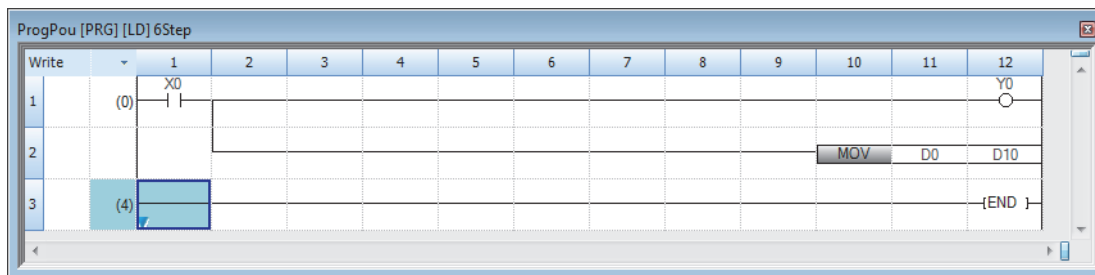
📖 GX Works3 Operating Manual

Programming languages

With the MELSEC iQ-R series, an optimal programming language can be selected and used according to the application.

Programming language	Description
Ladder diagram (Ladder)	A graphic language which describes ladders consisting of contacts and coils. This language is used to describe logical ladders using symbolized contacts and coils to enable easy-to-understand sequence control.
Structured text language (ST)	A textual language used to describe programs using statements (such as IF) and operators. Compared with the ladder diagram, this language can describe hard-to-describe operation processing concisely and legibly, and therefore is suitable for programming complicated arithmetic operations and comparison operations. Also, as with C, ST language can describe syntax control such as selective branches with conditional statements and repetitions with iteration statements, and thus can describe easy-to-understand, concise programs.
Function block diagram/ladder diagram (FBD/LD)	A graphic language which describes programs by connecting blocks that perform predefined processing, variable elements, and constant elements along the flow of data and signals. This language facilitates programming of DDC (direct digital control) processing which is difficult to describe in ladder diagram, and improves the productivity of programs.
Sequential function chart (SFC program)	SFC is a program description format in which a sequence of control operations is split into a series of steps to enable a clear expression of each program execution sequence and execution conditions.

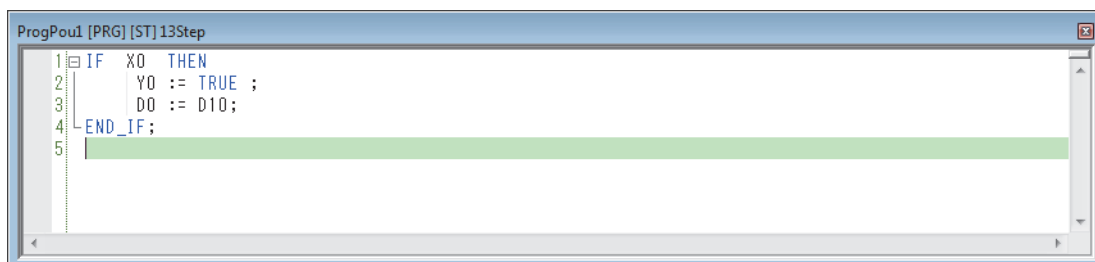
■ Ladder diagram (Ladder)



For details, refer to the following.

📖 Page 53 LADDER DIAGRAM

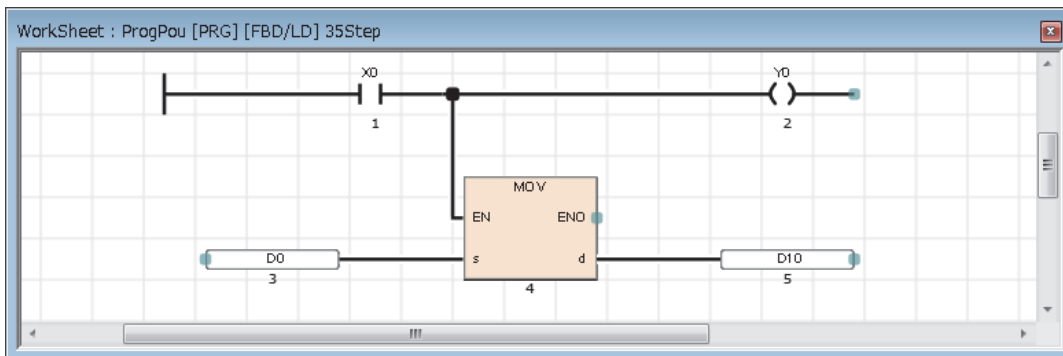
■ Structured text language (ST)



For details, refer to the following.

📖 Page 58 STRUCTURED TEXT LANGUAGE

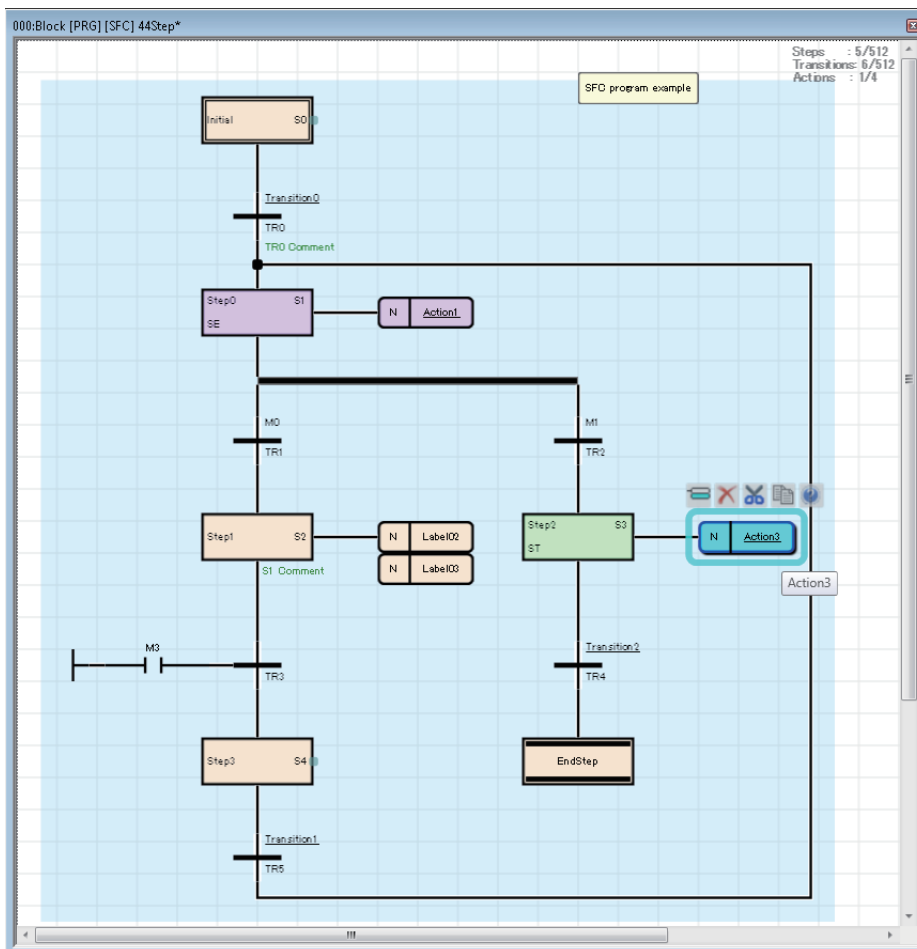
■Function block diagram/ladder diagram (FBD/LD)



For details, refer to the following.

📖 Page 74 FBD/LD

■SFC program



For details, refer to the following.

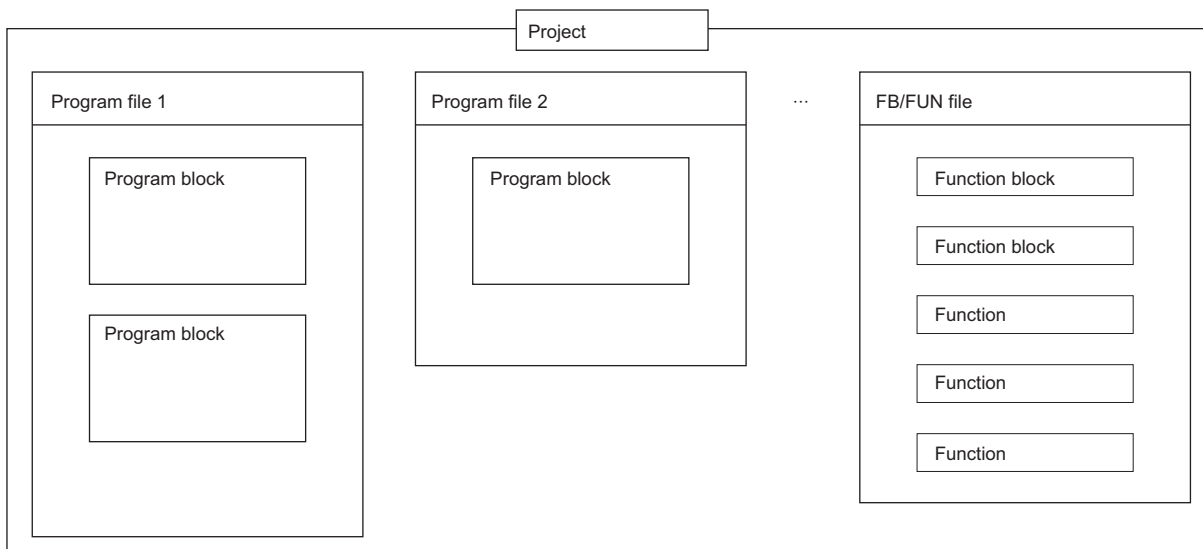
📖 Page 86 SFC PROGRAM

Point

- Programming in ladder is suitable for users who have knowledge and experience of sequence control and logical ladders. Programming in ST is suitable for users who have knowledge and experience of C programming. Programming in FBD/LD is suitable for users who perform process control. SFC program is suitable for creating program blocks for each actual control of machines and controlling the transition of each operation.
- Using labels in programs can improve readability of programs, and make it easy to immigrate programs to a system having a different module configuration.

2 PROGRAM CONFIGURATION

Using the engineering tool, multiple programs and program organization units (POUs) can be created. Programs and POUs can be divided according to processing. This chapter describes the program configuration.



For POUs, refer to the following.

☞ Page 13 PROGRAM ORGANIZATION UNITS

Project

A project is a group of data (such as programs and parameters) to be executed in a CPU module.

Only one project can be written to a single CPU module.

At least one program file needs to be created in a project.

Program file

A program file is a group of programs and POUs.

A program file consists of at least one program block. (☞ Page 14 Program Blocks)

The following operations are performed in units of program file: changing the program execution type from the fixed scan execution type to the standby type and writing data to the CPU module.

MEMO

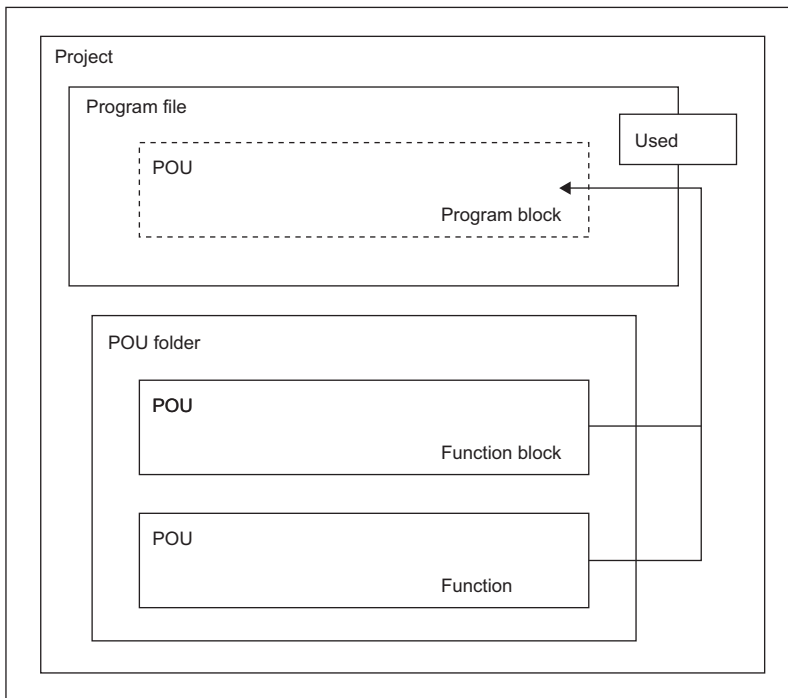
3 PROGRAM ORGANIZATION UNITS

There are three types of program organization units (POUs).

- Program block
- Function
- Function block

Processing can be described in the programming language that suits the control performed in each POU. Processing can be described in the ladder diagram, structured text language, or FBD/LD in a function or a function block.

Functions and function blocks are called and executed by program blocks.



3

Point

A structured program is a program created by components. Processes in lower levels of hierarchical sequence program are divided into several components according to their processing information and functions.

Each component of a program is specified to have a high degree of independence for easy addition and replacement.

The following are the examples of processing that would be ideal to be structured.

- Processing which is used repeatedly in a program
- Processing which can be separated as one function

This chapter describes two types of POUs using labels.

Devices can also be used in the program (worksheet) of each POU. For details on devices, refer to the following.

MELSEC iQ-R CPU Module User's Manual (Application)

Point

Up to 32 worksheets can be created in one POU in the structured text language and FBD/LD.

Set the execution order of multiple worksheets on the "Worksheet Execution Order Setting" window of the engineering tool. (GX Works3 Operating Manual)

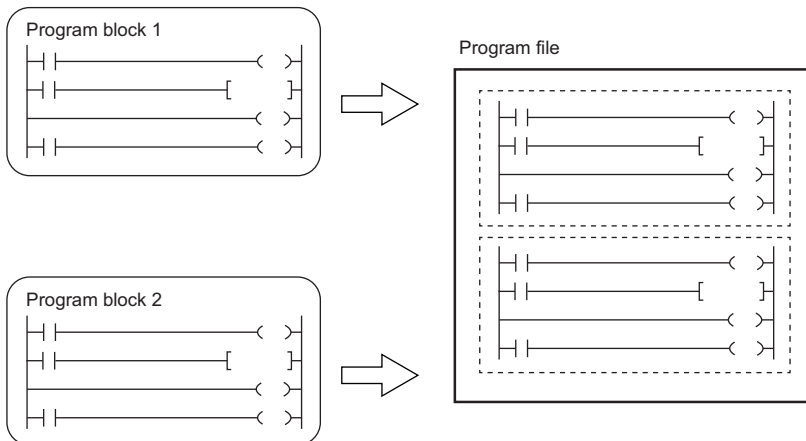
3.1 Program Blocks

A program block is a unit for making up a program.

Multiple program blocks can be created in a program file and executed in the order specified in the program file setting. If the order is not specified in the program file setting, the program blocks are executed in ascending order of their names.

By separating program blocks for individual functions and processing, the order of programs can be changed easily and programs can be exchanged easily.

The program of a program block is stored by each registration destination program in a program file.



Creating a main routine program, subroutine program, and interrupt program separately in individual program blocks enables creation of easy-to-understand programs. ^{*1}

Program type	Description
Main routine program	A program beginning with step 0 and ending with the FEND instruction
Subroutine program	A program beginning with a pointer (P) and ending with the RET instruction. This program is executed only when it is called by a subroutine call instruction (such as the CALL and ECALL instructions).
Interrupt program	A program beginning with an interrupt pointer (I) and ending with the IRET instruction. When an interrupt factor occurs, the interrupt program corresponding to the interrupt pointer number is executed.

^{*1} Subroutine programs and interrupt programs cannot be created in safety programs. Subroutine programs cannot be executed from safety programs.

For details on the main routine program, subroutine program, and interrupt program, refer to the following.

MELSEC iQ-R CPU Module User's Manual (Application)

- Create a subroutine program and interrupt program after the FEND instruction of the main routine program. Any program after the FEND instruction is not executed as a main routine program. For example, when the FEND instruction is used at the end of the second program block, the third program block or later runs as a subroutine program or interrupt program.
- To create an easy-to-understand program, use a pair of instructions, such as the FOR and NEXT instructions or the MC and MCR instructions, within a single program block.
- A simple program can be executed in the CPU module simply by writing the main routine in one program block.

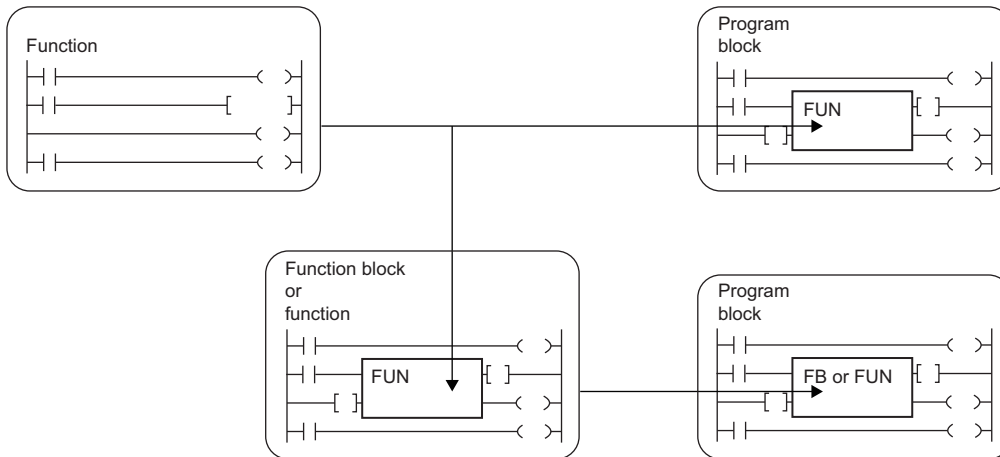
3.2 Functions (FUN)

A function is a POU called and executed by program blocks, function blocks, and other functions.

After the processing completes, a function passes a value to the calling source. This value is called a return value.

A function always outputs the same return value, as the processing result, for the same input.

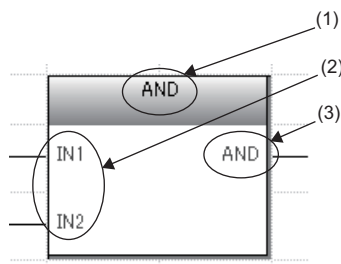
By defining simple, independent algorithms that are frequently used, functions can be reused efficiently.



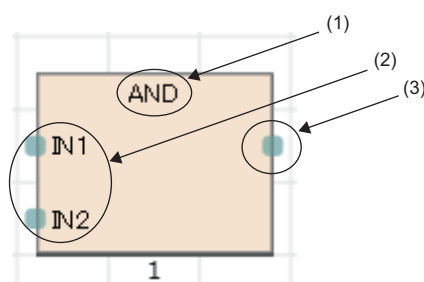
Input variables and output variables

Input and output variables can be defined in functions. Output data which is different from the return value can be assigned to the output variable.

Ladder program



FBD/LD program



- (1) Function name
- (2) Input variable
- (3) Output variable

The return value of the function is not displayed.

For the classes in which input and output variables can be set, refer to the following.

📖 Page 38 Classes

Point

Variables defined in the function are overwritten every time the function is called.


To hold the data in the variables, create a program by using function blocks or so that the data in the output variable is saved in another variable.

EN and ENO

EN (enable input) and ENO (enable output) can be appended to a function to control execution processing.

- Set a boolean variable used as an execution condition of a function to EN.
- A function with EN is executed only when the execution condition of EN is TRUE.
- Set a boolean variable used to output a function execution result to ENO.

For the boolean type, refer to the following.

 Page 39 Data Types

The following table lists the ENO states and operation results according to the EN states.

EN	ENO	Operation result
TRUE (executed)	TRUE	Operation result output value
FALSE (not executed)	FALSE	Undefined value

Point

- Setting an output label to ENO is not always required for the program written in ladder or FBD/LD.
- When EN/ENO is used in a standard function, the function with EN is represented by "function-name_E".

Creating programs

The program of a function can be created by using the engineering tool.



 [Navigation window] ⇒ [FB/FUN] ⇒ Right-click ⇒ [Add New Data]

The created program is stored in the FB/FUN file.

 [CPU Parameter] ⇒ [Program Setting] ⇒ [FB/FUN File Setting]

Up to 64 programs can be stored in one FB/FUN file.

For details on program creation, refer to the following.

Item	Reference
How to create function programs	 GX Works3 Operating Manual
Number of FB/FUN files that can be written to a CPU module	 MELSEC iQ-R CPU Module User's Manual (Startup)

■Applicable devices and labels

The following table lists the devices and labels that can be used in function programs.

○: Applicable, △: Applicable only in instructions (Cannot be used to indicate the program step.), ×: Not applicable

Type of device/label	Availability
Label (other than the pointer type)	Global label
	Local label
Label (pointer type)	Pointer type global label
	Pointer type local label
Device	Global device
	Local device
Pointer	Global pointer
	Local pointer

*1 The following data types cannot be used.

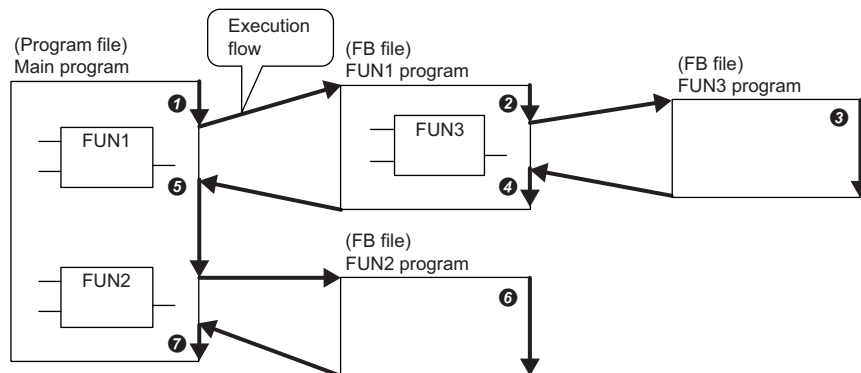
Timer, retentive timer, counter, long timer, long retentive timer, and long counter

Point

Program a function name as a label in a function to set a return value of the function. Setting function names as labels is not necessary. The data type set in "Result Type" in the properties of the function can be used.

Operation overview

The program of a function is stored in the FB/FUN file and called by the calling source program when executed.

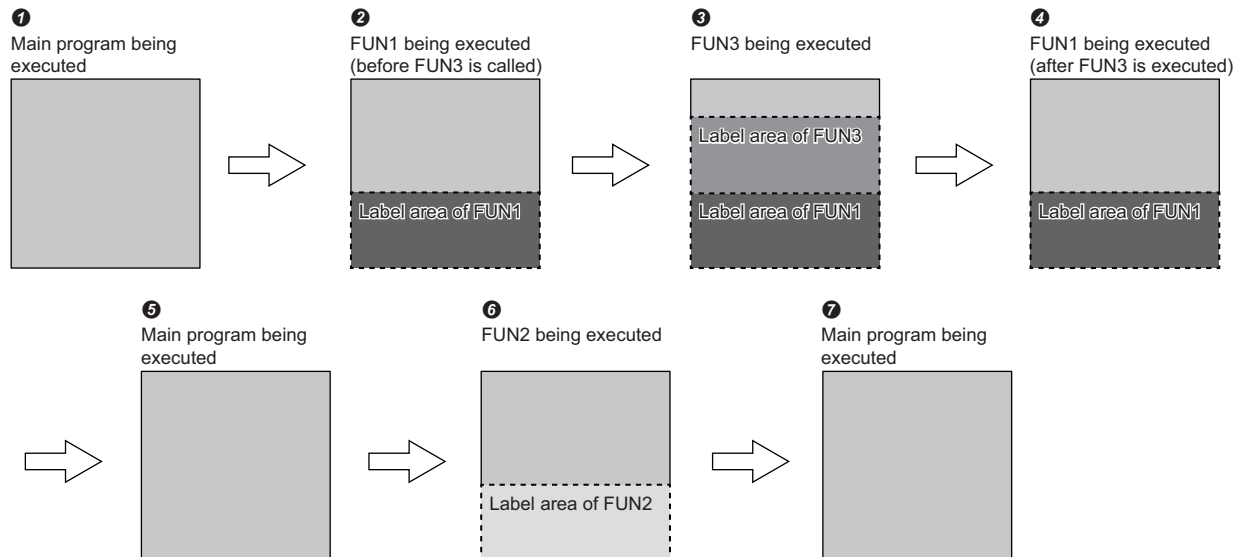


Up to 32 subroutine type function blocks, macro type function blocks, and functions in total can be nested.

Labels defined by a function

The labels defined by a function are assigned in the temporary areas of the storage-target memory during execution of the function, and the areas are freed after the processing completes.

The following figure shows the label assignments while the above functions are being executed.



For the types of labels can be defined by a function, refer to the following.

☞ Page 38 Classes

Point

The label to be defined by a function must be initialized by a program before the first access because the label value will be undefined.

Number of steps

To call a function, the number of steps is required not only for the program itself but also for the processing that passes the argument and return value and the processing that calls the program.

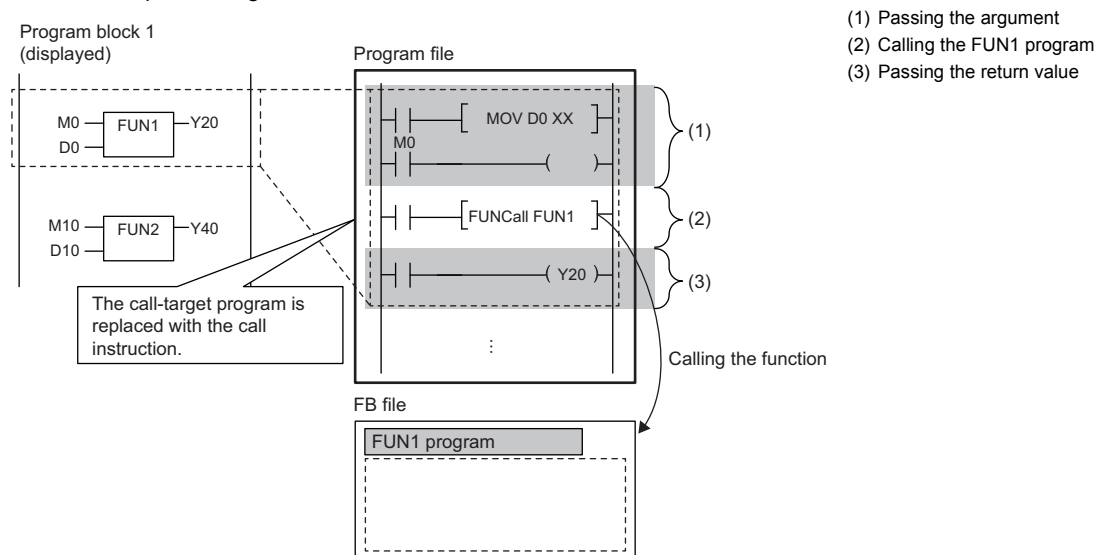
■Program

The number of steps required for a function program is the total number of instruction steps plus 22 steps. For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

■Calling source

When calling a function, the calling source generates the processing that passes the argument and return value before and after the call processing.



• Passing the argument

The instruction used to pass the argument differs depending on the class and data type of the argument. The following table summarizes the instructions that can be used to pass the argument.

Argument class	Data type	Instruction used	Number of steps
VAR_INPUT	Bit	LD+OUT LD+MOVB (Either of the instruction sets is used depending on the combination of programming language, function, and input argument used.)	For the number of steps required for each instruction, refer to the following. 📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)
	Word [unsigned]/bit string [16 bits] Double word [unsigned]/bit string [32 bits] Word [signed] Double word [signed]	LD+MOV LD+DMOV	
	Single-precision real number	LD+EMOV	
	Double-precision real number	LD+EDMOV	
	Time	LD+DMOV	
	String	LD+\$MOV	
	String [Unicode]	LD+\$MOV_WS	
	Array, Structure	LD+BMOV	

• Calling the program

A total of 26 steps are required to call the program of a function.

- Passing the return value

The instruction and the number of steps used for passing the return value are identical to those for passing the argument.

Argument class	Data type	Instruction used	Number of steps
VAR_OUTPUT	Same as for passing the argument	Same as for passing the argument	Same as for passing the argument

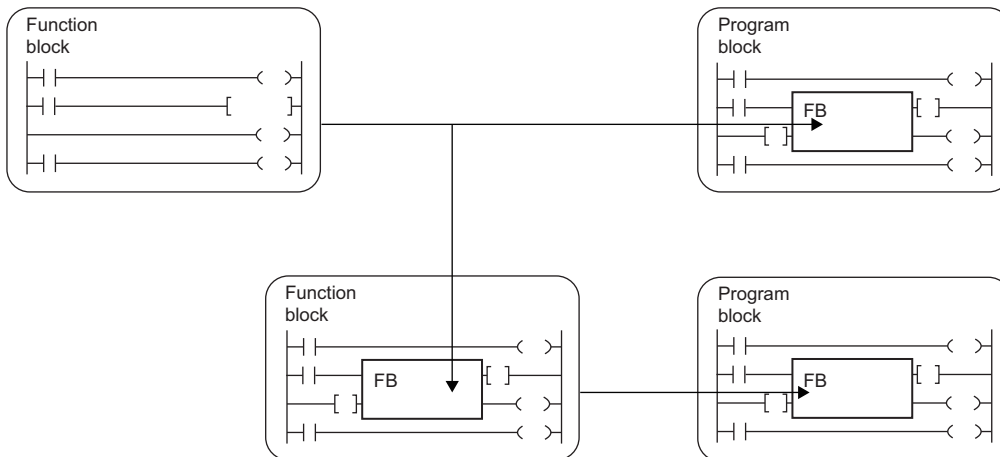
- EN/ENO

The following table lists the number of steps required for EN/ENO.

Item	Number of steps
EN	3
ENO	2

3.3 Function Blocks (FB)

A function block is a POU called and executed by program blocks and other function blocks.

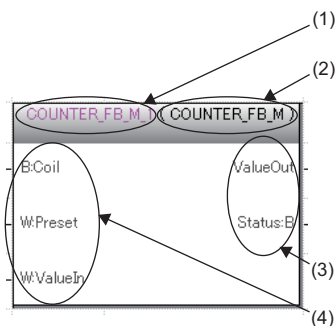


Unlike a function, a function block does not have a return value.

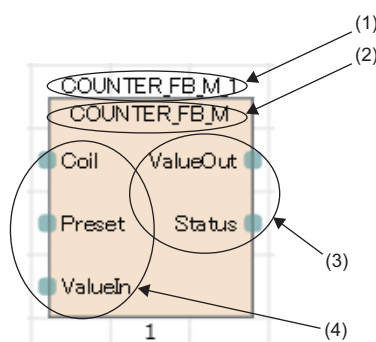
A function block can hold values in variables and thus can hold input states and processing results.

A function block uses the value it holds for the next processing and therefore it does not always output the same result even with the same input value.

Ladder program



FBD/LD program



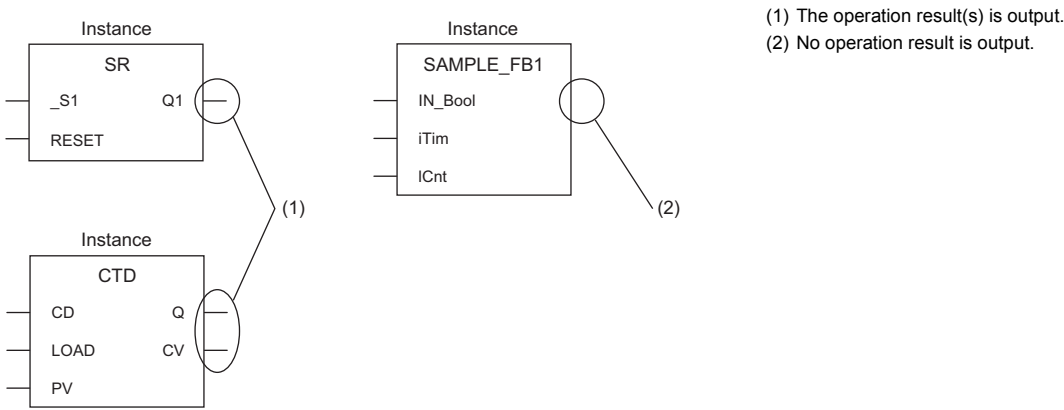
- (1) Instance name
- (2) Function block name
- (3) Output variable
- (4) Input variable

A function block needs to be instantiated to be used in programs.

📖 Page 22 Instances

Input variables, output variables, and input/output variables

Input variables, output variables, and input/output variables need to be defined in function blocks. A function block can output multiple operation results. It can also be set not to output operation results.

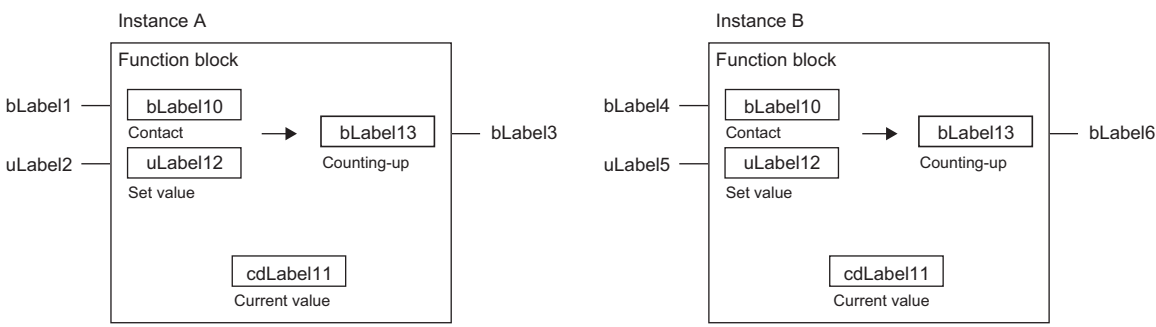


For the classes in which input variables, output variables, and input/output variables can be set, refer to the following.
[Page 38 Classes](#)

Internal variables

Function blocks use internal variables. For each instance of a function block, labels are assigned to the different areas. Even though the same label names are used, different states are held for each instance.

Ex.



The above function block starts counting when the input variables turn on and turns on the output variable when the current value held in the internal variable reaches the set value. Even though the same function block is used, the output timings differ because the instances A and B hold different states.

For the classes in which internal variables can be set, refer to the following.
[Page 38 Classes](#)

External variables

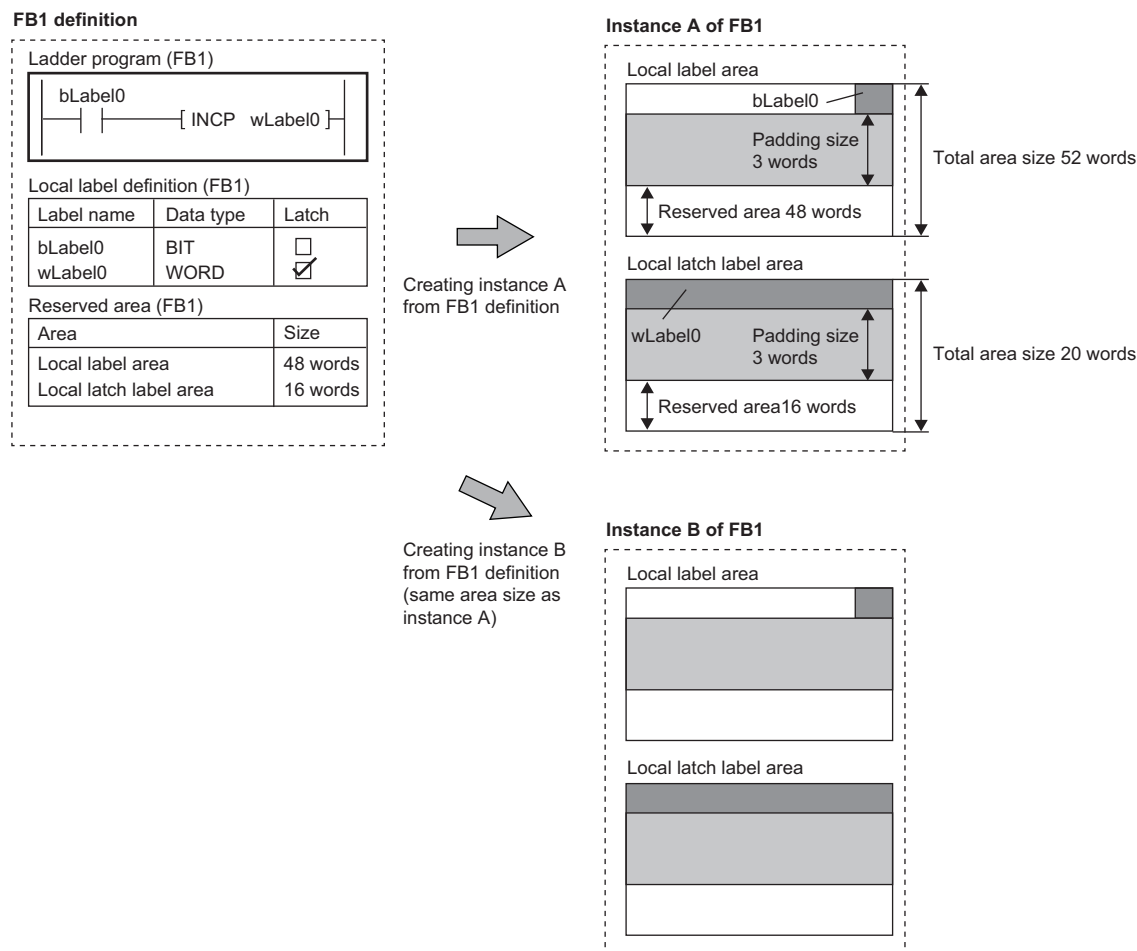
Function blocks can use external variables. For the classes in which external variables can be set, refer to the following.
[Page 38 Classes](#)

Instances

■Instances

An instance is a label assigned to realize a function block definition. Multiple instances can be created from one function block definition.

The following figure shows the instance structure.



(Since the label area is secured in units of four words, three-word areas (padding size) are secured in the above example.)

■Creating instances

A function block needs to be instantiated to be used in programs.

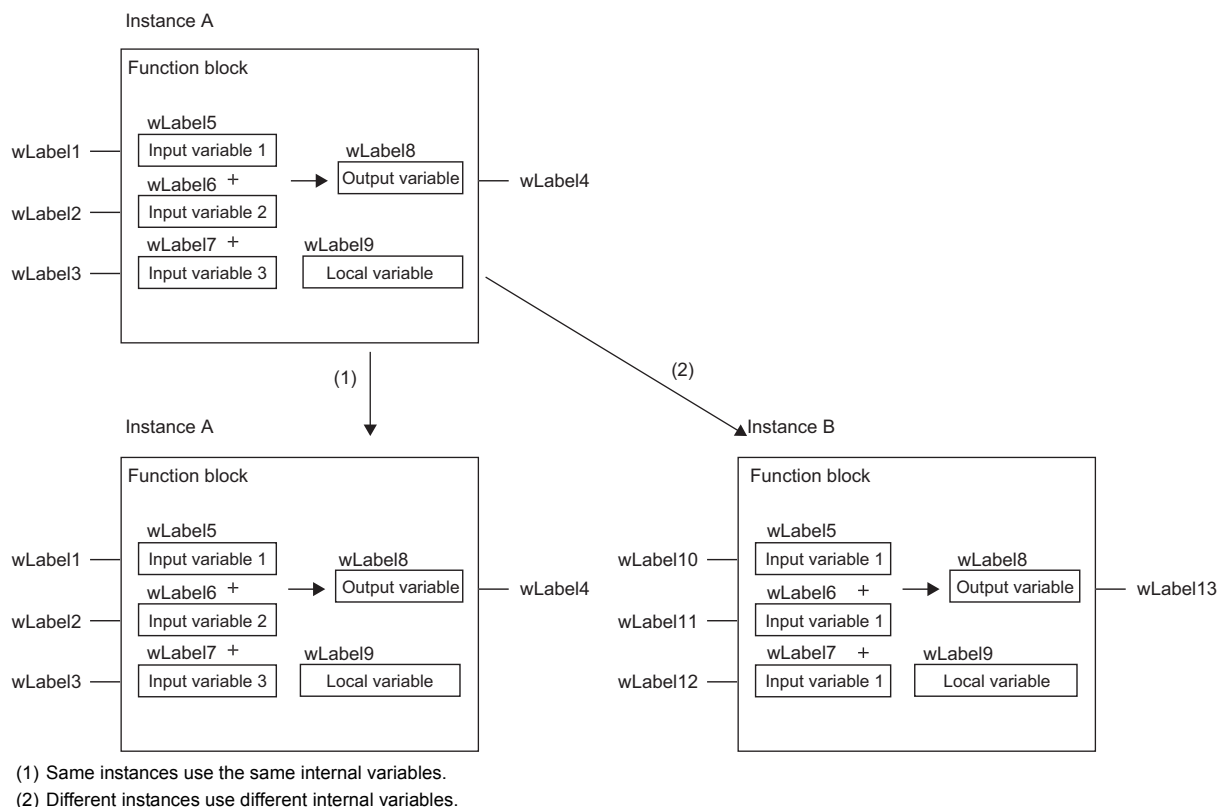
By creating instances, a function block can be called and executed by a program block or another function block.

Declare instances with global labels or local labels.

Label type	Instance type	Class
Global label	Global FB	VAR_GLOBAL
Local label ^{*1}	Local FB	VAR

^{*1} Local labels can be declared as the local labels of a program block or function block. Local labels cannot be declared in a function.

Same function blocks can be instantiated with different names in a single POU.



■ Structure of instance

An instance consists of the following data areas.

Data area	Description
Local label area	Used to assign the local label of the function block.
Local latch label area	Used to assign the latch attribute local label of the function block.

■ Capacity of instance

The capacity of each data area of an instance should be calculated as follows.

- Local label area

Capacity of local label area of instance = Total capacity of data of local labels (except the ones with latch attribute) + Capacity of reserved area

Breakdown	Description
Capacity of local labels (except the ones with latch attribute)	Total capacity of the data areas used for local labels
Capacity of reserved area	The capacity of the area reserved to add local labels except the ones with latch attribute and local instances by executing the online program change function (fixed to 48 words)

- Local latch label area

Capacity of local latch label area of instances = Total capacity of data of local labels with latch attribute + Capacity of reserved area

Breakdown	Description
Capacity of latch attribute local labels	Total capacity of the data areas used for latch attribute local labels
Capacity of reserved area	The capacity of the area reserved to add latch attribute local labels and local instances by executing the online program change function (fixed to 16 words)

The local label area capacity is assigned by using the engineering tool. For details, refer to the following.

GX Works3 Operating Manual

Setting initial values

Initial values of local label

For the local label of a function block, an initial value can be set for each function block definition or instance. Local labels whose initial value can be set differ depending on the type and attribute.

Page 41 Definable data types and initial values

Initial values of instance

The following table summarizes the types of the initial values of instances.

Type	Description																												
Default initial value	<div><div>An initial value predefined for each data type. If an initial value is not set for the local label of a function block, the default initial value will be used.</div><div><div><div><div>FB1 definition</div><div>FB1</div></div><div><div>Local label definition (FB1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>AAA</td><td></td></tr><tr><td>BBB</td><td></td></tr><tr><td>CCC</td><td></td></tr></tbody></table><div>(1)</div></div><div><div>Global label definition</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_a</td><td>-</td></tr><tr><td>- AAA</td><td>0</td></tr><tr><td>- BBB</td><td>0</td></tr><tr><td>- CCC</td><td>0</td></tr></tbody></table><div><div>Local label definition (program block 1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_b</td><td>-</td></tr><tr><td>- AAA</td><td>0</td></tr><tr><td>- BBB</td><td>0</td></tr><tr><td>- CCC</td><td>0</td></tr></tbody></table><div>(2)</div></div></div></div><div><div>(1) The initial values have not been set for the local labels in the FB1 definition.</div><div>(2) The default initial values are used.</div></div></div></div>	Label name	Initial value	AAA		BBB		CCC		Label name	Initial value	FB1_a	-	- AAA	0	- BBB	0	- CCC	0	Label name	Initial value	FB1_b	-	- AAA	0	- BBB	0	- CCC	0
Label name	Initial value																												
AAA																													
BBB																													
CCC																													
Label name	Initial value																												
FB1_a	-																												
- AAA	0																												
- BBB	0																												
- CCC	0																												
Label name	Initial value																												
FB1_b	-																												
- AAA	0																												
- BBB	0																												
- CCC	0																												
FB definition initial value	<div><div>An initial value that is set when the local label of a function block is defined. If this initial value has been set, the same definition initial value will be used for all the instances.</div><div><div><div><div>FB1 definition</div><div>FB1</div></div><div><div>Local label definition (FB1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>AAA</td><td>1111</td></tr><tr><td>BBB</td><td>2222</td></tr><tr><td>CCC</td><td>3333</td></tr></tbody></table><div>(1)</div></div><div><div>Global label definition</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_a</td><td>-</td></tr><tr><td>- AAA</td><td>1111</td></tr><tr><td>- BBB</td><td>2222</td></tr><tr><td>- CCC</td><td>3333</td></tr></tbody></table><div><div>Local label definition (program block 1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_b</td><td>-</td></tr><tr><td>- AAA</td><td>1111</td></tr><tr><td>- BBB</td><td>2222</td></tr><tr><td>- CCC</td><td>3333</td></tr></tbody></table><div>(2)</div></div></div></div><div><div>(1) The initial values have been set for the local labels in the FB1 definition.</div><div>(2) All the instances of FB1 will be initialized by the same definition initial value.</div></div></div></div>	Label name	Initial value	AAA	1111	BBB	2222	CCC	3333	Label name	Initial value	FB1_a	-	- AAA	1111	- BBB	2222	- CCC	3333	Label name	Initial value	FB1_b	-	- AAA	1111	- BBB	2222	- CCC	3333
Label name	Initial value																												
AAA	1111																												
BBB	2222																												
CCC	3333																												
Label name	Initial value																												
FB1_a	-																												
- AAA	1111																												
- BBB	2222																												
- CCC	3333																												
Label name	Initial value																												
FB1_b	-																												
- AAA	1111																												
- BBB	2222																												
- CCC	3333																												
Instance initial value	<div><div>An initial value that is set for an instance included in the global label and program block local label definition.</div><div><div><div><div>FB1 definition</div><div>FB1</div></div><div><div>Local label definition (FB1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>AAA</td><td>1111</td></tr><tr><td>BBB</td><td>2222</td></tr><tr><td>CCC</td><td>3333</td></tr></tbody></table><div>(1)</div></div><div><div>Global label definition</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_a</td><td>-</td></tr><tr><td>- AAA</td><td>3333</td></tr><tr><td>- BBB</td><td>4444</td></tr><tr><td>- CCC</td><td>5555</td></tr></tbody></table><div><div>Local label definition (program block 1)</div><table><thead><tr><th>Label name</th><th>Initial value</th></tr></thead><tbody><tr><td>FB1_b</td><td>-</td></tr><tr><td>- AAA</td><td>7777</td></tr><tr><td>- BBB</td><td>8888</td></tr><tr><td>- CCC</td><td>9999</td></tr></tbody></table><div>(1)</div></div></div></div><div><div>(1) The initial value can be set for each instance in the FB1 definition.</div></div></div></div>	Label name	Initial value	AAA	1111	BBB	2222	CCC	3333	Label name	Initial value	FB1_a	-	- AAA	3333	- BBB	4444	- CCC	5555	Label name	Initial value	FB1_b	-	- AAA	7777	- BBB	8888	- CCC	9999
Label name	Initial value																												
AAA	1111																												
BBB	2222																												
CCC	3333																												
Label name	Initial value																												
FB1_a	-																												
- AAA	3333																												
- BBB	4444																												
- CCC	5555																												
Label name	Initial value																												
FB1_b	-																												
- AAA	7777																												
- BBB	8888																												
- CCC	9999																												

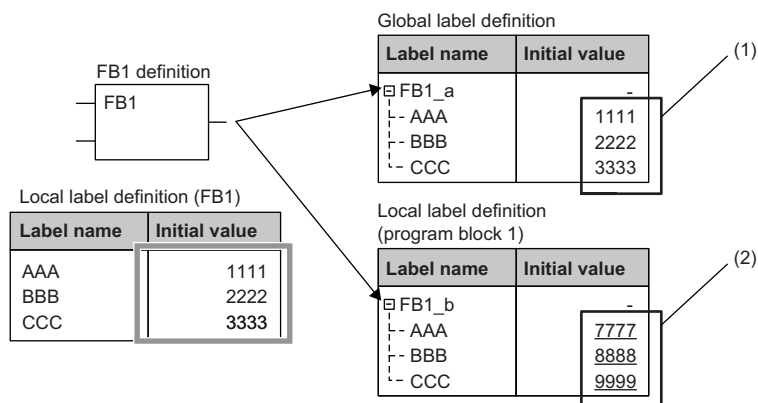
For function blocks, both the FB definition and instance initial values can be set.

If both initial values are set, the initial values used will take the following priority.

Priority	Type	Remarks
High ↑	Instance initial value	—
↓	FB definition initial value	—
Low	Default initial value	Used if neither the instance nor FB definition initial value has been set.

Point

If two instances of a function block for which the FB definition initial value has been set are created and the instance initial value is set for only one of them, the FB1 definition initial value will be used for the instance for which the instance initial value has not been set and the instance initial value will be used for the other.

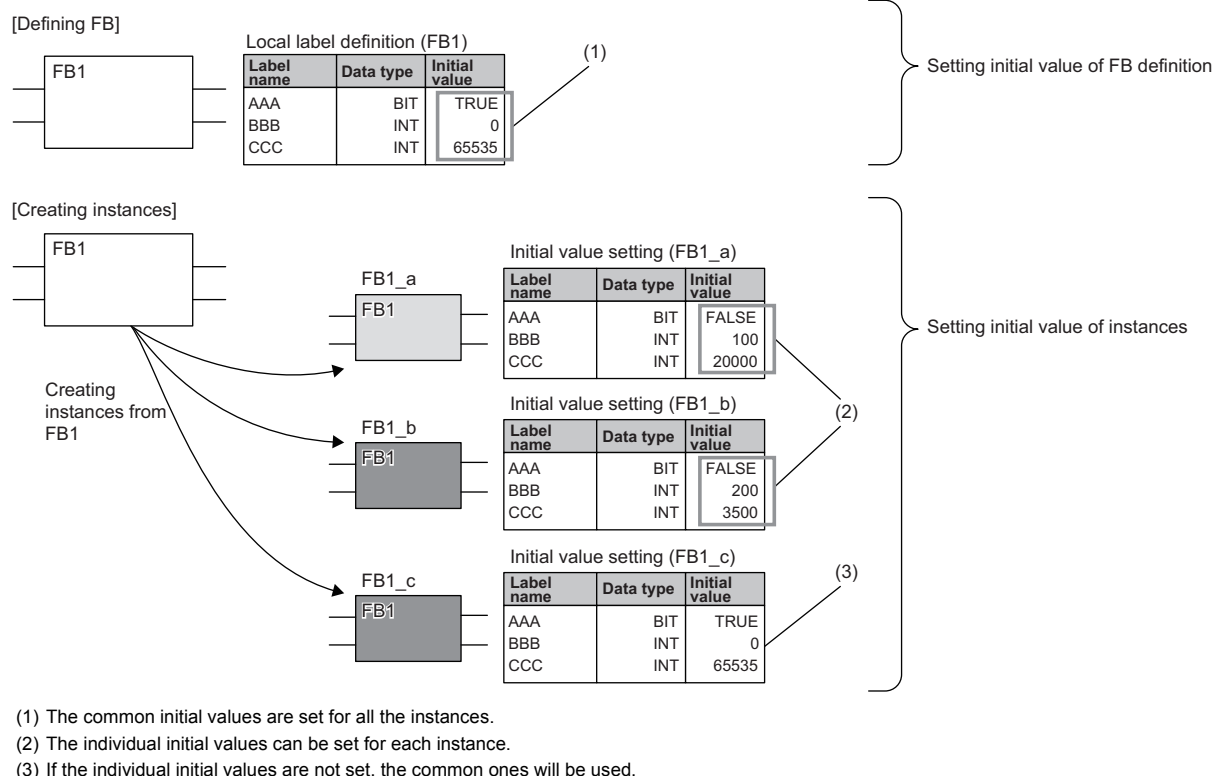


(1) When the instance initial value is not set, the FB definition initial value will be used.

(2) When the instance initial value is set, it will be used.

■Example

The following figure shows an example where the function block initial values are used.



EN and ENO

In the same way as a function, EN (enable input) and ENO (enable output) can also be appended to a function block to control execution processing.

☞ Page 16 EN and ENO

When the instance of a function to which EN/ENO has been appended is called, an actual argument must be assigned to EN.

Creating programs

The program of a function block can be created by using the engineering tool.

☞ [Navigation window] ⇒ [FB/FUN] ⇒ Right-click ⇒ [Add New Data]

The created program is stored in the FB/FUN file.

☞ [CPU Parameter] ⇒ [Program Setting] ⇒ [FB/FUN File Setting]

Up to 64 programs can be stored in one FB/FUN file.

For details on program creation, refer to the following.

Item	Reference
How to create function programs	☞ GX Works3 Operating Manual
Number of FB/FUN files that can be written to a CPU module	☞ MELSEC iQ-R CPU Module User's Manual (Startup)

■Types of program

There are two types of function blocks and the program of each function block type is stored in different ways.

- Macro type function block
- Subroutine type function block

For details, refer to the following.

☞ Page 27 Operation overview

The above cannot be selected for module function blocks, standard functions, and standard function blocks.

■Applicable devices and labels

The following table lists the devices and labels that can be used by function block programs.

○: Applicable, △: Applicable only in instructions (Cannot be used to indicate the program step.), ×: Not applicable

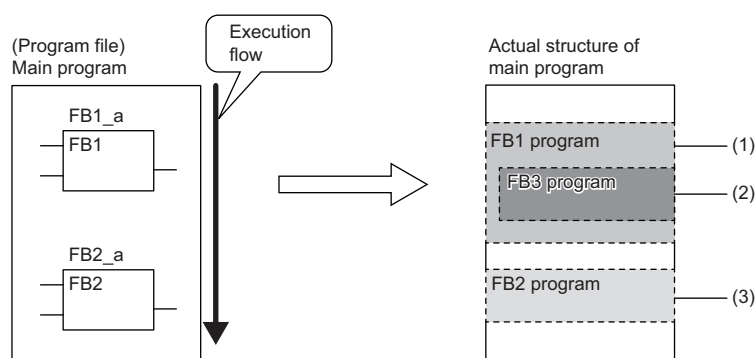
Type of device/label		Availability
Label (other than the pointer type)	Global label	○
	Local label	○
Label (pointer type)	Pointer type global label	△
	Pointer type local label	○
Device	Global device	○
	Local device	×
Pointer	Global pointer	△
	Local pointer	×

Operation overview

■Macro type function blocks

The program of a macro type function block is loaded by a calling source program along the execution flow. At the time of program execution, the loaded program is executed in the same way as the main program.

Use a macro type function block when giving a higher priority to the processing speed of the program.



(1) The FB1 program is loaded into the main program and executed.

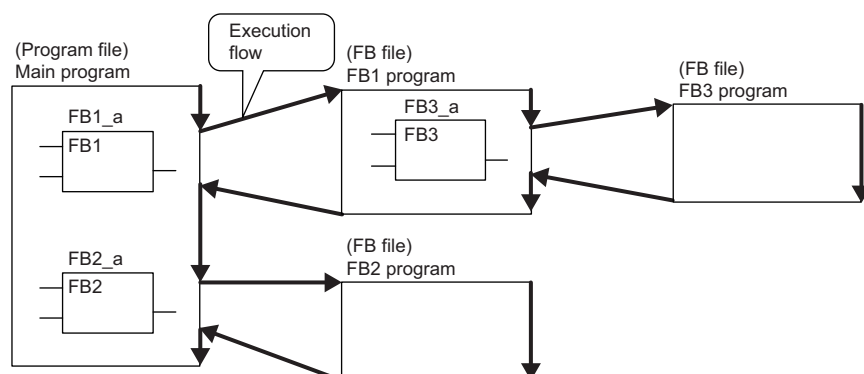
(2) FB3 is loaded into the FB1 program.

(3) The FB2 program is loaded into the main program and executed in the same way as the FB1 program.

■Subroutine type function blocks

The program of a subroutine type function block is stored in the FB/FUN file and called by the calling source program when executed.

Use a subroutine type function block to reduce the program size.

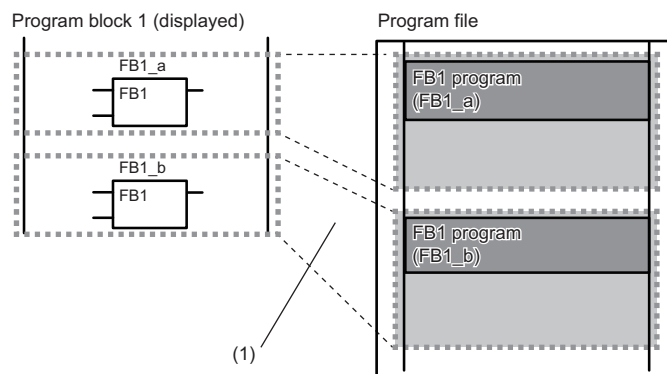


Up to 32 subroutine type function blocks, macro type function blocks, and functions in total can be nested.

Number of steps (Macro type function blocks)

■Calling source

When calling a macro type function block, the calling source loads the call-target program during compilation.



(1) The program is loaded in two or more call locations.

■Program

The number of steps required for a function block program is the total number of instruction steps, like usual programs.

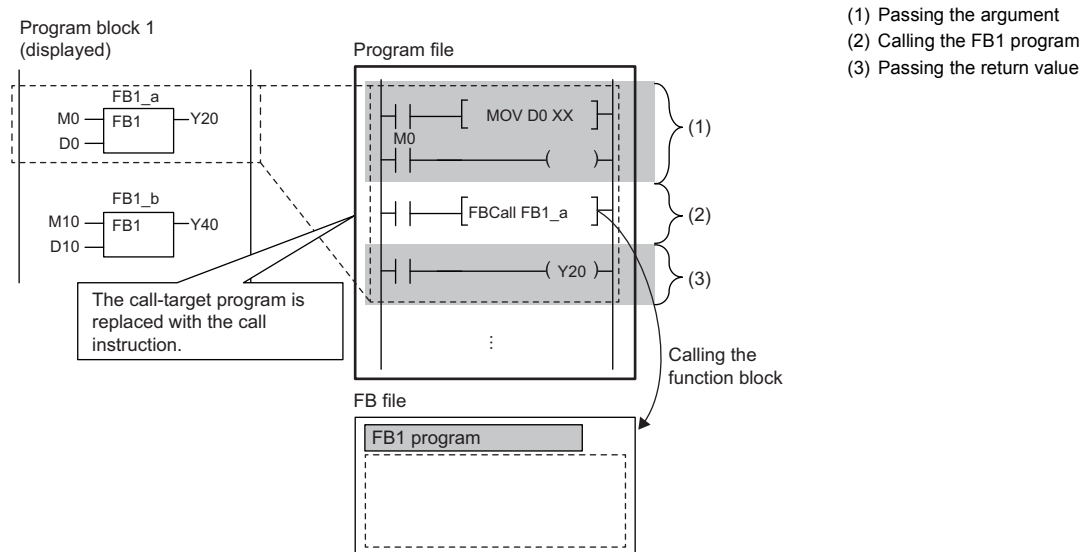
For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

Number of steps (Subroutine type function blocks)


■Calling source

When calling a subroutine type function block, the calling source generates the processing that passes the argument and return value before and after the call processing.



- Passing the argument

The instruction used to pass the argument differs depending on the class and data type of the argument. The following table summarizes the instructions that can be used to pass the argument.

Argument class	Data type	Instruction used	Number of steps
VAR_INPUT VAR_IN_OUT	Bit	LD+OUT LD+MOVB (Either of the instruction sets is used depending on the combination of programming language, function, and input argument used.)	For the number of steps required for each instruction, refer to the following.  MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)
	Word [unsigned]/bit string [16 bits] Double word [unsigned]/bit string [32 bits] Word [signed] Double word [signed]	LD+MOV LD+DMOV	
	Single-precision real number	LD+EMOV	
	Double-precision real number	LD+EDMOV	
	Time	LD+DMOV	
	String	LD+\$MOV	
	String [Unicode]	LD+\$MOV_WS	
	Array, Structure	LD+BMOV	

- Calling the program

A total of 10 steps are required to call the function block program.

- Passing the return value

The instruction and the number of steps used for passing the return value are identical to those for passing the argument.

Argument class	Data type	Instruction used	Number of steps
VAR_OUTPUT VAR_IN_OUT	Same as for passing the argument	Same as for passing the argument	Same as for passing the argument

- EN/ENO

The following table lists the number of steps required for EN/ENO.

Item	Number of steps
EN	3
ENO	2


Point

The number of steps may increase or decrease, depending on the following conditions.

- The actual argument or return value of the function block are index-modified.
- The address specifying the device exceeds 16 bits in length.
- Digit specification is performed.

■Program

The number of steps required for a function block program is the total number of instruction steps, like usual programs. For the number of steps required for each instruction, refer to the following.

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

3.4 Precautions

When a function is used

■Global pointer/local pointer/pointer type global labels

Global pointer, local pointer, and pointer type global labels cannot be used as labels indicating program steps in the function program.

When a function block is used

■Global pointer/local pointer/pointer type global labels

Global pointer, local pointer, and pointer type global labels cannot be used as labels indicating program steps in the function block program.

■When the index register is used

When the index register is used in the function block program, ladder programs for saving and returning the index register values are required to protect the values.

Setting the index register data to 0 when saving can prevent an error that could be caused by an index modification validity check. (Whether the device number exceeds the device range or not is checked.)

Ex.

A program that saves the values in the index register Z1 and Z2 before the program execution and returns the saved values after the program execution

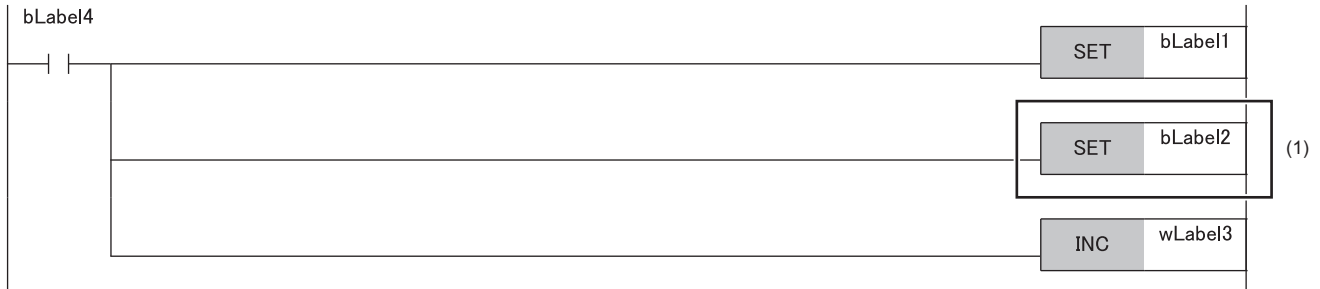


■When a conversion error occurs in VAR_INPUT, VAR_OUTPUT, or VAR_IN_OUT in a macro type function block

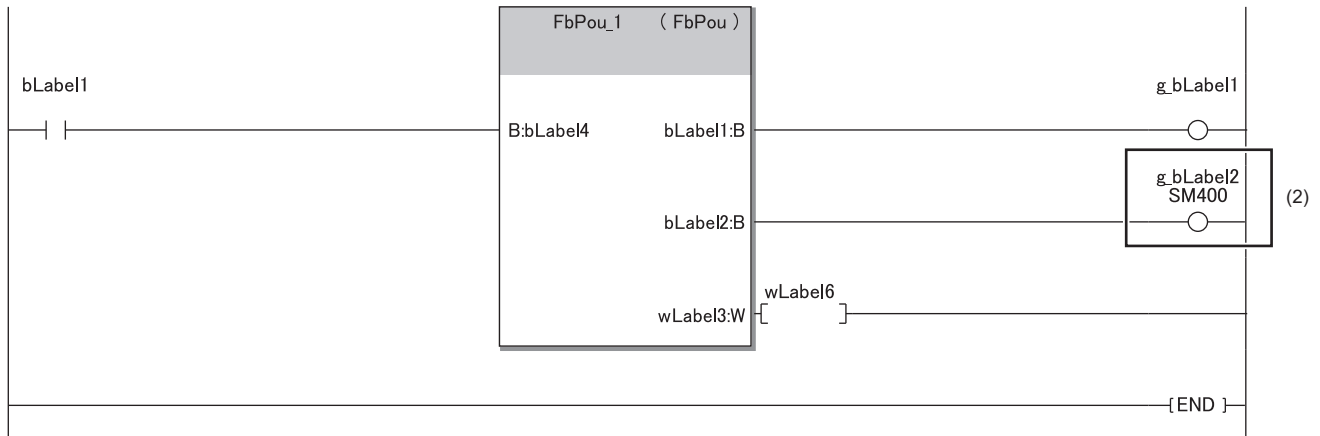
A program block that is the calling source of the function block or the function block may cause the error. In this case, check the inputs and outputs of the program block that is the calling source of the function block and the function block.

Ex.

A conversion error (1) occurs in VAR_OUTPUT in the macro type function block (FbPou)



If no error was found in (1), check the inputs and outputs (2) of the corresponding function block in the program block that is the calling source.



Since the output variables of the function block have been passed to the write-protected label/device, a conversion error has occurred in the above example.

■Specifying the start I/O number of intelligent function module

When accessing the buffer memory or I/O signals of the intelligent function module, specify the start I/O number using the index register.

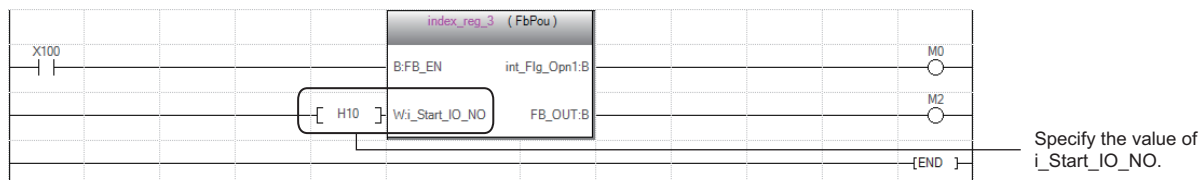
By receiving the start I/O number as an input variable, the same function block can be shared in multiple intelligent function modules without changing the start I/O number.

Ex.

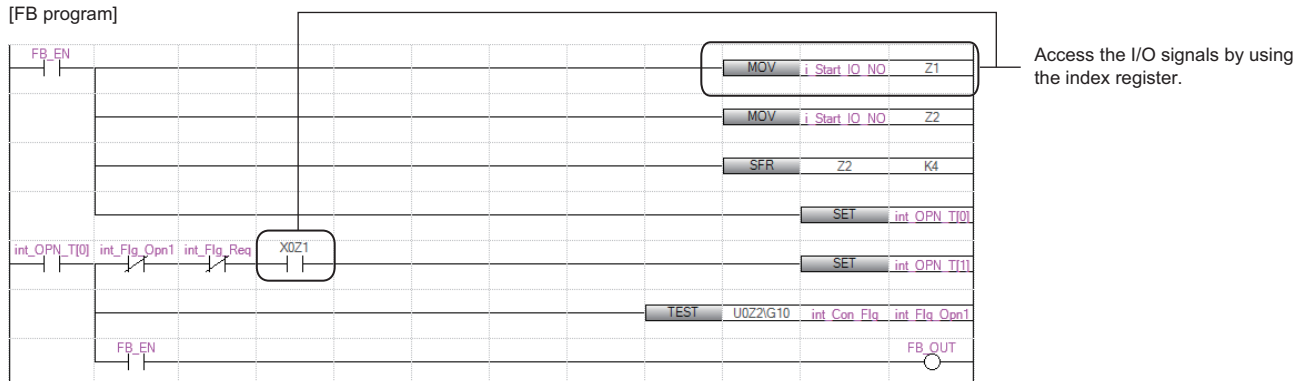
To access the I/O signals of the intelligent function module

Use the index register.

[Sequence program]



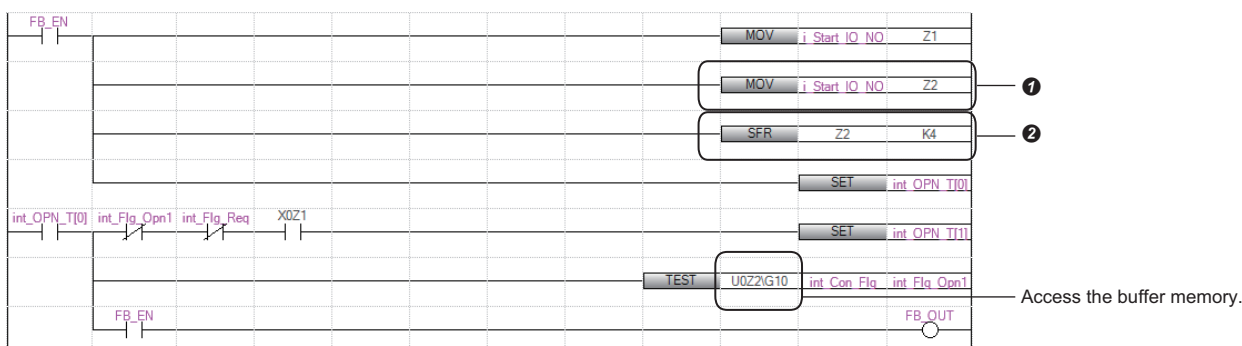
[FB program]



Ex.

To access the buffer memory of the intelligent function module

- 1 Input the start I/O number of the target intelligent function module in the index register.
- 2 Shift the four bits in the value to the right by using the SFR instruction, or use the quotient obtained by dividing the value by 16.



■Restrictions for module function blocks

The following describes the restrictions for the use of module function blocks.

- Do not turn off the contact of the MC instruction when calling a module function block between the MC instruction and MCR instruction.
- Do not perform the jump processing that prevents module function blocks from being called by the CJ instruction, SCJ instruction, and JMP instruction.
- Execute a subroutine program every scan when calling a module function block in the subroutine program. Do not perform the non-execution processing of a subroutine program by using the FCALL(P) instruction, EFCALL(P) instruction, or XCALL instruction.
- Do not call a module function block in an interrupt program or event execution type program.
- Do not call a module function block between the FOR and NEXT instructions, in the inline ST, or the control syntax of the structured text language (IF statement, FOR statement, and CASE statement.)
- Do not use the program control instructions (PSTOP(P) instruction, POFF(P) instruction, and PSCAN(P) instruction) to the program that calls a module function block.

■When the CPU module that controls the target module of a module function block is changed


The module function block used in the program will be deleted if the CPU module set to "Control PLC Settings" of "I/O Assignment" in system parameters is changed to another CPU module.

Copy the program into the project of another CPU module before changing the parameter setting.


■Changing operating parameters of module function blocks

Operating parameters (external variables) other than input and output labels of module function blocks can be changed on the label setting window by using the engineering tool.

- When the instance of a module function block is set to local label, change the parameters on the "Local Label Setting" window.

 [Project view] ⇒ [Program] ⇒ execution type ⇒ program file ⇒ program block ⇒ [Local Label]

- When the instance of a module function block is set to global label, change the parameters on the "Global Label Setting" window.

 [Project view] ⇒ [Label] ⇒ [Global Label]

Ex.

Local Label Setting window

ProgPou [PRG] [Local Label Setting]

<Filter>

	Label Name	Data Type	Comment
1	M_RCPU_Mayynchronization_Delay1OUT_00	M+RCPU_Mayynchronization_Delay1OUT_00A	
2	M_RJ71GF11_DeviceRead_00B_1	M+RJ71GF11_DeviceRead_00B	
3			

Extended Display: Automatic

M+RJ71GF11_DeviceRead_00B
(5) M+RJ71GF11

M_RJ71GF11_DeviceRead_00B_1(M+RJ71GF11_DeviceRead_00B)

	Class	Label Name	Data Type	Initial Value	Comment
VAR_INPUT	i_bEN		Bit		
VAR_INPUT	i_stModule		M+RJ71GF11		
VAR_INPUT	i_u2TargetAddress		Word (Unsigned)/Bit String [16-bit](0..1)		
VAR_INPUT	i_uDataLength		Word (Unsigned)/Bit String [16-bit]		
VAR_INPUT	i_s32TargetDevice		String(32)		
VAR_INPUT	i_uChannel		Word (Unsigned)/Bit String [16-bit]		
VAR_OUTPUT	o_bENO		Bit		
VAR_OUTPUT	o_bOK		Bit		
VAR_OUTPUT	o_bErr		Bit		
VAR_OUTPUT	o_uErrId		Word (Unsigned)/Bit String [16-bit]		
VAR_OUTPUT	o_uReadData		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbi_uCPU_Type		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbi_uResendCountMax		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbi_uTimeUnit		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbi_uMonitorTime		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbi_bStationSpecific		Bit		
VAR_PUBLIC	pbo_uResendCount		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbo_u4ErrTime		Word (Unsigned)/Bit String [16-bit](0..3)		
VAR_PUBLIC	pbo_uErrNetworkNo		Word (Unsigned)/Bit String [16-bit]		
VAR_PUBLIC	pbo_uErrStationNo		Word (Unsigned)/Bit String [16-bit]		
VAR	u18ControlData		Word (Unsigned)/Bit String [16-bit](0..17)		
VAR	bStart		Bit		
VAR	b2Comp		Bit(0..1)		

Set operating parameters in the "Initial Value" field.

3.5 When a Safety Program Is Used

A function used in a safety program is called a safety function, and a function block used in a safety program is called a safety function block. Information not described in this section is same as that of standard functions and function blocks. (📖 Page 15 Functions (FUN), 📖 Page 20 Function Blocks (FB))

Safety functions (Safety FUN)

This section describes safety functions.

3

Creating programs

■Applicable devices and labels

The following table lists the devices and labels that can be used in safety functions.

○: Applicable, ×: Not applicable

Type of device/label		Availability
Label (other than the pointer type)	Global label	×
	Local label	×
	Standard/safety shared label	×
	Safety global label	×
	Safety local label	○*1
Label (pointer type)	Pointer type global label	×
	Pointer type local label	×
Device	Global device	×
	Local device	×
	Safety global device	○
	Safety local device	×
Pointer	Global pointer	×
	Local pointer	×

*1 The following data types cannot be used.

Timer, retentive timer, counter, long timer, long retentive timer, and long counter

Number of steps

■Passing the argument

When calling a safety function, the calling source generates the processing that passes the argument before and after the call processing. The instruction used to pass the argument differs depending on the class and data type of the argument. The following table summarizes the instructions that can be used to pass the argument.

○: Applicable, ×: Not applicable

Argument class	Data type	Instruction used	Availability
VAR_INPUT	Bit	LD+OUT	○
	Word [unsigned]/bit string [16 bits]	LD+MOV	○
	Double word [unsigned]/bit string [32 bits]	LD+DMOV	
	Word [signed]		
	Double word [signed]		
	Single-precision real number	LD+EMOV	×
	Double-precision real number	LD+EDMOV	×
	Time	LD+DMOV	×
	String	LD+\$MOV	×
	String [Unicode]	LD+\$MOV_WS	×
	Array, Structure	LD+BMOV	○

For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

Safety function blocks (Safety FB)

This section describes safety function blocks.

Instances

■Structure of instance

An instance of a safety function block consists of the following data areas.

○: Applicable, ×: Not applicable

Data area	Description	Availability
Local label area	Used to assign the local label of the function block.	○
Local latch label area	Used to assign the latch attribute local label of the function block.	×

Creating programs

■Applicable devices and labels

The following table lists the devices and labels that can be used in safety function blocks.

○: Applicable, ×: Not applicable

Type of device/label		Availability
Label (other than the pointer type)	Global label	×
	Local label	×
	Standard/safety shared label	○
	Safety global label	○
	Safety local label	○
Label (pointer type)	Pointer type global label	×
	Pointer type local label	×
Device	Global device	×
	Local device	×
	Safety global device	○
	Safety local device	×
Pointer	Global pointer	×
	Local pointer	×

Number of steps (subroutine type function blocks)

■Passing the argument

When calling a safety function block, the calling source generates the processing that passes the argument and return value before and after the call processing. The instruction used to pass the argument differs depending on the class and data type of the argument. The following table summarizes the instructions that can be used to pass the argument.

○: Applicable, ×: Not applicable

Argument class	Data type	Instruction used	Availability
VAR_INPUT VAR_IN_OUT	Bit	LD+OUT	○
	Word [unsigned]/bit string [16 bits]	LD+MOV LD+DMOV	○
	Double word [unsigned]/bit string [32 bits]		
	Word [signed]		
	Double word [signed]		
	Single-precision real number	LD+EMOV	×
	Double-precision real number	LD+EDMOV	×
	Time	LD+DMOV	×
	String	LD+\$MOV	×
	String [Unicode]	LD+\$MOV_WS	×
	Array, Structure	LD+BMOV	○

For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)


4 LABELS

A label is a variable consisting of a specified string used in I/O data or internal processing.

Using labels in programming enables creation of programs without being aware of devices and buffer memory sizes.

For this reason, a program using labels can be reused easily even in a system having a different module configuration.

When labels are used, there are some precautions on programming and functions used. For details, refer to the following.

 Page 48 Precautions

4.1 Label Types

There are two types of labels described in this manual.

- Global labels
- Local labels

Global labels

A global label is a label that provides the same data within a single project. It can be used in all programs in the project.

A global label can be used in program blocks and function blocks.

The settings of a global label include a label name, class, and data type.

By opening global labels, they can be referenced from GOT and other stations, and can be used for monitoring and accessing data.

■Device assignment

Devices can be assigned to global labels.

Item	Description
Label to which no device is assigned	<ul style="list-style-type: none">• Programming without being aware of devices is possible.• Defined labels are allocated to the label area or latch label area in the device/label memory.
Label to which a device is assigned	<ul style="list-style-type: none">• If a device is to be programmed as a label against a device that is being used for input or output, the device can be assigned directly.• Defined labels are allocated to the device area in the device/label memory.

Local labels

A local label is a label that can be used only in the declared POU. Local labels outside the declared POU cannot be used.

The settings of a local label include a label name, class, and data type.



There are other types of labels available in addition to the global labels and local labels.

[System labels]

A system label is a label that provides the same data in all projects compatible with iQ Works. It can be referenced from the GOT and the CPU modules and Motion controllers on other stations, and used for monitoring and accessing data.

For details, refer to the following.

 iQ Works Beginner's Manual

[Module labels]

A module label is a label defined uniquely by each module. A module label is automatically generated by the engineering tool from the module used, and can be used as a global label.

For details, refer to the following.

 Function Block Reference for the module used

4.2 Classes

The label class indicates from which POU and how a label can be used.
Different classes can be selected depending on the type of POU.

Global label				
Class	Description	Applicable POU		
		Program block	Function block	Function
VAR_GLOBAL	A common label that can be used in both program blocks and function blocks	○	○	×
VAR_GLOBAL_CONSTANT	A common constant that can be used in both program blocks and function blocks	○	○	×
VAR_GLOBAL_RETAIN	A latch type label that can be used in both program blocks and function blocks	○	○	×
Local label				
Class	Description	Applicable POU		
		Program block	Function block	Function
VAR	A label that can be used within the range of a declared POU. This label cannot be used in other POUs.	○	○	○
VAR_CONSTANT	A constant that can be used within the range of a declared POU. This label cannot be used in other POUs.	○	○	○
VAR_RETAIN	A latch type label that can be used within the range of a declared POU. This label cannot be used in other POUs.	○	○	×
VAR_INPUT	A label that inputs a value into a function or function block. This label receives a value, and the received value cannot be changed in a POU.	×	○	○
VAR_OUTPUT	A label that outputs a value from a function or function block	×	○	○
VAR_OUTPUT_RETAIN	A latch type label that outputs a value from a function or function block	×	○	×
VAR_IN_OUT	A local label that receives a value and outputs the value from a POU. The value can be changed in a POU.	×	○	×
VAR_PUBLIC	A label that can be accessed from other POUs	×	○	×
VAR_PUBLIC_RETAIN	A latch type label that can be accessed from other POUs	×	○	×

4.3 Data Types

The data types of a label are classified according to the bit length, processing method, and value range.

There are two data types.

- Primitive data type
- Generic data type (ANY type)

Primitive data type

The following table lists the data types included in the primitive data type.

Data type		Description	Value range	Bit length
Bit	BOOL	Represents the alternative status, such as on or off.	0 (FALSE), 1 (TRUE)	1 bit
Word [unsigned]/bit string [16 bits]	WORD	16-bit array	0 to 65535	16 bits
Double word [unsigned]/bit string [32 bits]	DWORD	32-bit array	0 to 4294967295	32 bits
Word [signed]	INT	Positive and negative integer values	-32768 to 32767	16 bits
Double word [signed]	DINT	Positive and negative double-precision integer values	-2147483648 to 2147483647	32 bits
Single-precision real number	REAL	Numerical values of decimal places (single-precision real number values) Number of significant digits: 7 (6 digits of decimal places)* ¹	-2 ¹²⁸ to -2 ⁻¹²⁶ , 0, 2 ⁻¹²⁶ to 2 ¹²⁸ E-3.402823+38 to E-1.175495-38, 0, E1.175495-38 to E3.402823+38	32 bits
Double-precision real number	LREAL	Numerical values of decimal places (double-precision real number values) Number of significant digits: 15 (14 digits of decimal places)* ¹	-2 ¹⁰²⁴ to -2 ⁻¹⁰²² , 0, 2 ⁻¹⁰²² to 2 ¹⁰²⁴ E-1.79769313486231+308 to E-2.22507385850721-308, 0, E2.22507385850721-308 to E1.79769313486231+308	64 bits
Time* ²	TIME	Numerical values as day, hour, minute, second, and millisecond	T#-24d20h31m23s648ms to T#24d20h31m23s647ms* ³	32 bits
String	STRING	Characters represented by ASCII code or Shift JIS code	255 one-byte characters maximum	Variable
String [Unicode]	WSTRING	Characters represented by Unicode	255 characters maximum	Variable
Timer	TIMER	Structure corresponding to the device, timer (T)	☞ Page 40 Timer and counter data types	
Retentive timer	RETENTIVETIMER	Structure corresponding to the device, retentive timer (ST)		
Long timer	LTIMER	Structure corresponding to the device, long timer (LT)		
Long retentive timer	LRETENTIVETIMER	Structure corresponding to the device, timer (LST)		
Counter	COUNTER	Structure corresponding to the device, counter (C)		
Long counter	LCOUNTER	Structure corresponding to the device, counter (LC)		
Pointer	POINTER	Type corresponding to the device, pointer (P) ☞ MELSEC iQ-R CPU Module User's Manual (Application)		

*1 If the single-precision real number exceeds the significant digit, 7, the 8th digit is rounded off. If the double-precision real number exceeds the significant digit, 15, the 16th digit is rounded off.

*2 The time type is used in a time data type function of standard functions. For standard functions, refer to the following.

☞ MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

*3 When using a constant for the time type label, add "T#" at the beginning of the constant.

- The bit data in the word type label can be used by specifying a bit number.
- The bit type array label can be used as 16-bit or 32-bit data by specifying the number of digits.

For the bit specification and digit specification methods, refer to the following.

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

■Timer and counter data types


The data types of the timer, counter, long counter, retentive timer, long retentive timer, and long timer are the structures having a contact, coil, or current value.

Data type		Member name	Member data type	Description	Value range
Timer	TIMER	S	Bit	Indicates a contact. The operation is the same as the contact (TS) of a timer device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (TC) of a timer device.	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/bit string [16 bits]	Indicates the current value. The operation is the same as the current value (TN) of a timer device.	0 to 65535 ^{*1}
Retentive timer	RETENTIVETIMER	S	Bit	Indicates a contact. The operation is the same as the contact (STS) of a retentive timer device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (STC) of a retentive timer device.	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/bit string [16 bits]	Indicates the current value. The operation is the same as the current value (STN) of a retentive timer device.	0 to 65535 ^{*1}
Long timer	LTIMER	S	Bit	Indicates a contact. The operation is the same as the contact (LTS) of a long timer device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (LTC) of a long timer device.	0 (FALSE), 1 (TRUE)
		N	Double word [unsigned]/bit string [32 bits]	Indicates the current value. The operation is the same as the current value (LTN) of a long timer device.	0 to 4294967295 ^{*1}
Long retentive timer	LRETENTIVETIMER	S	Bit	Indicates a contact. The operation is the same as the contact (LSTS) of a long retentive timer device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (LSTC) of a long retentive timer device.	0 (FALSE), 1 (TRUE)
		N	Double word [unsigned]/bit string [32 bits]	Indicates the current value. The operation is the same as the current value (LSTN) of a long retentive timer device.	0 to 4294967295 ^{*1}
Counter	COUNTER	S	Bit	Indicates a contact. The operation is the same as the contact (CS) of a counter device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (CC) of a counter device.	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/bit string [16 bits]	Indicates the current value. The operation is the same as the current value (CN) of a counter device.	0 to 65535
Long counter	LCOUNTER	S	Bit	Indicates a contact. The operation is the same as the contact (LCS) of a long counter device.	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates a coil. The operation is the same as the coil (LCC) of a long counter device.	0 (FALSE), 1 (TRUE)
		N	Double word [unsigned]/bit string [32 bits]	Indicates the current value. The operation is the same as the current value (LCN) of a long counter device.	0 to 4294967295

^{*1} The unit of the current value is set in CPU parameters ("Timer Limit Setting").

For details on the operation of each device, refer to the following.

 MELSEC iQ-R CPU Module User's Manual (Application)

The specification method of each member is the same as that of the structure data type. ( Page 45 Structures)

Generic data type (ANY type)

The generic data type is the data type of the labels which summarize several primitive data types.

Generic data types are used when multiple data types are allowed for function and function block arguments and return values.

Labels defined in generic data types can be used in any sub-level data type.

For the types of generic data types and the primitive data types, refer to the following.

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

Definable data types and initial values

The following tables list the definable data types and initial value setting possibilities for each label class.

Global label		
Class	Definable data type	Initial value setting possibility
VAR_GLOBAL	Primitive data type, array, structure, function block	○
VAR_GLOBAL_CONSTANT	Primitive data type ^{*1}	×
VAR_GLOBAL_RETAIN	Primitive data type ^{*1} , array, structure	○
Local label (program block)		
Class	Definable data type	Initial value setting possibility
VAR	Primitive data type, array, structure, function block	○
VAR_CONSTANT	Primitive data type ^{*1}	×
VAR_RETAIN	Primitive data type ^{*1} , array, structure	○
Local label (function)		
Class	Definable data type	Initial value setting possibility
VAR	Primitive data type ^{*2} , array, structure	×
VAR_CONSTANT	Primitive data type ^{*1}	×
VAR_INPUT	Primitive data type ^{*1*2} , array, structure	×
VAR_OUTPUT		×
Return value		×
Local label (function block)		
Class	Definable data type	Initial value setting possibility
VAR	Primitive data type, array, structure, function block	○
VAR_CONSTANT	Primitive data type ^{*1}	×
VAR_RETAIN	Primitive data type ^{*1} , array, structure	○
VAR_INPUT		○
VAR_OUTPUT		○
VAR_OUTPUT_RETAIN		○
VAR_IN_OUT		×
VAR_PUBLIC		○
VAR_PUBLIC_RETAIN		○

*1 The pointer type cannot be defined.

*2 None of the timer, retentive timer, long timer, counter, long timer, long retentive timer, and long counter types can be defined.

Point

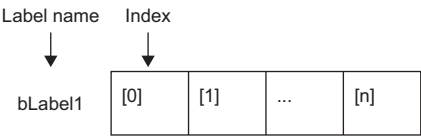
- The initial value of the global label where the device has been assigned follows that in the device.
- The initial value of the function block follows the local label setting within the function block.
- The initial value of the structure type follows that of the structure definition.

4.4 Arrays

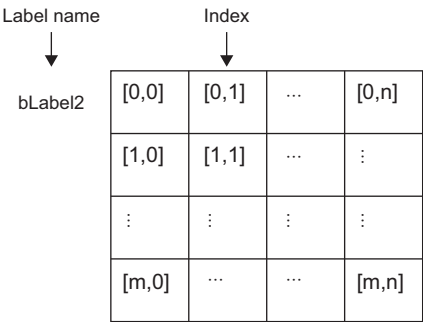
An array represents a consecutive aggregation of same data type labels as a single name.

Primitive data types and structures can be defined as arrays.

• One-dimensional array



• Two-dimensional array



Defining arrays

■Array elements

When an array is defined, the number of elements, or the length of array, must be determined. For the range of the number of elements, refer to the following.

📖 Page 43 Range of the number of array elements

■Dimension number of multidimensional array

Up to three-dimensional array can be defined.

■Definition format

The following table lists definition format.

The range from the array start value to the array end value is the number of elements.

Number of array dimensions	Format	Remarks
One dimension	Array of primitive data type/structure name (array start value..array end value)	• For the primitive data type: 📖 Page 39 Primitive data type • For the structure name: 📖 Page 45 Structures
	[Definition example] Bit (0..15)	
Two dimensions	Array of primitive data type/structure name (array start value..array end value, array start value..array end value)	
	[Definition example] Bit (0..1, 0..15)	
Three dimensions	Array of primitive data type/structure name (array start value..array end value, array start value..array end value, array start value..array end value)	
	[Definition example] Bit (0..2, 0..1, 0..15)	

■Initial value

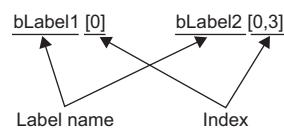
One initial value can be set for a single array definition. (Different initial values cannot be set for each element.)

The same initial value is stored in all the array elements.

How to use arrays

To use an array, add an index enclosed by '[']' after each label name to identify individual labels.

An array with two or more dimensions should be represented with indexes delimited by a comma (,) in '[']'.

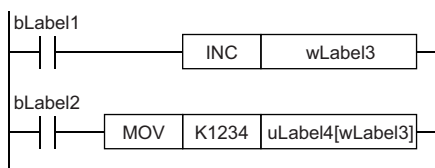


The following table lists the types of indexes that can be specified for arrays.

Type	Specification example	Remarks
Constant	bLabel1[0]	An integer can be specified.
Device	bLabel1[D0]	A word device, double-word device, decimal constant, or hexadecimal constant can be specified. (ST, LST, G, and HG cannot be specified.)
Label	bLabel1[uLabel2]	The following data types can be specified. <ul style="list-style-type: none"> • Word [unsigned]/bit string [16 bits] • Double word [unsigned]/bit string [32 bits] • Word [signed] • Double word [signed]
Expression	bLabel1[5+4]	Expressions can be specified only in ST language.

Point

- The data storage location becomes dynamic by specifying a label for the array index. This enables arrays to be used in a program that executes loop processing. The following is a program example that consecutively stores "1234" in the "uLabel4" array.



- The element number of the array can be omitted in ladder diagram. If the element number is omitted, it is regarded as the start number and converted. For example, when the defined label name is "boolAry" and the data type is "Bit (0..2, 0..2)", the operation of "boolAry[0,0]" is the same as that of "boolAry".
- When a multidimensional array is specified as setting data of instructions, functions, and function blocks that use arrays, the rightmost element is regarded as a one-dimensional array.

Range of the number of array elements

The maxim number of array elements varies depending on the data type.

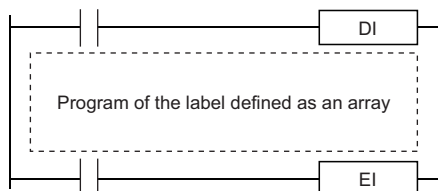
Data type	Setting range
Bit Word [unsigned]/bit string [16 bits] Word [signed] Timer Counter Retentive timer	1 to 2147483648
Double word [unsigned]/bit string [32 bits] Double word [signed] Single-precision real number Time Long counter Long retentive timer Long timer	1 to 1073741824
Double-precision real number	1 to 536870912
String	1 to 2147483648 ÷ String length
String [Unicode]	1 to 1073741824 ÷ String length

Precautions

■When an interrupt program is used

When a label or device is specified for the array index, the operation is performed with a combination of multiple instructions. For this reason, if an interrupt occurs during operation of the label defined as an array, data inconsistency may occur producing an unintended operation result.

To prevent data inconsistency, create a program using the DI/EI instructions that disables/enables interrupt programs as shown below.



For the DI/EI instructions, refer to the following.

📖 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

■Array elements

When accessing the element defined in an array, access it within the range of the number of elements.

If a constant out of the range defined for the array index is specified, a compile error will occur.

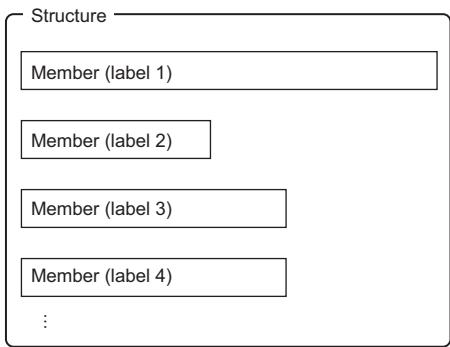
If the array index is specified with data other than a constant, a compile error will not occur. The processing will be performed by accessing another label area or latch label area.

4.5 Structures

A structure is a data type containing one or more labels and can be used in all POUs.
Members (labels) included in a structure can be defined even when their data types are different.

Creating structures

To create a structure, first define the structure, and then define members in the structure.



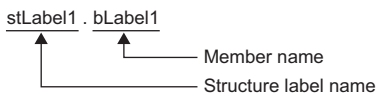
4

How to use structures

To use a structure, register a label using the defined structure as the data type.
To specify each member in a structure, add the member name after the structure label name with a period '.' as a delimiter in between.

Ex.

Specifying a member in the structure

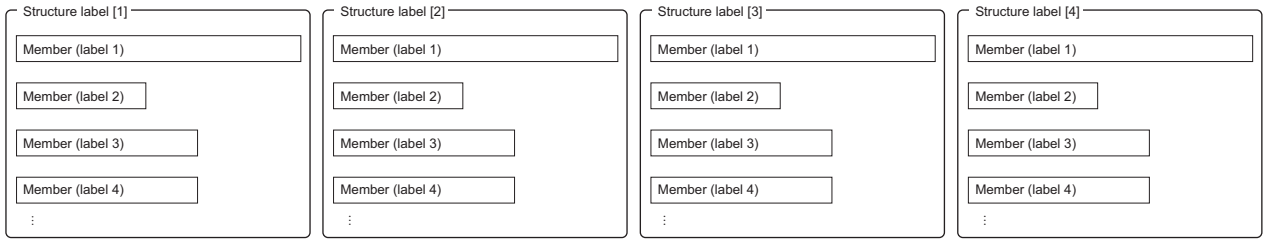


Point

- When labels are registered by defining multiple data types in a structure and used in a program, the order the data is stored after operation is not the order the data types were defined. When programs are converted using the engineering tool, labels are classified into type and data type, and then assigned to the memory (memory assignment by packing blocks).
- GX Works3 Operating Manual
- If the label of a structure is specified for an instruction that uses control data (a group of operands that determines operation of the instruction), the labels are not assigned in the order defined by packing blocks.

Structure arrays

A structure can also be used as an array.

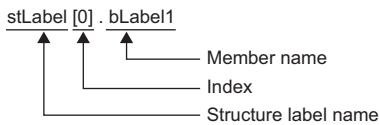


When a structure is declared as an array, add an index enclosed by '[']' after the structure label name.

A structure array can also be specified as an argument of a function or function block.

Ex.

Specifying an element of a structure declared as an array



Data types that can be specified

The following data types can be specified as structure members.

- Primitive data type
- Pointer type
- Array
- Other structures

Types of structures

Each of the following labels is predefined as a structure.

Type	Reference
Module label	Function Block Reference for the module used Page 39 Data Types
Timer type	
Retentive timer type	
Counter type	
Long timer type	
Long retentive timer type	
Long counter type	

4.6 Constants

Types of constants

The following table provides information on how to set constants to labels.

Applicable data type	Type	How to set	Example
Bit	Boolean	Set FALSE or TRUE.	TRUE, FALSE
	Binary	Add "2#" before the binary number to be used.	2#0, 2#1
	Octal	Add "8#" before the octal number to be used.	8#0, 8#1
	Decimal	Directly enter the decimal number to be used. Or, add "K" before the number.	0, 1, K0, K1
	Hexadecimal	Add "16#" before the hexadecimal number to be used. Or, add "H" before the number.	16#0, 16#1, H0, H1
<ul style="list-style-type: none">• Word [unsigned]/bit string [16 bits]• Double word [unsigned]/bit string [32 bits]• Word [signed]• Double word [signed]	Binary ^{*1}	Add "2#" before a binary number.	2#0010, 2#01101010, 2#1111_1111
	Octal ^{*1}	Add "8#" before an octal number.	8#0, 8#337, 8#1_1
	Decimal ^{*1}	Directly enter a decimal number, or add "K" before the number.	123, K123, K-123, 12_3
	Hexadecimal ^{*1}	Add "16#" before a hexadecimal number. Or, add "H" before the number.	16#FF, HFF, 16#1_1
<ul style="list-style-type: none">• Single-precision real number• Double-precision real number	Real number ^{*1}	Directly enter a real number including the decimal point and numerical values of decimal places. Or, add "E" before the number.	2.34, E2.34, E-2.34, 3.14_15
	Real number (exponential notation)	Add "E" before the exponential notation or real number, and add "+" or "-" before the exponent.	1.0E6, E1.001+5, 1.0E-6, E1.001-5
<ul style="list-style-type: none">• String• String [Unicode]	String String [Unicode]	Enclose a string in single quotation marks (') or double quotation marks (").	'ABC', "ABC"
Time	Time	Add "T#" at the beginning.	T#1h, T#1d2h3m4s5ms, TIME#1h

*1 In the notation of binary, octal, decimal, hexadecimal, and real numbers, the numbers can be separated using an underscore "_" to make programs easy to see. (The underscores are ignored in program processing.)

Using "\$" in a string type constant

"\$" is used as an escape sequence.

Two hexadecimal digits following "\$" is recognized as an ASCII code and the character corresponding to the ASCII code is inserted into the string.

If the two hexadecimal digits following "\$" do not correspond to the ASCII code, a conversion error occurs.

Note, however, that no error occurs when the character following "\$" is one of the following.

Notation	Symbol used in a string or printer code
\$\$	\$
\$'	'
\$"	"
\$L or \$l	Line feed
\$N or \$n	New line
\$P or \$p	Page feed
\$R or \$r	Return
\$T or \$t	Tab

4.7 Precautions

Functions with restrictions

The following functions have restrictions on the use of labels.

Item	Description
CPU parameter	<ul style="list-style-type: none"> • Trigger of an event execution type program • Refresh setting among multiple CPU modules Use devices because global labels nor local labels cannot be specified for these functions. ^{*1}
Module parameter	
	<ul style="list-style-type: none"> • Predefined protocol support function
	<ul style="list-style-type: none"> • Refresh setting of intelligent function module • Refresh setting of network module (SB/SW only) Use module labels for these functions. Use devices if module labels are not used. ^{*1}
	<ul style="list-style-type: none"> • Refresh setting of network module (other than SB/SW) Use devices because global labels nor local labels cannot be specified for these functions. ^{*1}
Data logging function Memory dump function Real-time monitor function	Use devices if there is a possibility for using these functions because global labels nor local labels cannot be specified for these functions. ^{*1} If a device cannot be assigned to the global label, add the transfer instruction to the scan program so that the data of the global label is copied to a specified device every scan, and then use the copy-target device.

^{*1} Global labels can be used as devices by assigning a device.

■Defining and using a global label with a device assigned

Define a global label following the procedure below, and use it when the functions having restriction on the use of labels are executed.

Since the device area in the device/label memory is used, secure the device area capacity.

1. Secure the device area to be used.

 [CPU Parameter] ⇒ [Memory/Device Setting] ⇒ [Device/Label Memory Area Setting]

2. Define a label as a global label, and assign a device manually.

3. Use the label defined in step 2 for the functions having no restrictions on the use of labels. Use the device assigned to the label for the function having restrictions on the use of labels.

■Copying the label data into a specified device

Copy the label data into a specified device following the procedure below, and use the copy-target device.

Since the device area in the device/label memory is used, secure the device area capacity.

1. Secure the device area to be used.

 [CPU Parameter] ⇒ [Memory/Device Setting] ⇒ [Device/Label Memory Area Setting]

2. Create a program using the label. The following is the program example for copying the data. (The data logging function uses the data in udLabel1.)



3. Use the device where the data has been transferred in step 2 for the function having restrictions on the use of labels. (In the program example in step 2, use D0.)

Point

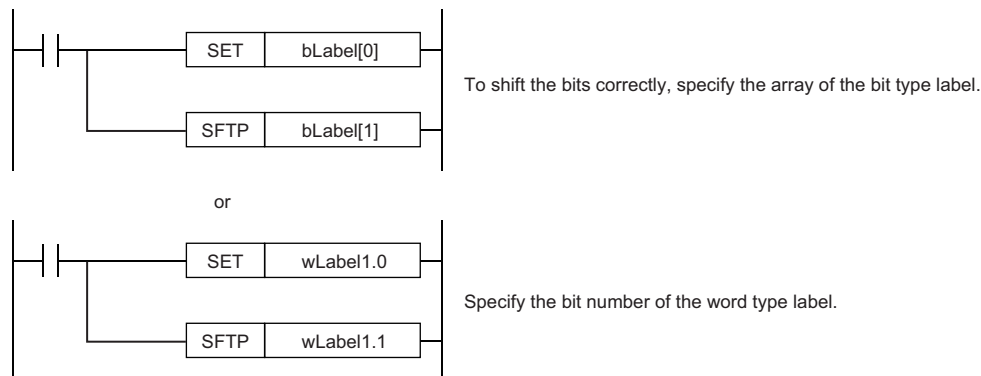
- The number of steps increases because of the transfer instruction. (The scan time increases.)
- Decide the transfer instruction position considering the timing of writing data to the label and executing the function.

Precautions for creating programs

When specifying a label as an operand used in instructions, match the data type of the label with that of the operand. In addition, when specifying a label as an operand used in instructions that control continuous data, specify the data range used in instructions within the data range of the label.

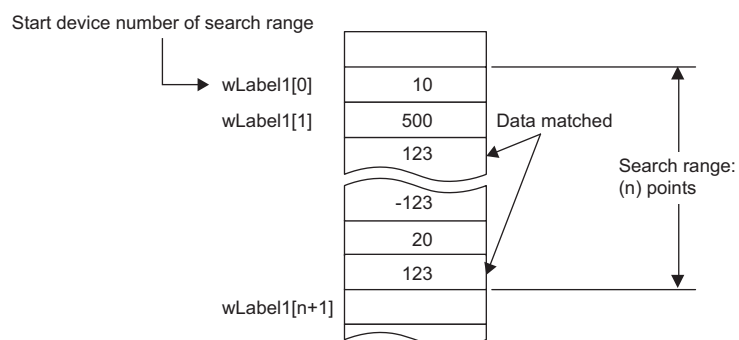
Ex.

SFT(P) instruction



Ex.

SER(P) instruction



Specify a label which has a larger data range than the search range (n) points.

Restrictions on naming labels

The following restrictions apply when naming labels.

- Start the name with a character or underline (_). Numbers cannot be used at the beginning of label names.
- Reserved words cannot be used.

For details on the reserved words, refer to the following.

GX Works3 Operating Manual

4.8 When a Safety Program Is Used

A label used in safety programs is called a safety label. Information not described in this section is same as that of standard labels. (👉 Page 37 LABELS)

Safety label types

There are three safety label types. Only the following labels can be used in safety programs.

- Safety global label*1
- Standard/safety shared label*2
- Safety local label

*1 Safety devices can be assigned.

*2 Can be used in standard programs and standard function blocks as well.



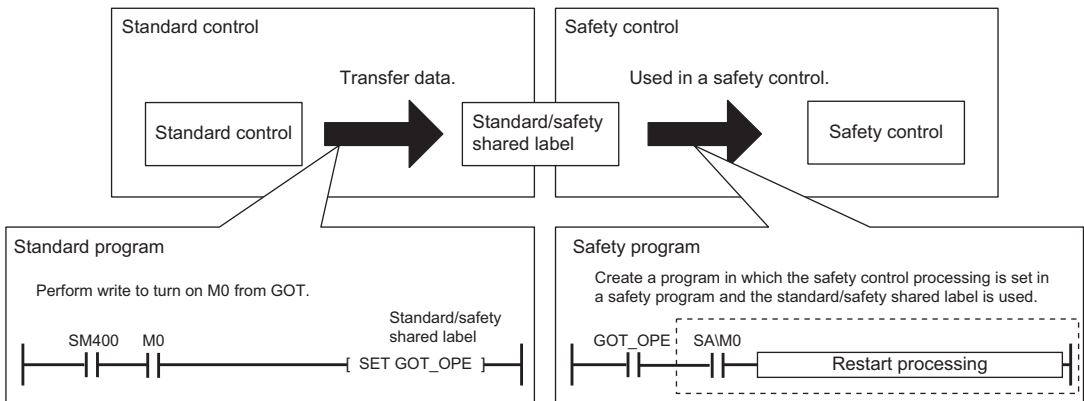
An initial value cannot be set to safety labels in the CPU parameter. For details, refer to the following.

📖 MELSEC iQ-R CPU Module User's Manual (Application)

How to use standard/safety shared labels

A standard/safety shared label is used to pass device data from a safety program to a standard program, and vice versa. When a standard/safety shared label is used in a safety program as shown in the examples below, the program needs to be created so that the safety state is secured.

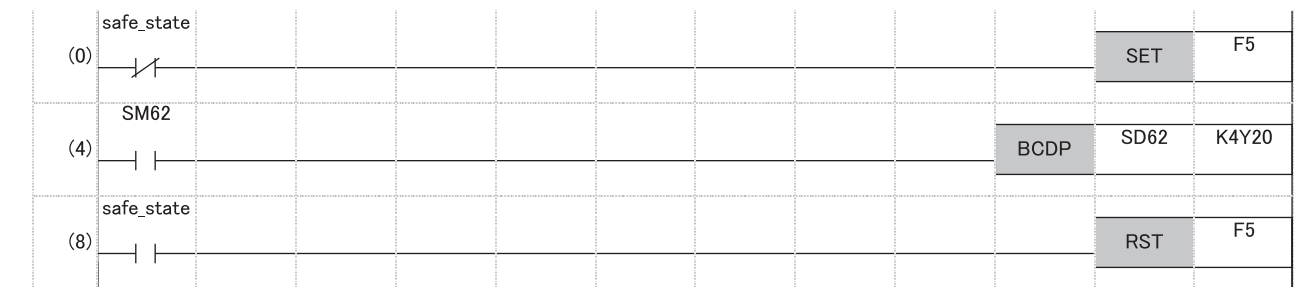
■To restart safety control by the command from the GOT



■To use the annunciator (F)

The safe state signal status can be controlled using the annunciator (F) in the standard program. The safe state signal status is passed from the safety program to the standard program via the standard/safety shared label (safe_state), and the status is controlled with the annunciator No.5. If an error is detected with the annunciator, the corresponding annunciator number is output to Y20.

- Standard program



- (0) When the safe state signal turns off, the annunciator No.5 turns on.
- (4) The annunciator number detected by SM62 (Annunciator) is output to Y20.
- (8) When the safe state signal turns on, the annunciator No.5 turns off.

Classes

The following table lists the availability of the classes of safety global labels and standard/safety shared labels.

○: Applicable, ×: Not applicable

Class	Availability	
	Safety global label	Standard/safety shared label
VAR_GLOBAL	○	○
VAR_GLOBAL_CONSTANT	○	○
VAR_GLOBAL_RETAIN	×	×

The following table lists the availability of the classes of safety local labels.

○: Applicable, ×: Not applicable

Class	Availability		
	Safety program	Safety function	Safety function block
VAR	○	○	○
VAR_CONSTANT	○	○	○
VAR_RETAIN	×	×	×
VAR_INPUT	×	○	○
VAR_OUTPUT	×	○	○
VAR_OUTPUT_RETAIN	×	×	×
VAR_IN_OUT	×	×	○
VAR_PUBLIC	×	×	○
VAR_PUBLIC_RETAIN	×	×	×

Data types

Primitive data type

The following table lists the availability of primitive data types.

○: Applicable, ×: Not applicable

Data type		Availability
Bit	BOOL	○
Word [unsigned]/bit string [16 bits]	WORD	○
Double word [unsigned]/bit string [32 bits]	DWORD	○
Word [signed]	INT	○
Double word [signed]	DINT	○
Single-precision real number	REAL	×
Double-precision real number	LREAL	×
Time	TIME	○
String	STRING	×
String [Unicode]	WSTRING	×
Timer	TIMER	○
Retentive timer	RETENTIVETIMER	○
Long timer	LTIMER	×
Long retentive timer	LRETENTIVETIMER	×
Counter	COUNTER	○
Long counter	LCOUNTER	×
Pointer	POINTER	×

Structures

The structure definition is shared by standard programs and safety programs. However, it cannot be used in the following cases.

- A member of the primitive data type which cannot be used in safety programs exists.
- An initial value is set in the structure definition.

5 LADDER DIAGRAM

RnCPU RnENCPU RnPCPU (Process) RnPCPU (Redundant) RnSFCPU (Standard) RnSFCPU (Safety)

Ladder diagram is a programming language used to describe sequence control. Each ladder consists of contacts and coils and represents logical operations consisting of AND/OR in combinations of series and parallel.

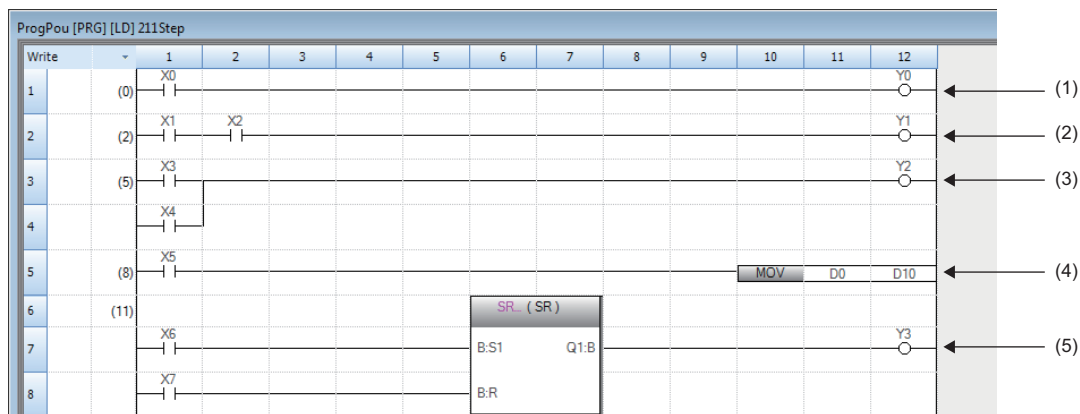
Point

This chapter describes the operation and specifications of the ladder diagram. For the operation method of the engineering tool for creating a ladder program, refer to the following.

GX Works3 Operating Manual

5.1 Configuration

The following are the programs written in ladder diagram.







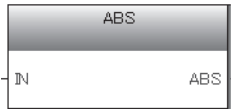
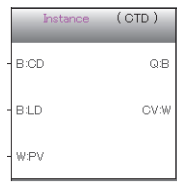


- (1) Ladder program consisting of a contact and a coil
- (2) Ladder program configured in series
- (3) Ladder program configured in parallel
- (4) Ladder program using an instruction
- (5) Ladder program using a standard function/function block

Ladder symbols

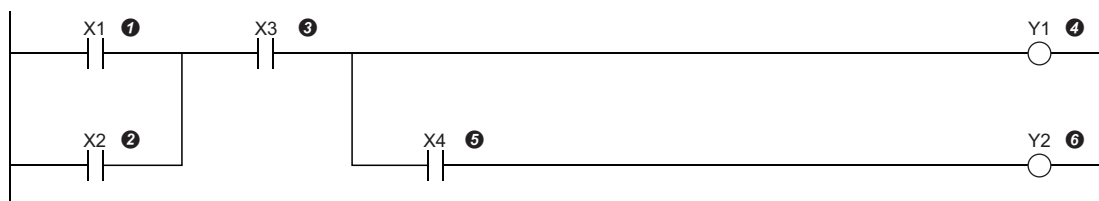
The following table lists the ladder symbols that can be used for programming in ladder diagram.

Item		Description
Normally open contact		Energized when the specified device or label is on.
Normally closed contact		Energized when the specified device or label is off.
Rising edge pulse		Energized on the rising edge (off to on) of the specified device or label.
Falling edge pulse		Energized on the falling edge (on to off) of the specified device or label.
Negated rising edge pulse		Energized when the specified device or label is off, on, or on the falling edge (on to off).
Negated falling edge pulse		Energized when the specified device or label is off, on, or on the rising edge (off to on).

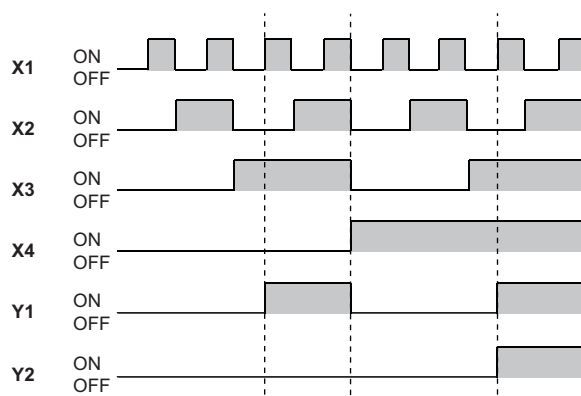
Item		Description
Operation result rising edge pulse conversion		Energized on the rising edge (off to on) of the operation result. De-energized while the operation result is not on the rising edge.
Operation result falling edge pulse conversion		Energized on the falling edge (on to off) of the operation result. De-energized while the operation result is not on the falling edge.
Operation result inversion		Inverts the previous operation results.
Coil		Outputs the operation result to the specified device or label.
Instruction		Executes the instruction specified in "[]".
Loopback		When the number of contacts that can be created on a single ladder line is exceeded, the ladder is looped back with loopback source and destination symbols created on it.
Function		Executes a function. <ul style="list-style-type: none"> • How to create functions (GX Works3 Operating Manual) • Standard functions (MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks))
Function block		Executes a function block. <ul style="list-style-type: none"> • How to create function blocks (GX Works3 Operating Manual) • Standard function blocks (MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)) • Module function blocks (Function Block Reference for the module used)

Program execution order

The program is executed in order of the following numbers.



When the above program is executed, Y1 and Y2 turn on while X1 to X4 turn on or off as shown below.



5

This function enables to create numerical operations and character string processing easily in ladder programs.

- [illegible]

- | | | | | | | |
|----|--|--|--|--|---|--|
| X0 | | | | | 1 | W0 := (D0+D1+D2+D3+D4+D5+D6+D7+D8+D9+D10+D11)/k12; |
|----|--|--|--|--|---|--|

- The inline ST cannot be used in safety programs.
- The inline ST cannot be used in the Zoom editor of SFC programs.

For the specifications of the inline ST, refer to the specifications of the ST language.

5 LADDER DIAGRAM

5.2 Inline ST

Precautions

- Only one inline ST can be created on a single line of a ladder program.
- Both a function block and inline ST box cannot be used on a single line of a ladder program.
- Creating an inline ST box at the position of an instruction corresponding to a contact creates an inline ST box at the position of an instruction corresponding to a coil.
- Up to 2048 characters can be input in the inline ST. (A newline is counted as two characters.)
- Using a RETURN statement in the inline ST causes the processing in the inline ST box to end, instead of the processing of the program block.
- No function block can be called from the inline ST.
- The inline ST uses the CJ instruction in conversion to control the operation of a program. When the contact of the inline ST is off, the processing in the inline ST is not executed by the CJ instruction. Thus, for example, the device that turned on by the assignment statement in the inline ST holds its output status even when the inline ST is not executed. For details on the CJ instruction, refer to the following.

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

5.3 Statements and Notes

Statements and notes can be used in ladder programs.

Statements

A statement is used to add a comment to a ladder block. Adding a comment makes it easy to understand the flow of processing.

There are three types of statements: line statement, P statement, and I statement.

The line statement can be displayed in the tree of the navigation window.

■Line statement

This type of statement adds a comment to the entire ladder block.

■P statement

This type of statement adds a comment to a pointer number.

■I statement

This type of statement adds a comment to an interrupt pointer number.

Notes

A note is used to add a comment to a coil and instruction in a program.

Adding a comment makes it easy to understand the contents of the coil and instruction.

Categories of statements and notes

Statements and notes are classified into two categories: "In PLC" and "In Peripheral".

Category	Type	Description
In PLC	<ul style="list-style-type: none">• Line statement• P statement• I statement• Note	Statements and notes can be stored in a CPU module. An In PLC Statement uses the following number of steps. (Assumed that only one-byte characters are entered. Values after the decimal point are rounded up.) • $2 + \text{Number of characters} \div 2$ (steps)
In Peripheral	<ul style="list-style-type: none">• Line statement• P statement• I statement• Note	Statements and notes cannot be stored in a CPU module. (Only position information is stored.) Statements and notes need to be stored in a peripheral. One line consumes one step. A text that has been entered is automatically preceded by an asterisk "***".

6 STRUCTURED TEXT LANGUAGE



ST language is defined by International Standard IEC61131-3 that defines the logic description system. ST language is a text language with a similar grammatical structure to C. This language is suitable for programming complicated processing that cannot be easily described by ladder diagram.

Point

This chapter describes the operation and specifications of the structured text language. For the operation method of the engineering tool for creating an ST program, refer to the following.

GX Works3 Operating Manual

The ST language supports control syntax, operational expressions, function blocks (FB), and functions (FUN), and can describe them as shown below.

Ex.

Control syntax such as selective branches by conditional statements and repetitions by iteration statements

(*Control conveyors, Line A to C.*)

```
CASE Line OF
  1:
    Start_switch := TRUE; (*The conveyor starts.*)
  2:
    Start_switch := FALSE; (*The conveyor stops.*)
  3:
    Start_switch := TRUE; (*The conveyor stops with an alarm.*)
ELSE
  Alarm_lamp := TRUE;
END_CASE;
IF Start_switch = TRUE THEN (*The conveyor starts and performs processing 100 times.*)
  FOR Count := 0 TO 100 BY 1 DO
    Count_No. := Count_No + 1;
  END_FOR;
END_IF;
```

Ex.

Expression using operators (*, /, +, -, <, >, =)

```
D0 := D1 * D2 + D3 / D4 - D5;
IF D0 > D10 THEN
  D0 := D10;
END_IF;
```

Ex.

Calling function blocks that have been defined

```
//FB data name: LINE1_FB
//Input variable: I_Test
//Output variable: O_Test
//Input/output variable: IO_Test
//FB label name: FB1
FB1(I_Test:= D0, O_Test => D1, IO_Test:= D100);
```

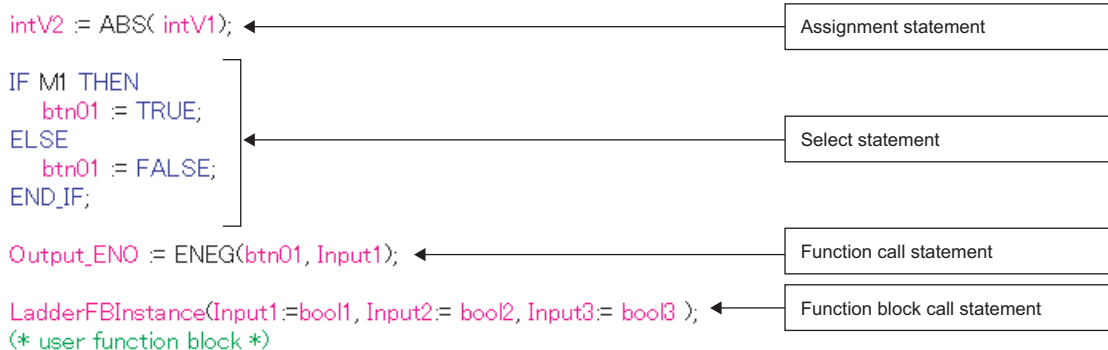
Ex.

Calling standard functions

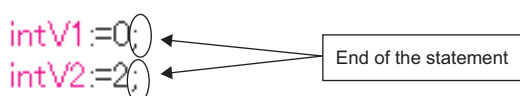
```
(*Convert BOOL data type to INT/DINT data type.*)
wLabel2 := BOOL_TO_INT (bLabel1);
```

6.1 Configuration

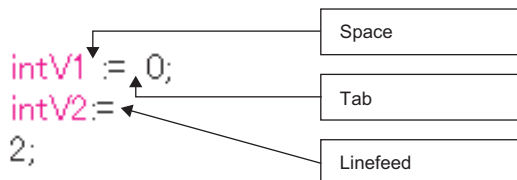
Programs written in ST language consist of operators and control statements.



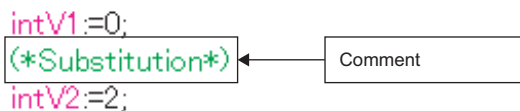
Each statement must end with a semicolon ";".



Spaces, tabs, and line feeds can be inserted between an operator and data.



Comments can be inserted into a program. Enclose a comment statement with "(*" and "*)".



Program components

An ST program consists of the following components.

Item	Example	Reference
Delimiter	;, (,)	Page 60 Delimiters
Operator	+, -, <, >, =	Page 60 Operators
Reserved word	Control statement	IF, CASE, WHILE, RETURN
	Device	X0, Y10, M100, ZR0
	Data type	BOOL, DWORD
	Standard function	ADD, REAL_TO_STRING_E
Constant	123, "abc"	Page 70 Constants
Label	Switch_A	Page 71 Labels and devices
Comment	(*Turn on.*)	Page 73 Comments
Other symbols	One-byte space, line feed code, TAB code	—

- Write delimiters, operators, and reserved words in one-byte characters.
- For details on the reserved words, refer to the following.

GX Works3 Operating Manual

Delimiters

ST language supports the following delimiters to make the program structure clear.

Symbol	Description
()	Parenthesized expression
[]	Specification of array element
. (period)	Specification of structure or function block members
, (comma)	Argument separation
: (colon)	Device specifier
; (semicolon)	Termination of statement
" (double quotation mark)	Notation of Unicode string
' (single quotation mark)	Notation of string (ASCII, Shift JIS)
.. (two periods)	Specification of integer range

Operators

The following table lists the operators, applicable data types, and operation result data types used in ST programs.

Operator	Applicable data type	Operation result type
*, /, +, -	ANY_NUM	ANY_NUM
<, >, <=, >=, =, <>	ANY_ELEMENTARY ^{*1}	Bit
MOD	ANY_INT	ANY_INT
AND, &, XOR, OR, NOT	ANY_BIT	ANY_BIT
**	ANY_REAL (base) ANY_NUM (exponent)	ANY_REAL

^{*1} WSTRING data type Unicode string cannot be specified.

The following table lists the operators in descending order of priority.

Operator	Description	Example	Priority
()	Parenthesized expression	(2+3)*(4+5)	1
Standard function ()	Argument of standard function	CONCAT('AB','CD')	2
**	Exponentiation	3.0**4	3
-	Sign inversion	-10	4
NOT	Bit type complement	NOT TRUE	5
*	Multiplication	10*20	
/	Division	20/10	
MOD	Remainder	17 MOD 10	
+	Addition	1.4+2.5	6
-	Subtraction	3-2	7
<, >, <=, >=	Comparison	10>20	
=	Equality	T#26h=T#1d2h	
<>	Inequality	8#15<>13	8
&, AND	AND operation	TRUE AND FALSE	9
XOR	XOR operation	TRUE XOR FALSE	10
OR	OR operation	TRUE OR FALSE	11

- If one expression includes multiple operators with the same priority, operation is performed in order from the leftmost operator.
- Up to 1024 operators can be used in a single expression.

Control statements

The following table lists the control statements that can be used in ST programs.

Item	Description	Reference
Assignment statement	Assignment statement	Page 61 Assignment statement
Subprogram control statement	Function block and function call statements	Page 63 Subprogram control statements
	RETURN statement	
Select statement	IF statement (IF THEN, IF ELSE, IF ELSIF)	Page 64 Select statements
	CASE statement	
Iteration statement	FOR statement	Page 65 Iteration statements
	WHILE statement	
	REPEAT statement	
	EXIT statement	

Write control statements in one-byte characters.

Assignment statement

Format	Description	Example
<Left side>:=<Right side>;	This statement assigns the result of the right side expression to the label or device on the left side. The data types of the result of the right side expression and the left side need to be the same.	intV1:=0; intV2:=2;

When using an array type label and structure label, note the data types of the left side and right side of the assignment statement.

For array type labels, the data type and the number of elements need to be the same between the left and right sides. When using array type labels, do not specify elements.

Ex.

intAry1:=intAry2;

For structure labels, the data type of the structure needs to be the same between the left and right sides.

Ex.

dutVar1:=dutVar2;

■ Automatic conversion of data type

When assignment or arithmetic operational expressions between different data types is described in ST language, one data type may be converted automatically.

Ex.

Example of automatic conversion

```
dintLabel1 := intLabel1;  
// Assignment statement: Automatically convert the INT type variable (intLabel1) to a DINT type variable, and assign it to the DINT type variable (dintLabel1).  
dintLabel1 := dintLabel2 + intLabel1;  
// Arithmetic operational expression: Automatically convert the INT type variable (intLabel1) to a DINT type variable, and perform addition in DINT type.  
DMOV(TRUE, wordLabel1, dwordLabel1);  
// Instruction, function, and function block call statement: Automatically convert the WORD type input argument (wordLabel1) to a DWORD type variable, and transfer the data.
```

Type conversion is performed in assignment statements, VAR_INPUT part where input argument is passed to function blocks and functions (including instruction, standard functions, and standard function blocks), and arithmetic operational expressions.

To prevent data from being lost during type conversion, conversion is performed only from a smaller size data type to a larger size data type. Type conversion is performed for the following data types among the primitive data types.

Data type	Description
Word [signed]	When the data type is converted to the double word [signed], the value is automatically sign extended. When the data type is converted to the single-precision real number or double-precision real number, the value is automatically converted to the same value as the integer before conversion.*1
Word [unsigned]/bit string [16 bits]	When the data type is converted to the double word [unsigned]/bit string [32 bits] or double word [signed], the value is automatically zero extended.*2 When the data type is converted to the single-precision real number or double-precision real number, the value is automatically converted to the same value as the integer before conversion.*1
Double word [signed]	When the data type is converted to the double-precision real number, the value is automatically converted to the same value as the integer before conversion.
Double word [unsigned]/bit string [32 bits]	
Single-precision real number	When the data type is converted to the double-precision real number, the value is automatically converted to the same value.

*1 When 16-bit data (word [signed] or word [unsigned]/bit string [16 bits]) is passed to the input argument of which data type is ANY_REAL, the data is automatically converted to single-precision real number.

*2 When data (word [unsigned]/bit string [16 bits]) is passed to the input argument of which data type is ANY32, the data is automatically converted to double word [unsigned]/bit string [32 bits].

For the data types other than the above, use the type conversion functions.

Use the type conversion functions for the following conversions as well.

- Type conversion between integral data types with different signs
- Type conversion between data types both of which lose data

For the precautions on assignment of the arithmetic operation result, refer to the following.

📖 Page 66 When the result of an arithmetic operation is assigned

For the precautions on the use of devices, refer to the following.

📖 Page 72 When performing automatic conversion of data type with devices

Subprogram control statements

■Function block call statement

Format	Description
Instance name (input variable1:=variable1,...output variable1=>variable2,...);	Enclose the assignment statements to the input and output variables in '()' after the instance name. When using multiple variables, delimit individual assignment statements with a comma ','.
Instance name.input variable1:=variable1; : Instance name(); Variable2:=instance name.output variable1;	List the assignment statements to input and output arguments between function block call statements.

The following table lists the symbols used in the arguments of the function block call statements and the expressions that can be assigned.

Type	Description	Attribute	Symbol used	Assignable expression
VAR_INPUT	Input variable	None, or RETAIN	:=	All expressions
VAR_OUTPUT	Output variable	None, or RETAIN	=>	Only variables
VAR_IN_OUT	Input/output variable	None	:=	All expressions
VAR_PUBLIC	External variable	None, or RETAIN	Cannot be specified.	—

The execution result of a function block is stored by specifying an output variable after the instance name with a period "." and assigning the result to the specified variable.

Function block	FB definition	Example
Having one input variable and one output variable	FB name: FBADD FB instance name: FBADD1 Input variable1: IN1 Output variable1: OUT1	FBADD1(IN1:= Input1); Output1 := FBADD1.OUT1;
Having three input variables and two output variables	FB name: FBADD FB instance name: FBADD1 Input variable1: IN1 Input variable2: IN2 Input variable3: IN3 Output variable1: OUT1 Output variable2: OUT2	FBADD1(IN1:= Input1 ,IN2:= Input2 ,IN3:= Input3); Output1 := FBADD1.OUT1; Output2 := FBADD1.OUT2;

■Function call statement

Format	Description
Function name (variable1,variable2,...);	Enclose arguments in "()" following the function name. When using multiple variables, delimit them with a comma ",".

The execution result of a function is stored by assigning it to a variable.

Function	Example
Having one input variable (example: ABS)	Output1 := ABS(Input1);
Having three input variables (example: MAX)	Output1 := MAX(Input1 , Input2 , Input3);
Having EN/ENO (except for standard functions) (example: MAX_E)	Output1 := MAX_E(boolEN , boolENO , Input1 , Input2 , Input3);
Standard function (example: MOV)	boolENO := MOV(boolEN , Input1 , Output1); (The execution result of the function is ENO, and the first argument (variable1) is EN.)

A user-defined function which returns no value or a function including VAR_OUTPUT in parameters of the call statement can be executed as a statement by describing it after ";" (semicolon).

■RETURN statement

Control statement	Format	Description	Example
■RETURN	RETURN;	<p>The statement is used to terminate a program, function block, or function during operation.</p> <p>When the RETURN statement is used in a program, control jumps to the step next to the last statement in the program.</p> <p>When the RETURN statement is used in a function block, control returns from the function block.</p> <p>When the RETURN statement is used in a function, control returns from the function.</p> <p>A single RETURN statement uses one point of the pointer type label in the system.</p>	<pre>IF bool1 THEN RETURN; END_IF;</pre>

Select statements

Control statement	Format	Description	Example
■IF THEN	<pre>IF<boolean expression>THEN <statement 1...>; END_IF;</pre>	<p>The statement is executed if the boolean expression (conditional formula) is TRUE. The statement is not executed if the boolean expression is FALSE.</p> <p>Any expression can be used for the boolean expression if it returns TRUE or FALSE as the boolean operation result of a single bit variable expression or a complicated expression including many variables.</p>	<pre>IF bool1 THEN intV1 := intV1 + 1; END_IF;</pre>
■IF...ELSE	<pre>IF<boolean expression>THEN <statement 1...>; ELSE <statement 2...>; END_IF;</pre>	<p>Statement 1 is executed if the boolean expression (conditional formula) is TRUE.</p> <p>If the boolean expression is FALSE, statement 2 is executed.</p>	<pre>IF bool1 THEN intV3 := intV3 + 1; ELSE intV4 := intV4 + 1; END_IF;</pre>
■IF...ELSEIF	<pre>IF<boolean expression 1>THEN <statement 1...>; ELSEIF<boolean expression 2>THEN <statement 2...>; ELSEIF<boolean expression 3>THEN <statement 3...>; END_IF;</pre>	<p>Statement 1 is executed if boolean expression 1 (conditional formula) is TRUE. If boolean expression 1 is FALSE and boolean expression 2 is TRUE, statement 2 is executed.</p> <p>If boolean expressions 1 and 2 are FALSE and boolean expression 3 is TRUE, statement 3 is executed.</p>	<pre>IF bool1 THEN intV1 := intV1 + 1; ELSIF bool2 THEN intV2 := intV2 + 2; ELSIF bool3 THEN intV3 := intV3 + 3; END_IF;</pre>
■CASE	<pre>CASE<integer expression>OF <selected integer 1>: <statement 1...>; <selected integer 2>: <statement 2...>; : <selected integer n>: <statement n...>; ELSE <statement n+1...>; END_CASE;</pre>	<p>The statement that has a selected integer value matching the value of the integer expression (conditional formula) is executed, and if no statement has a matching value, the statement following the ELSE statement is executed.</p> <p>The CASE statement can be used to execute a select statement according to a single integer value or the integer value of the result of a complicated expression.</p>	<pre>CASE intV1 OF 1: bool1 := TRUE; 2: bool2 := TRUE; ELSE intV1 := intV1 + 1; END_CASE;</pre>

Iteration statements

Control statement	Format	Description	Example
■FOR...DO	FOR<iteration variable initialization> TO<final value> BY<increase expression>DO <statement...>; END_FOR;	Data to be used as an iteration variable is initialized. One or more statements between the DO statement and the END_FOR statement are executed repeatedly, adding or subtracting the initialized iteration variable according to the increase expression until the final value is exceeded. The iteration variable after the FOR...DO statement is completed retains the value at the end of the processing.	FOR intV1 := 0 TO 30 BY 1 DO intV3 := intV1 + 1; END_FOR;
■WHILE...DO	WHILE<boolean expression>DO <statement...>; END_WHILE;	One or more statements are executed while the boolean expression (conditional formula) is TRUE. The boolean expression is determined before execution of the statement and, if it is FALSE, any statement in the DO...END_WHILE statement is not executed. Since <boolean expression> in the WHILE statement is only necessary to return TRUE or FALSE as the result, any expression that can be specified in <boolean expression> of the IF statement can be used.	WHILE intV1 = 30 DO intV1 := intV1 + 1; END_WHILE;
■REPEAT...UNTIL	REPEAT <statement...>; UNTIL<boolean expression> END_REPEAT;	One or more statements are executed while the boolean expression (conditional formula) is FALSE. The boolean expression is determined after execution of the statement and, if the value is TRUE, any statement in the REPEAT...UNTIL statement is not executed. Since <boolean expression> in the REPEAT statement is only necessary to return TRUE or FALSE as the result, any expression that can be specified in <boolean expression> of the IF statement can be used.	REPEAT intV1 := intV1 + 1; UNTIL intV1 = 30 END_REPEAT;
■EXIT	EXIT;	This statement can be used only within an iteration statement to terminate the iteration statement in the middle of processing. When the EXIT statement is reached during execution of the iteration loop, the iteration loop processing after the EXIT statement is not executed. The program execution continues from the line next to the one where the iteration statement was terminated.	FOR intV1 := 0 TO 10 BY 1 DO IF intV1 > 10 THEN EXIT; END_IF; END_FOR;

Precautions

■When an assignment statement is used

- Up to 255 characters can be assigned to a string. If characters are assigned exceeding the valid range, a conversion error occurs.
- Timer type and counter type contacts and coils cannot be used at the left side of an assignment statement.
- Instances of function blocks cannot be used at the left side of an assignment statement. Use the input variable, output variable, or external variable of an instance.

■When the step relay (S) or SFC block device (BL) is used

If the step relay (S) or SFC block device (BL) is used at the left side of an assignment statement or as an input argument of a function or function block, a conversion error may occur. If an error occurs, change the assignment statement.

Ex.

The following is the example of rewriting.

Before change	After change
M0 := S0;	IF S0 THEN M0 := TRUE; ELSE M0 := FALSE; END_IF;

In addition, to use the digit-specified step relay (S) or the step relay with block specification (BL□\S□), the data size must be specified correctly. Since the step relay (S) and the step relay with block specification (BL□\S□) are not targeted for automatic conversion of data type, a conversion error may occur if the data size is not the same.

Ex.

The following is the example of rewriting.

Before change	After change
(*Conversion error because K4S0 is 16 bits and D0:UD is 32 bits*) D0:UD := K4S0; (*Conversion error because BL1\K4S10 is 16 bits and the second argument of DMOV is 32 bits*) DMOV(TRUE, BL1\K4S10, D100);	(*Assign data to the 16-bit device.*) D0 := K4S0; (*Specify 32-bit data for DMOV.*) DMOV(TRUE, BL1\K8S10, D100:UD);

■When the result of an arithmetic operation is assigned

When assigning the result of an arithmetic operation to a data type variable with larger data size, convert the variable of the arithmetic operational expression to the data type of the left side in advance.

Ex.

To assign the arithmetic operation result with a data size of 16 bits (INT type) to the 32-bit data type (DINT type):
varDint1 := varInt1 * 10; //The varInt1 is an INT type variable, and the varDint1 is a DINT type variable.

The result of the arithmetic operation will be the same data type as the input operand. For this reason, if the operation result of varInt1*10 in the above program exceeds the INT type range (-32768 to 32767), the operation result of overflow or underflow is assigned to varDint1.

In this case, convert the operand of the operational expression to the data type of the left side in advance.

```
varDint2 := INT_TO_DINT( varInt1 ); //An INT type variable is converted to a DINT type variable.  
varDint1 := varDint2 * 10; //Multiplication is performed in DINT type, and the operation result is assigned.
```

■When a sign inversion operator is used in the arithmetic operational expression

If a sign inversion operator is used for the minimum value of each data type, the value remains unchanged.

For example, the minimum value of INT type data will be $-(-32768) = -32768$.

If a sign inversion operator is used for variables targeted for automatic conversion of data type, an intended operation result may not be obtained.

Ex.

When the value of varInt1 (INT type) is -32768 and the value of varDint1 (DINT type) is 0

```
varDint2 := -varInt1 + varDint1;
```

In this program, the value of $(-varInt1)$ remains unchanged. Therefore, the value, -32768, is assigned to varDint2.

To use a sign inversion operator in an arithmetic operational expression, automatically convert data type before the arithmetic operation or do not use a sign inverted operator in the program.

Ex.

When data type is automatically converted before the arithmetic operation

```
varDint3 := varInt;
```

```
varDint2 := -varDint3 + varDint1;
```

Ex.

When a sign inversion operator is not used

```
varDint2 := varDint1 - varInt1
```

■When the data type is converted from single-precision real number to double-precision real number

When the type conversion function, REAL_TO_LREAL, is executed, an error may occur in the conversion result.

Consequently, when the data type is automatically converted or when a function with a return value of real number type (such as SIN function) is used as the right side of an assignment statement or an operand of arithmetic operational expression, an intended operation result may not be obtained.

Ex.

An error occurs.

```
varReal1 := -1234.567;
```

```
varLReal1 := ABS(varReal1);
```

In the above program, the data type of the return value of $ABS(varReal1)$ is single-precision real number. Since the return value is converted to a double-precision real number and assigned to varReal1, an error occurs.

Create a program using a function with the data type same as the assignment target.

Ex.

No error occurs.

```
varLReal2 := -1234.567;
```

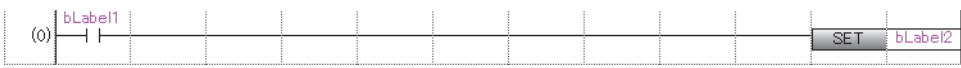
```
varLReal1 := ABS(varLReal2);
```

■When a bit type label is used

Once the boolean expression (conditional formula) is established in a select or iteration statement and if a bit type label is set to on in <statement>, the state of the label will be always on.

Ex.

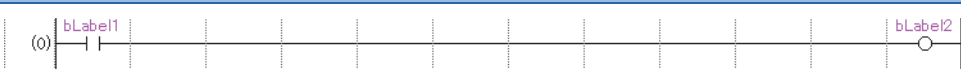
Program that keeps the label status on

ST program	Ladder program performing the processing equivalent to ST program
<pre>IF bLabel1 THEN bLabel2 := TRUE; END_IF;</pre>	

To prevent the label from being always on, add a program that turns off the bit type label as shown below.

Ex.

Program that avoids the label from being always on

ST program ^{*1}	Ladder program performing the processing equivalent to ST program
<pre>IF bLabel1 THEN bLabel2 := TRUE; ELSE bLabel2 := FALSE; END_IF;</pre>	

^{*1} The above program can also be described as follows.

```
bLabel2:=bLabel1;
or
OUT(bLabel1,bLabel2);
```

Note however that if the OUT instruction is used in the <statement>, the program will be in the same state as a program that keeps the label status on.

■When a timer function block or counter function block is used

In a select statement, a boolean expression (conditional formula) differs from the execution conditions of timer function blocks or counter function blocks.

Ex.

Timer function block

Program example before change

```
IF bLabel1 THEN
  TIMER_100_FB_M_1 (Coil := bLabel2, Preset := wLabel3, ValueIn := wLabel4, ValueOut => wLabel5, Status => bLabel6);
END_IF;
(*When bLabel1 is on and bLabel2 is also on, counting starts.*)
(*When bLabel1 is on and bLabel2 is off, the counted value is cleared.*)
(*When bLabel1 is off and bLabel2 is on, counting stops. The counted value is not cleared.*)
(*When bLabel1 is off and bLabel2 is also off, counting stops. The counted value is not cleared.*)
```

Program example after change

```
TIMER_100_FB_M_1 (Coil := (bLabel1 & bLabel2), Preset := wLabel3, ValueIn := wLabel4, ValueOut => wLabel5, Status => bLabel6);
```

Ex.

Counter function block

Program example before change

```
IF bLabel1 THEN
  COUNTER_FB_M_1 (Coil := bLabel2, Preset := wLabel3, ValueIn := wLabel4, ValueOut => wLabel5, Status => bLabel6);
END_IF;
(*When bLabel1 is on and bLabel2 is on/off, the value is incremented by one.*)
(*When bLabel1 is off and bLabel2 is on/off, the value is not counted.*)
(*The counting operation does not depend on the on/off status of bLabel1.*)
```

Program example after change

```
COUNTER_FB_M_1 (Coil := (bLabel1 & bLabel2), Preset := wLabel3, ValueIn := wLabel4, ValueOut => wLabel5, Status => bLabel6);
```

The above examples of programs before change cause problems because the statement related to the timer or counter is not executed unless the select statement is established.

To operate the timer or counter on the basis of the bLabel1 condition and bLabel1 AND condition, do not use control statements but use only function blocks.

The timer and counter can be operated by using the programs after change.

■When a FOR...DO statement is used

- A structure member or array element cannot be used for the iteration variable.
- Match the type used in the iteration variable with the types of <final value expression> and <increase expression>.
- <Increase expression> can be omitted. If omitted, <increase expression> is assumed to be 1.
- If 0 is assigned to <increase expression>, the portions after the FOR statement may no longer be executed or an infinite loop may occur.
- In the FOR...DO statement, the iteration variable count processing is performed after execution of <statement--> in the FOR statement. If the count processing is executed in such a way that it exceeds the maximum value of the iteration variable data type or is below the minimum value, an infinite loop occurs.

■When instructions executed at the rising edge or falling edge are used

- The following table lists the operations of when instructions executed at the rising edge or falling edge used in the IF or CASE statement.

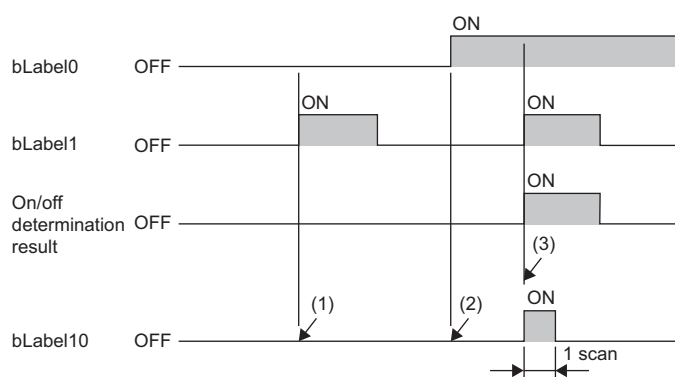
Condition			Operation result		
Conditional formula of IF or CASE statement	Instruction execution condition (EN)	On/off determination result of the instruction in the last scan	On/off determination result of the instruction	Instruction at the rising edge	Instruction at the falling edge
TRUE or CASE match	TRUE	On	On	Not executed	Not executed
		Off	On	Executed	Not executed
	FALSE	On	Off	Not executed	Executed
		Off	Off	Not executed	Not executed
TRUE or CASE mismatch	TRUE	On	Off	Not executed	Not executed ^{*1}
		Off	Off	Not executed	Not executed
	FALSE	On	Off	Not executed	Not executed ^{*1}
		Off	Off	Not executed	Not executed

*1 On the falling edge (on to off), the instruction is not executed because the condition of the IF or CASE statement is not satisfied.

Ex.

When the PLS instruction (execution condition: rising edge) is used in the IF statement

```
IF bLabel0 THEN
  PLS(bLabel1, bLabel10);
END_IF;
```



- (1) When bLabel0 is off (the conditional formula of the IF statement is FALSE), the on/off determination result will be off. The PLS instruction is not executed. (The bLabel10 remains off.)
- (2) When bLabel0 is on (the conditional formula of the IF statement is TRUE) and bLabel1 is off (the instruction execution condition is off), the on/off determination result will be off. The PLS instruction is not executed. (The bLabel10 remains off.)
- (3) When bLabel0 is on (the conditional formula of the IF statement is TRUE) and bLabel1 is also on (the instruction execution condition is on), the on/off determination result will be off to on (rising edge). The PLS instruction is executed. (The bLabel10 is on for one scan.)

- To execute instructions at the rising edge or falling edge in an iteration statement (FOR, WHILE, or REPEAT statement), use the edge relay (V) or perform index modification. In this case, one point of the edge relay (V) is used for each instruction that uses the edge relay (V) in the system. For this reason, secure the edge relay (V) for the number of instructions used in addition to the number of points used in the iteration statement.

Ex.

When instructions executed at the rising edge or falling edge are used in the FOR statement

Example of using the edge relay (V) in one location

(The edge relay (V) is used up to a total of 11 points (V0 to V10 for the INC instruction).)

```
FOR Z0 := 0 TO 9 BY 1 DO
  INC(EGP(M100Z0 , V0Z0) , D100Z0);
END_FOR;
```

Example of using the edge relay (V) in two locations

(The edge relay (V) is used up to a total of 22 points (V0 to V10 for the INC instruction, V11 to V21 for the DEC instruction).)

```
FOR Z0 := 0 TO 9 BY 1 DO
  INC(EGP(M100Z0 , V0Z0) , D100Z0);
  DEC(EGF(M200Z0 , V11Z0) , D200Z0);
END_FOR;
```

■When the master control instruction is used

Operations when the master control is off will be as follows.

- A statement in the select statement (IF or CASE statement) or iteration statement (FOR, WHILE, or REPEAT statement) performs no processing.
- For a statement outside the select statement or iteration statement, no processing is performed if it is an assignment statement, and a statement itself is not executed if it is other than assignment statement.

Ex.

Statement in the select statement (IF statement)

MC(M0 , N1 , M1); //Master control is off.

IF M2 THEN

M3 := M4; //M3 retains the value in the last scan because no processing is performed when the master control is off.

END_IF;

M20 := MCR(M0, N1);

Ex.

Statement (bit assignment statement) outside the select statement or iteration statement

MC(M0 , N1 , M1); //Master control is off.

M3 := M4; //M3 retains the value in the last scan because no processing is performed when the master control is off.

M20 := MCR(M0, N1);

Ex.

Statement (OUT instruction) outside the select statement or iteration statement

MC(M0 , N1 , M1); //Master control is off.

OUT(M2, M3); //M3 turns off because the instruction is not executed when the master control is off.

M20 := MCR(M0, N1);

Constants

Notation of constants

The following table lists the notations of strings in ST programs.

Data type		Notation	Example
String	STRING	Enclose a string (ASCII, Shift JIS) in single quotation marks (' ').	Stest:='ABC';
String [Unicode]	WSTRING	Enclose a Unicode string in double quotation marks (" ").	Stest:="ABC";

For the notations of constants other than the above, refer to the following.


 Page 47 Constants

Labels and devices


Specification method

Labels and devices can be directly described on ST programs. Labels and devices can be used for the left or right side of an expression or as an argument or return value of a standard function/function block.

For the applicable labels, refer to the following.

 Page 37 LABELS

For the applicable devices, refer to the following.

 MELSEC iQ-R CPU Module User's Manual (Application)

■Notation of devices with a type specifier

A word device can be used as any data type by adding a device type specifier to its name.

Device specifier	Data type	Example	Description of example
None	Generic data type ANY16. Word [signed], when only devices are used in such as an arithmetic operational expression. However, when it is specified as a device without a type specifier in a FUN/FB argument, it becomes a argument-defined data type.	D0	D0 without a type specifier
:U	Word [unsigned]/bit string [16 bits]	D0:U	D0 specified as the word [unsigned]/bit string [16 bits]
:D	Double word [signed]	D0:D	D0 and D1 specified as the double word [signed]
:UD	Double word [unsigned]/bit string [32 bits]	D0:UD	D0 and D1 specified as the double word [unsigned]/bit string [32 bits]
:E	Single-precision real number	D0:E	D0 and D1 specified as the single-precision real number
:ED	Double-precision real number	D0:ED	D0, D1, D2, and D3 specified as the double-precision real number

The following devices can use a device type specifier.

- Data register (D)
- Link register (W)
- Link direct device (J□\W□)
- Module access device (U□\G□)
- File register (R/ZR)
- Refresh data register (RD)


A device type specifier cannot be added to a digit-specified or index-modified device.

■Device specification method

Devices can be specified in the following methods.

- Index modification
- Bit specification
- Digit specification
- Indirect specification

For details, refer to the following.

 MELSEC iQ-R CPU Module User's Manual (Application)

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

Precautions

- The pointer type cannot be used in ST programs.
- When the timer, counter, or retentive timer device is used as current value, the data type will be the word [unsigned]/bit string [16 bits]. When the long timer, long counter, or long retentive timer device is used as current value, the data type will be the double word [unsigned]/bit string [32 bits].
- When performing assignment using digit specification, match the data type between the left and right sides.

Ex.

```
D0:=K5X0;
```

In this example, a program error occurs because K5X0 is the double word type and D0 is the word type.

- If the right side is greater than the left side when performing assignment using digit specification, data is transferred to within the range of the number of applicable points of the left side.

Ex.

```
K5X0:=2#1011_1101_1111_0111_0011_0001;
```

In this example, K5X0 has 20 applicable points and therefore 1101_1111_0111_0011_0001 (20 digits) is assigned to K5X0.

- When using the current value (such as TNn) of the counter (C), timer (T), or retentive timer (ST) as a type other than the word [unsigned]/bit string [16 bits], or when using the current value (such as LTNn) of the long counter (LC), long timer (LT), or long retentive timer (LST) as a type other than the double word [unsigned]/bit string [32 bits], use the type conversion functions.

Ex.

```
varInt:=WORD_TO_INT(T0);(*A type conversion function is used.*)
```

- If a coil (TC, STC, LTC, LSTC, CC, or LCC) of timer or counter devices is used at the right side of an assignment statement or as an input argument of a function or function block, it operates as a contact (TS, STS, LTS, LSTS, CS, or LCS).
- To use a coil of the timer or counter as an input argument, use a timer type label or counter type label.

Ex.

Timer device and timer type label


```
M1 := TC0; (*Assign a value of the contact (TS0) to M1.*)
```

```
M2 := INV(TC1); (*Assign the inversion result of the contact (TS1) to M2.*)
```

```
M1 := tLabel0.C; (*Assign a value of the coil of the timer type label, tLabel0, to M1.*)
```

```
M2 := INV(tLabel1.C); (*Assign the inversion result of the coil of the timer type label, tLabel0, to M1.*)
```

■When performing automatic conversion of data type with devices

Add a device type specifier when using a word device in a data type other than word [signed]. ( Page 71 Notation of devices with a type specifier)

Ex.

When transferring the values of D2 and D3 to dwordLabel1, a double word [unsigned] label

```
//Example of when adding a device type specifier and transferring the values in a correct data type
```

```
dwordLabel1 := D2:UD;
```

```
//Since the data type of D2:UD (D2 with a device type specifier) is double word [unsigned], the values of D2 and D3 are transferred to dwordLabel1.
```

```
//Example of an unintended transfer result
```

```
dwordLabel1 := D2;
```

```
//Since the data type of D2 without a device type specifier is word [signed], the data type is automatically converted into double word [unsigned] and data is transferred to dwordLabel1.
```

```
//Therefore, the value of D3 is not transferred but only the value of D2 is transferred.
```

Comments

The following table lists the comments that can be used in ST programs.

Type	Symbol	Description	Example
Single-line comment	//	The portion from the start symbol "//" to the end of the line is regarded as a comment.	//comment
Multiple-line comment	(**)	The portion from the start symbol "(" to the end symbol ")" is regarded as a comment. A line feed can be inserted in a comment.	■No line feed (*comment*) ■With a line feed (*line-1 comment line-2 comment*)
	/**/	The portion from the start symbol "/" to the end symbol "/" is regarded as a comment. A line feed can be inserted in a comment.	■No line feed /*comment*/ ■With a line feed /*line-1 comment line-2 comment*/

Do not write a comment containing an end symbol in a multiple-line comment.

7 FBD/LD



FBD/LD (function block diagram/ladder diagram) is a graphic language which describes programs by connecting blocks that perform predefined processing, variable elements, and constant elements along the flow of data and signals.

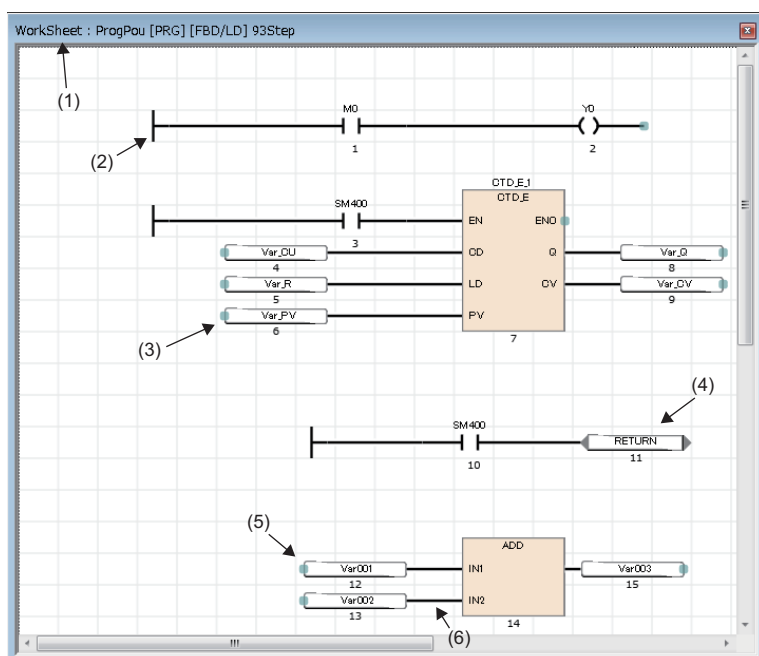
Point

This chapter describes the operation and specifications of the FBD/LD. For the operation method of the engineering tool for creating an FBD/LD program, refer to the following.

GX Works3 Operating Manual

7.1 Configuration

The following are the programs written in FBD/LD.



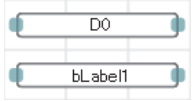
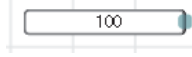
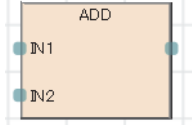
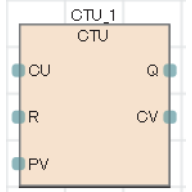
- (1) Worksheet
- (2) LD element
- (3) FBD element
- (4) Common element
- (5) Connection point
- (6) Connection line

In FBD/LD programs, data flows from the output point of a function block, function, variable (label or device), or constant to the input point of another function block or variable.

Program elements

FBD elements

The following table lists FBD elements consisting an FBD/LD program.

Item		Description
Variable		Stores a value. A specific data type is assigned to each variable, and only data of the assigned data type are stored. Labels and devices can be specified as a variable.
Constant		Outputs a specified value.
Function (FUN)		Executes a function. <ul style="list-style-type: none">• How to create functions (GX Works3 Operating Manual)• Standard functions (MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks))
Function block (FB)		Executes a function block. <ul style="list-style-type: none">• How to create function blocks (GX Works3 Operating Manual)• Standard function blocks (MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks))• Module function blocks (Function Block Reference for the module used)

7

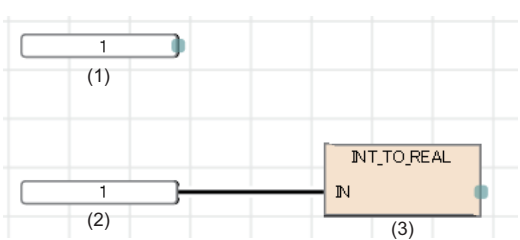
Data type of constant elements

The data type of the value input to a constant is not determined when the value is input. It is determined when the constant element is connected to another FBD element with a connection line. The data type will be the same as that of the connected FBD element.

Ex.

When the constant value is 1

The possible data types are BOOL, WORD, DWORD, INT, DINT, REAL, and LREAL, but the data type of the input value cannot be determined at this point. The data type is determined when the constant is connected to another FBD element.



- (1) The data type has not been determined.
- (2) INT data type
- (3) INT data type

Automatic conversion of data type

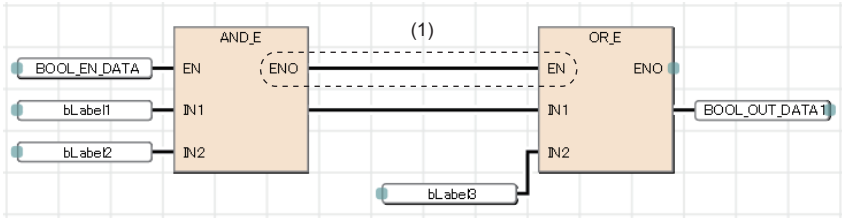
If the data types differ between connected elements, one data type may be converted automatically.

To prevent data from being lost during type conversion, conversion is performed only from a smaller size data type to a larger size data type. The data type automatic conversion processing in FBD/LD is same as that in ST language. For details, refer to the following.

Page 62 Automatic conversion of data type

■Input/output points of a function

- Connect all input points of a function with another FBD element.
- The data type is assigned to each input variable and output variable of a function. Match the data type of an element connected to the input point or output point with that of the input variable or output variable.
- Connect the output variable (except ENO) of an CPU module instruction or module dedicated instruction to the input variable of a function (or function block) via a variable element.
- When connecting two functions, always connect a function with EN to a function with EN so that the the functions do not use an undefined value. (Do not connect a function with EN to a function without EN.) In this case, connect ENO of the first function to EN of the second function.



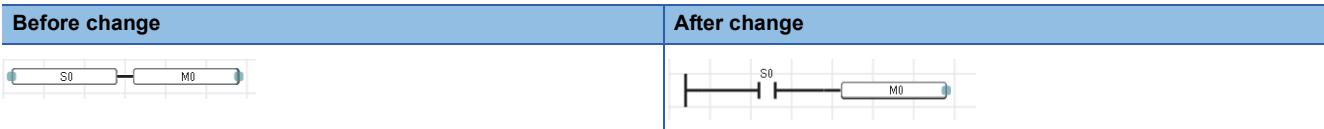
(1) Connect ENO to EN.

■When the step relay (S) or SFC block device (BL) is used

If the step relay (S) or SFC block device (BL) is used as a variable element, a conversion error may occur. If an error occurs, change the variable element to a contact element.

Ex.

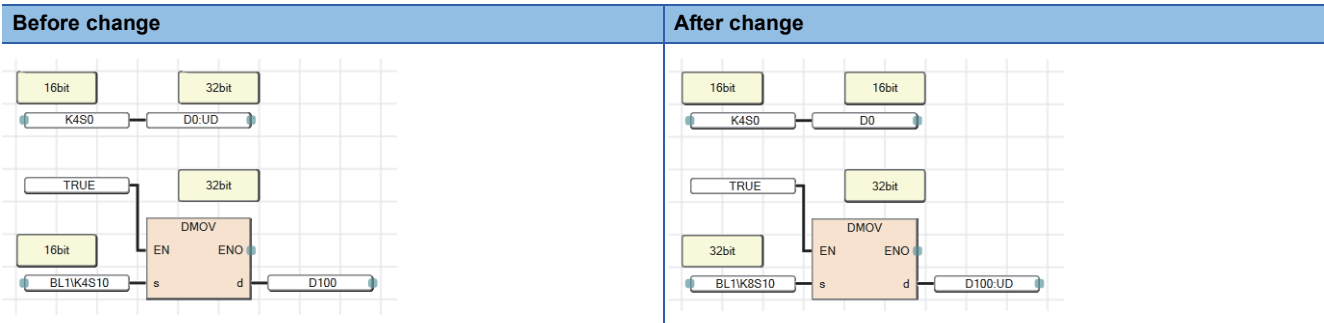
The following is the example of rewriting.



In addition, to use the digit-specified step relay (S) or the step relay with block specification (BL□\S□), the data size must be specified correctly. Since the step relay (S) and the step relay with block specification (BL□\S□) are not targeted for auto data type conversion, a conversion error may occur if the data size are not the same.












Ex.

The following is the example of rewriting.



LD elements

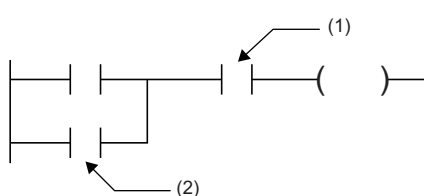
The following table lists the LD elements that can be used in FBD/LD programs.

Item		Description
Left rail		A start point of ladder program. The output of the left rail is always on.
Normally open contact		Energized when the specified device or label is on.
Normally closed contact		Energized when the specified device or label is off.
Rising edge pulse		Energized on the rising edge (off to on) of the specified device or label.
Falling edge pulse		Energized on the falling edge (on to off) of the specified device or label.
Negated rising edge pulse		Energized when the specified device or label is off, on, or on the falling edge (on to off).
Negated falling edge pulse		Energized when the specified device or label is off, on, or on the rising edge (off to on).
Coil		Outputs the operation result to the specified device or label.
Inverse coil		When the operation result turns off, the specified device or label turns on.
Set coil		When the operation result turns on, the specified device or label turns on. The device or label keeps on state even after the operation result turns off.
Reset coil		When the operation result turns on, the specified device or label turns off. If the operation result is off, the status of the device or label does not change.

■AND operation and OR operation of contact elements

At each contact, an AND operation or OR operation is performed depending on the connection status, and the operation result is output.

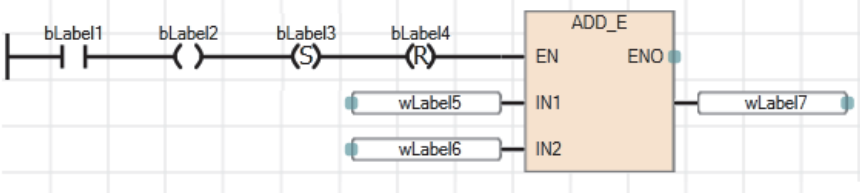
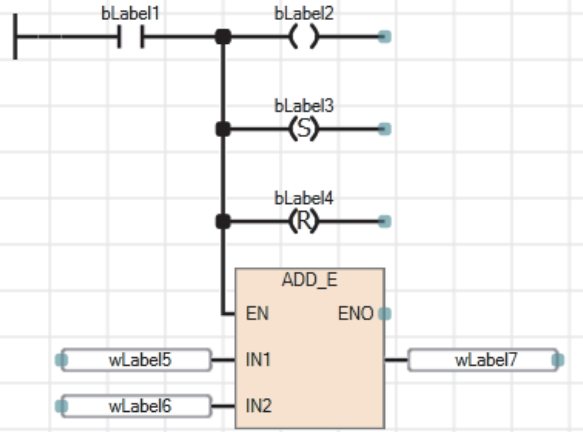
- Series connection (1): An AND operation is performed with the previous operation result.
- Parallel connection (2): An OR operation is performed with the previous operation result.



- (1) Contact connected in series
(2) Contact connected in parallel






■When another program element is connected to a connection point of coil for output

When another program element is connected to a connection point of coil for output in series, the operation is the same as that of the parallel connection.

Item	Description
Program example	
Operation at the above example	

Common elements

The following table lists the common elements that can be used in FBD/LD programs.

Item		Description
Jump* ¹		Jumps the execution processing to a jump element. Processing between this element and a jump label is not performed. Whether to perform jump processing or not is controlled by inputting on/off information to the element. On: Jump processing performed Off: Jump processing not performed
Jump label* ¹		A jump destination of a jump element in the same program. When the jump processing is performed, the program continues from the processing located after the jump label.
Connector		Used as a substitute for a connection line. Processing moves to a connector element used as a pair. One or more input connectors can be used as a pair of one output connector.
Return* ¹		Stops the execution processing after this element. Use the element not to execute the program, functions, and function blocks after the element. Whether to perform return processing or not is controlled by inputting on/off information to the element. On: Return processing performed Off: Return processing not performed
Comment		Used to write a comment.

*¹ These elements cannot be used in the Zoom editor of SFC programs.

Jump element

- If the timer whose coil is on is jumped, the time cannot be measured correctly.
- A jump label can be located before the jump element in the program sequence. If located, create a program so that the watchdog timer setting value is not exceeded. (How to exit from a loop must be considered.)
- Specify a pointer type local label for a jump element and a jump label. The pointer devices cannot be used.
- The CJ, SCJ, and JMP instructions (for pointer branch) cannot be used. Use a jump element when jump processing is required.
- Processing cannot be jumped to/from the outside of the program block.

Jump related operation	Availability
Jumping to the outside of the program block* ¹	Not supported
Jumping from the outside of the program block* ¹	Not supported
Calling a subroutine program	Supported
Being called as a subroutine program	Not supported

*¹ Branching by using the BREAK instruction is included.

Return element

- The operation of a return element differs depending on the POU (program, function, or function block) used.

POU used	Description
Program	Execution of the program is stopped.
Function	Execution of the function is stopped, and processing returns to the step next to the instruction that called the function.
Function block	Execution of the function block is stopped, and processing returns to the step next to the instruction that called the function block.

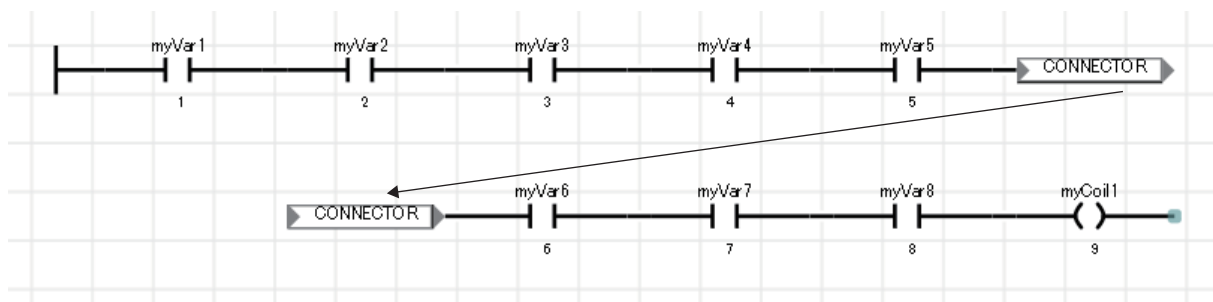
- If a return element is used in a macro type function block, do not allocate multiple function block elements with the same FB instance name.
- When a program where a return element is used is converted, a local label is automatically registered. The following restrictions apply to those labels.

Operation to the label registered automatically	Availability
Changing a label name	Not available ^{*1}
Changing a data type	Not available
Changing a class	Not available
Deleting a label	Not available ^{*1}
Changing a registration line	Available

^{*1} If the program is converted again after the label is changed or deleted, another local label is registered.

Connector element

A connector element is used to place the program within the area to be displayed in the FBD/LD editor or to be printed.



Connection line





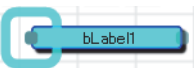
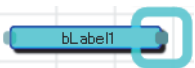
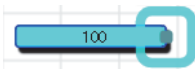
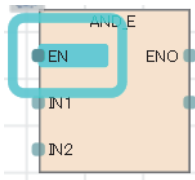
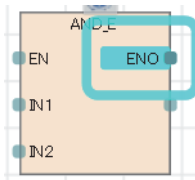
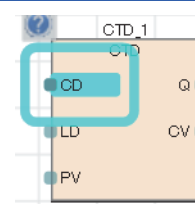
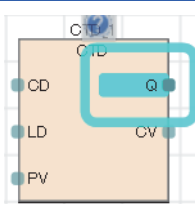
A connection line is a line that connects the end points of FBD element, LD element, and common element.

When connected, the value is passed from the left end to the right end of the line. The data types of the program elements connected must be identical, or support automatic data type conversion.

Connection point

A connection point is an end point of FBD element, LD element, and common element for connecting them to create an FBD/LD program.

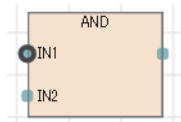
The left-side end point is for input, and the right-side end point is for output.

Item	Connection point for input	Connection point for output
Contact		
Coil		
Variable		
Constant	—	
Function		 The return value of the function is not displayed.
Function block		

When connected, the connection point is hidden.

I/O point inversion

The input value to a program element or the output value from a program element can be inverted at the connection point. The connection point where the value is inverted (FALSE to TRUE or TRUE to FALSE) is displayed with a black circle.

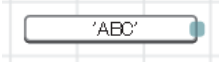
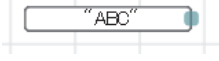


The value of the following data types can be inverted: BOOL, WORD, DWORD, ANY_BIT, and ANY_BOOL.

Constant

Notation of constants

The following table lists the notations of strings in FBD/LD programs.

Data type		Notation	Example
String	STRING	Enclose a string (ASCII, Shift JIS) in single quotation marks ('').	
String [Unicode]	WSTRING	Enclose a Unicode string in double quotation marks (").	

For the notations of constants other than the above, refer to the following.


 Page 47 Constants

Labels and devices


Specification method

Labels and devices can be directly described on FBD/LD programs. Labels and devices can be used as an input point or output point of a program element and an argument or return value of a standard function/function block.

For the applicable labels, refer to the following.

 Page 37 LABELS


For the applicable devices, refer to the following.

 MELSEC iQ-R CPU Module User's Manual (Application)

■Notation of devices with a type specifier

A word device can be used as any data type by adding a device type specifier to its name.

The device type specifiers and applicable devices are the same as those for ST programs. For details, refer to the following.

 Page 71 Notation of devices with a type specifier

If the data type of a word device is not specified, it will be determined by the device type.


Word device	Data type
Current value of the timer (TN), current value of the retentive timer (STN), current value of the counter (CN)	WORD
Current value of the long timer (LTN), current value of the long retentive timer (LSTN), current value of the long counter (LCN)	DWORD
Long index register (LZ)	DINT
Other than above	ANY16

Precautions

When labels are used



- Local devices cannot be used as an array index. To use local devices as an array index, assign the target device to another device, and specify the assigned device.

When performing automatic conversion of data type with devices

Add a device type specifier when using a word device in a data type other than word [signed]. ( Page 82 Notation of devices with a type specifier)

Ex.

When the values of D2 and D3 are transferred to dwordLabel1, a double word [unsigned] label

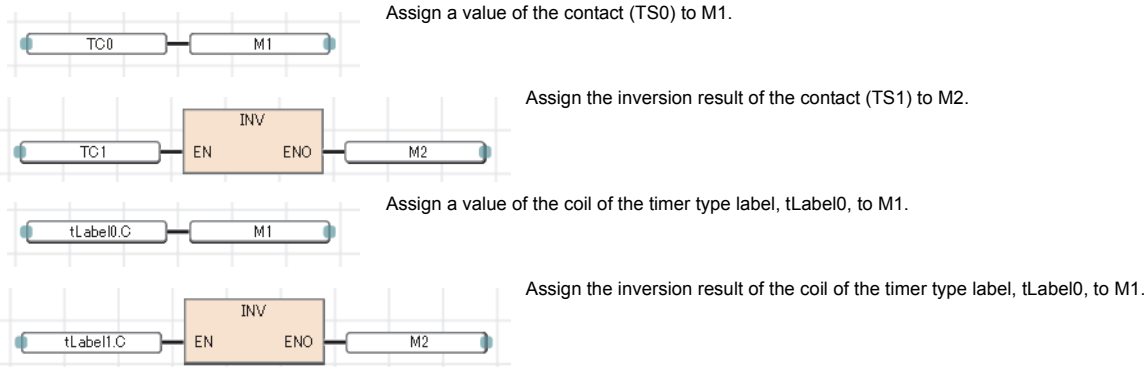
Example of when adding a device type specifier and transferring the values in a correct data type	Example of an unintended transfer result
<div>  <p>Since the data type of D2:UD (D2 with a device type specifier) is double word [unsigned], the values of D2 and D3 are transferred to dwordLabel1.</p> </div>	<div>  <p>Since the data type of D2 without a device type specifier is word [signed], the data type is automatically converted into double word [unsigned] and data is transferred to dwordLabel1. Therefore, the value of D3 is not transferred but only the value of D2 is transferred.</p> </div>

When a timer or counter is used

- If a coil (TC, STC, LTC, LSTC, CC, or LCC) of timer and counter devices is used as an input of variable, function or function block, it operates as a contact (TS, STS, LTS, LSTS, CS, LCS).
- To use a coil of timer or counter as an input, use a timer type or counter type label.

Ex.

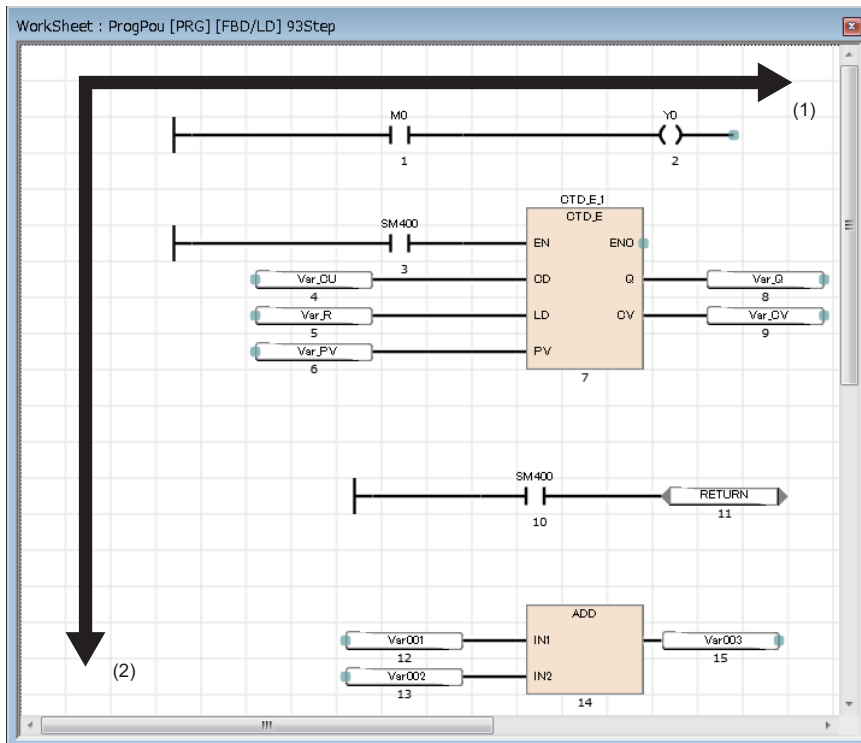
Timer device and timer type label



7.2 Program Execution Order

Execution order of program elements

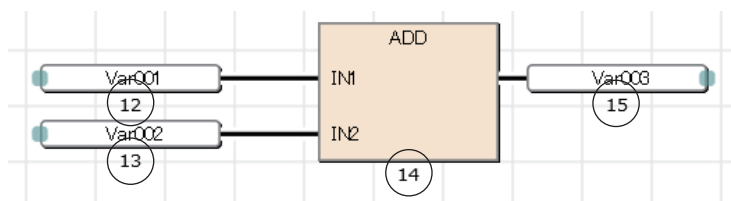
The execution order of program elements in the FBD/LD editor is determined by the location and connecting status.



(1) Program elements are executed from left to right.

(2) Program elements are executed from top to bottom.

The execution order is displayed under each program element. After the program is converted, the determined execution order is displayed.

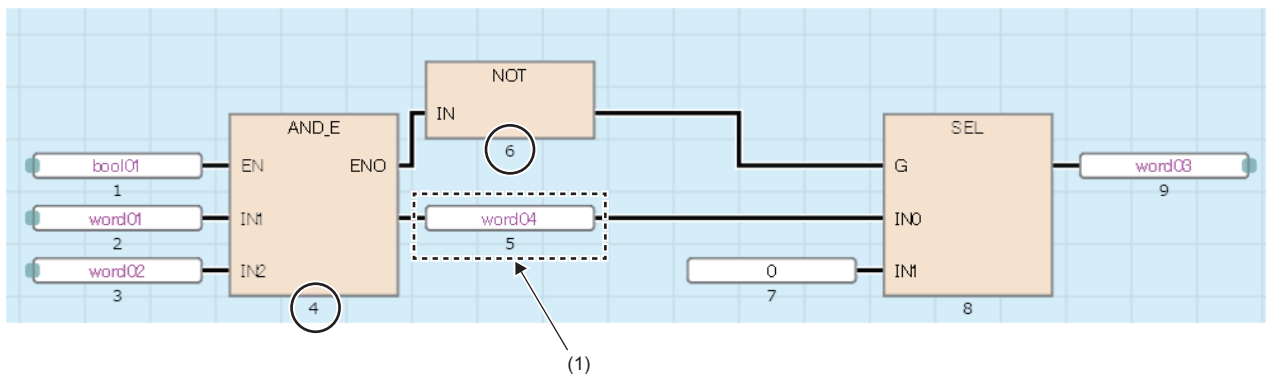


Precautions

For a program that uses functions, do not directly connect the return value of a function and an input variable of another function, but connect a variable element between them.

Ex.

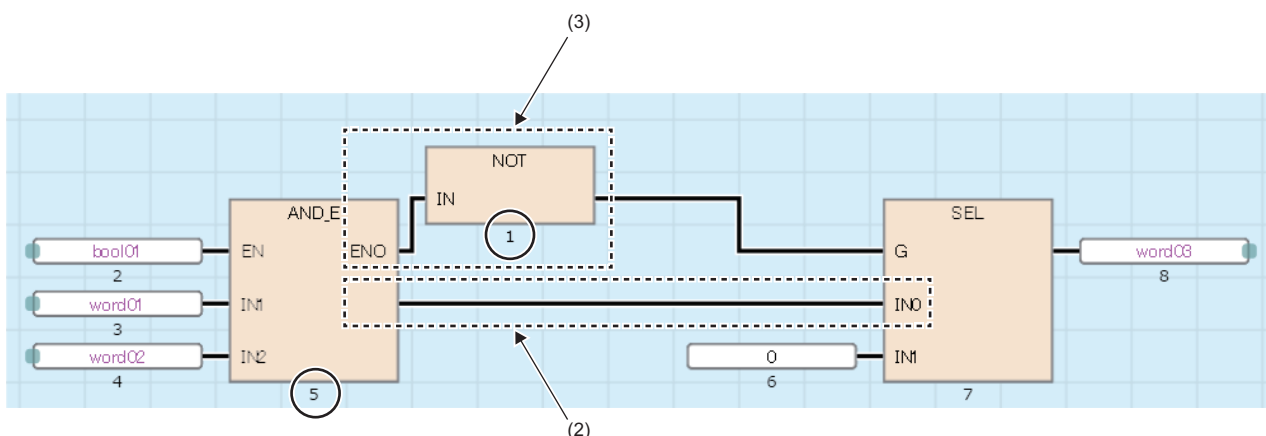
When the variable element (1) is connected between the return value and input variable



Connecting the return value of a function and the input variable of another function directly may lead an execution order to an unintended one.

Ex.

An unintended execution order



Since both the output variable (3) that comes from another program element and the return value (2) of the program element arranged on the left are connected as inputs of the program element arranged on the right, the execution order has changed.

8 SFC PROGRAM



SFC is a program description format in which a sequence of control operations is split into a series of steps to enable a clear expression of each program execution sequence and execution conditions.

Point

- This chapter describes the operations and specifications of SFC programs. For details on the information not described in this chapter, refer to the following.

GX Works3 Operating Manual

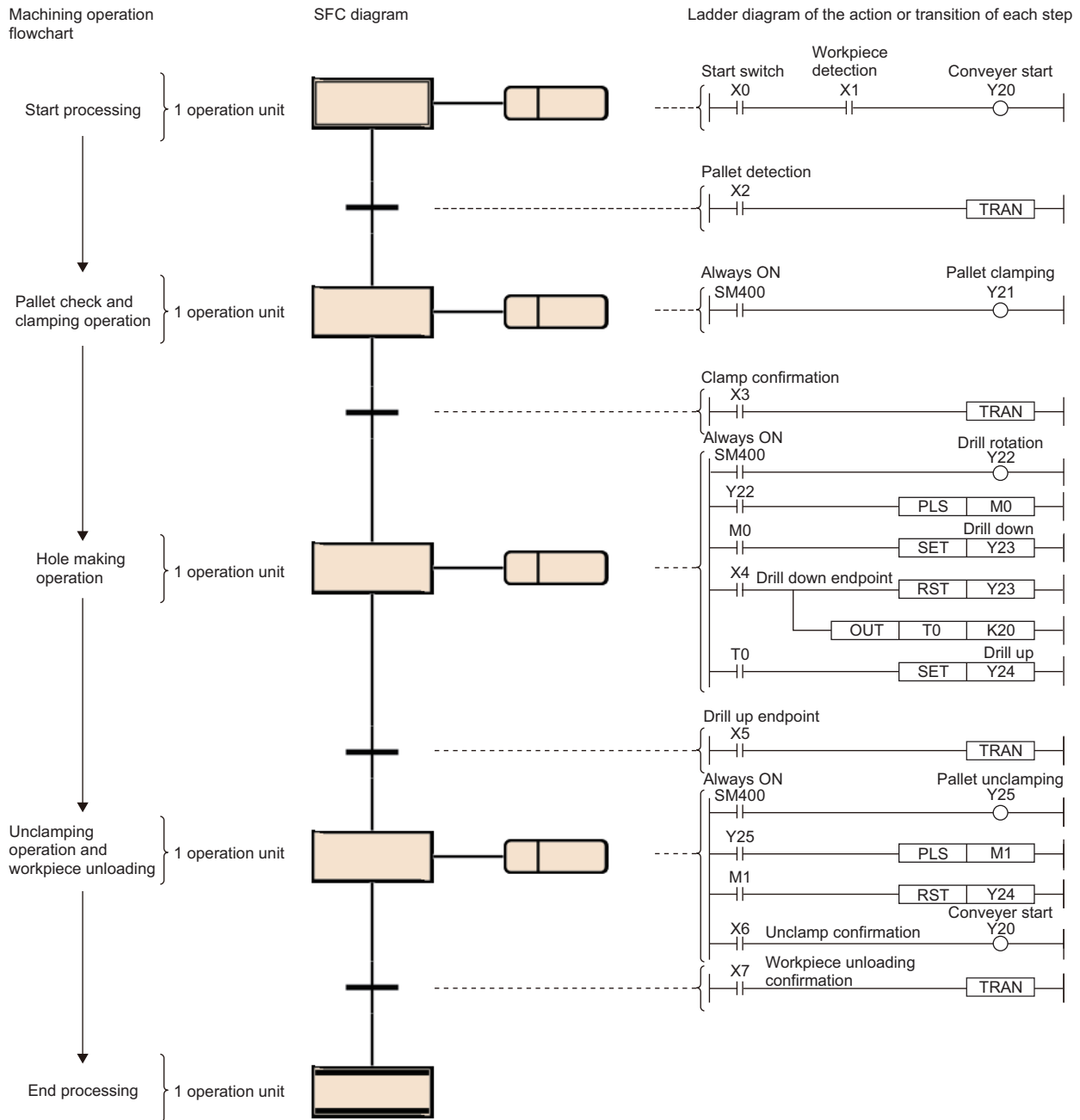
MELSEC iQ-R CPU Module User's Manual (Application)

Restriction

Check the versions of the CPU module and the engineering tool before using the SFC language. For the versions of the CPU module and the engineering tool, refer to the following.

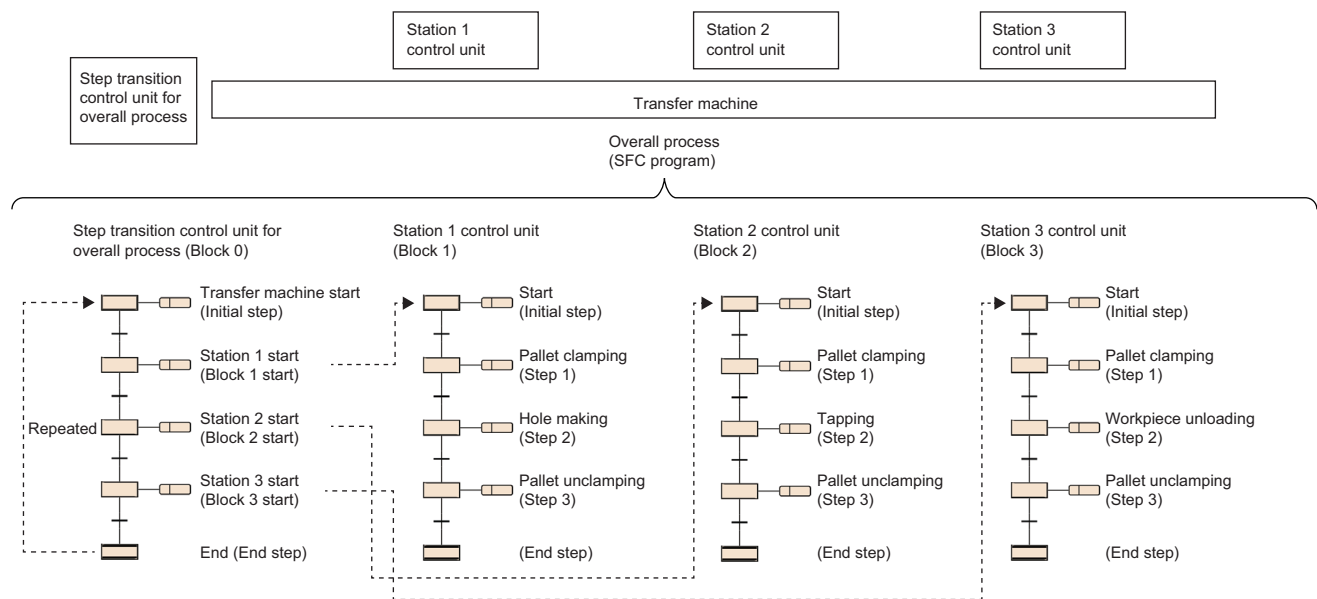
MELSEC iQ-R CPU Module User's Manual (Application)

The SFC program consists of steps that represent units of operations in a series of machine operations. In each step, the actual detailed control is programmed.



An SFC program starts at an initial step, executes an action of the next step in due order every time the relevant transition becomes TRUE, and ends a series of operations at an end step.

It is possible to correspond the controls of the entire facility, mechanical devices of each station, and all machines to the blocks and steps of the SFC program on a one-to-one basis.



8.1 Specifications

This section lists the performance specifications related to SFC Programs.

Item		Specifications
Number of executable SFC programs		1
Number of blocks		320 blocks maximum
Number of SFC steps		All blocks in total: 16384 maximum One block alone: 512 maximum
Number of branches		32 branches maximum
Number of simultaneously active steps		All blocks in total: 1280 maximum One block alone: 256 maximum
Number of initial steps		32 blocks maximum
Number of actions		4 actions maximum per step
Number of sequential steps	Action	About 32K sequential steps per block (No restriction on the number per SFC step)
	Transition	Only one per ladder block

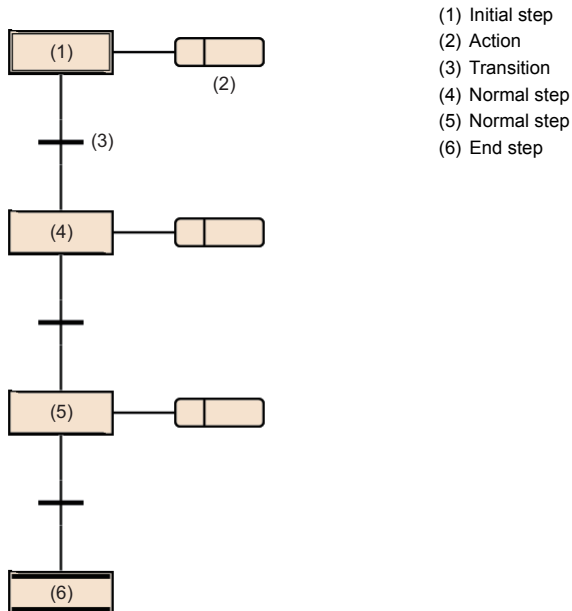


- For the processing time of the SFC program, refer to the following.
 MELSEC iQ-R CPU Module User's Manual (Application)

8.2 Structure



Basic operation

An SFC program starts at an initial step, executes the next step every time the relevant transition becomes TRUE, and ends a series of operations at an end step.



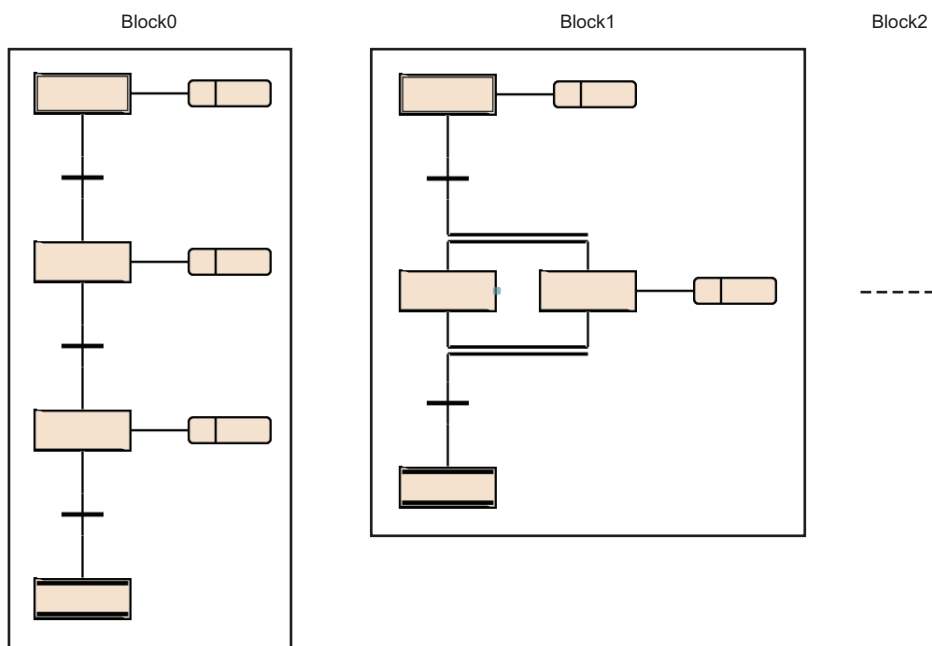
1. When starting a block, the initial step (1) is activated first and then the action (2) is executed. After execution of the action (2), the program checks whether the next transition (3) has become TRUE.
2. The program executes only the action (2) until the transition (3) becomes TRUE. When the transition (3) becomes TRUE, the program ends the action (2), deactivates the initial step (1), and activates the next normal step (4).
3. After execution of the action of the normal step (4), the program checks whether the next transition has become TRUE. If the next transition does not become TRUE, the program repeats the execution of the action of the normal step (4).
4. When the transition becomes TRUE, the program ends the action, deactivates the step (4), and activates the next step (5).
5. Every time the transition becomes TRUE, the program activates the next step and ends the block when it finally activates the end step (6).

Point

- Up to 4 actions can be created in one step. When multiple actions are created, they are executed in order from the top. ( Page 103 Action)
- The operation of the initial step and normal step can be changed by adding the attribute. ( Page 93 Step types)

Block

A block is a unit showing a series of operation consisting of steps and transitions.



Up to 320 blocks can be created in an SFC program.

A block begins with an initial step, a step and a transition are connected alternately, and ends with an end step or jump sequence.

A block has an either state of active or inactive.

- Active: The block has an active step.
- Inactive: All steps in the block are inactive.

When the block state changes from inactive to active, the initial step becomes active to start sequential processing. (☞

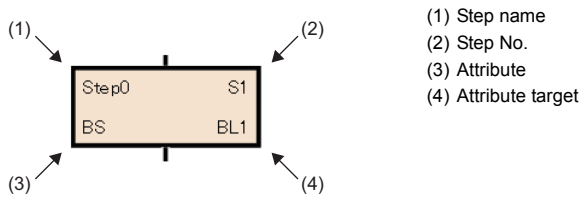
Page 137 Block execution sequence)

Point

- Setting CPU parameters enables only block 0 to be started automatically when the SFC program starts. In this case, when the end step is activated and the block 0 is finished, the block 0 is automatically restarted and execution of steps is started again from the initial step. (☞ Page 131 Start condition setting)
- If a start request is issued to a step in an inactive block by using the SET instruction (activating a step), the block is activated to execute processing from the specified step.

Step

A step is the basic unit for comprising a block.



Steps have the following characteristics.

- When the step becomes active, the related action is executed.
- Up to 512 steps can be created in one block.
- A step No. is assigned to each step. Step Nos. are used to monitor a specific step being executed or forcibly start or stop the step by using the SFC control instruction. (Page 102 Assigning the step relay (S) areas to steps)
- Each step name and No. are unique within each block. (Each cannot be a blank.)

Point

The step name, step No., attribute, and attribute target can be changed from the "Step Properties" window. Select a step and select [Edit]⇒[Properties] in the menu. The "Step Properties" window is displayed. (GX Works3 Operating Manual)

Step types

The following table lists the types of steps.

Item		Description
Initial step		<p>A step that indicates the beginning of a block.</p> <p>While this type of step is active, the transition following the step is always checked, and when the transition becomes TRUE, the next step becomes active.</p> <p>Attributes of SC, SE, ST, and R can be added.</p> <p>This step can also be used as a step without an action.</p>
Normal step		<p>A basic step used to comprise a block.</p> <p>While this type of step is active, the transition following the step is always checked, and when the transition becomes TRUE, the next step becomes active.</p> <p>Attributes of SC, SE, ST, R, BC, and BS can be added.</p> <p>This step can also be used as a step without an action.</p>
End step		<p>A step that ends a block.</p> <p>An action cannot be created.</p>

The following table lists the attributes of steps.

Attribute	Item		Description
SC	Coil HOLD step [SC]		A step that holds the outputs of a coil that has been turned on by the action even after the active state transitions.
SE	Operation HOLD step (without transition check) [SE]		A step which continues the operation of the action even after the active state transitions. After the transition becomes TRUE and the next step is activated, the transition is not checked.
ST	Operation HOLD step (with transition check) [ST]		A step which continues the operation of the action even after the active state transitions. Even after the transition becomes TRUE and the next step is activated the transition is checked repeatedly.
R	Reset step [R]		A step that deactivates the specified step.
BC	Block start step (with END check) [BC]		A step that activates the specified block. When the specified block becomes inactive and the transition becomes TRUE, the active state transitions to the next step. An action cannot be created.
BS	Block start step (without END check) [BS]		A step that activates the specified block. When the transition becomes TRUE, the active state transitions to the next step. An action cannot be created.

Point

- The type of a step can be changed by changing the setting of "Type" in the "Step Properties" window.
- For the reset step [R], block start step (with END check) [BC], or block start step (without END check) [BS], specify a step name or a block No. in "Attribute Target" in the property window.

For the setting method, refer to the following.

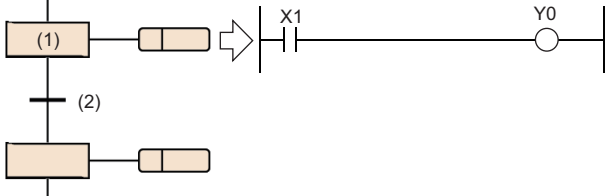
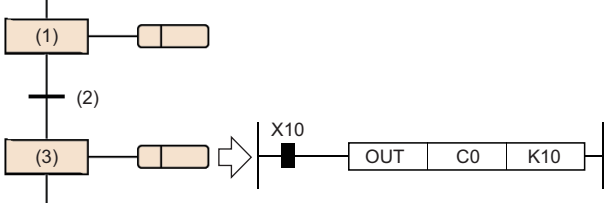
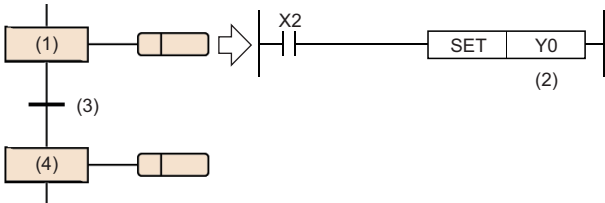
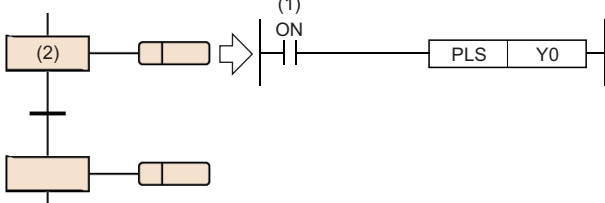
GX Works3 Operating Manual

Normal step (without attribute)

Normal step is a basic step used to comprise a block.

While this type of step is active, the transition following the step is always checked, and when the transition becomes TRUE, the next step becomes active.

The output status of the action of a step, when a transition to the next step occurs, varies depending on the instruction used.


Item	Description	Example
When the OUT instruction is used (Other than the OUT C instruction)	When a transition to the next step occurs and the relevant step becomes inactive, the output by using the OUT instruction turns off automatically. Similarly, the timer also clears the current value and turns off the contact. However, the select statement of structured text language or the output by using the OUT instruction which is repeatedly using within the statement does not turn off automatically.	 <p>When the transition (2) becomes TRUE while Y0 is turned on by using the OUT instruction triggered by the action of step (1), Y0 is automatically turned off.</p>
When the OUT C instruction is used	If the execution condition of the counter in the action is already on when the transition becomes TRUE and activate the step, the counter is incremented by 1. When a transition to the next step occurs before reset instructions of the counter is executed, the present value of the counter and the ON state of the contact is held even if the step becomes inactive. To reset the counter, use the RST instruction in another step.	 <p>If X10 is already on while step (1) is active, counter C0 counts once when execution proceeds to step (3) after the transition (2) becomes TRUE.</p>
When the SET, basic, or application instruction is used	If a transition to the next step occurs and the step becomes inactive, the ON state or the data stored in the device/label is held. To turn off the ON device/label or clear the data stored in the device/label, use the RST instruction in another step.	 <p>When Y0 is turned on by using the SET instruction triggered by the action in step (1), the ON state will be held even when the transition (3) becomes TRUE and a transition to step (4) occurs.</p>
When the PLS instruction or instructions executed at the rising edge is used	Even when the contact of the execution condition is always on, the instruction is executed every time the step changes from inactive to active.	 <p>Even when the contact of the execution condition is on (1), the PLS instruction is executed every time the step (2) becomes active.</p>

Step without action

A step without an action can also be used as a waiting step.

- While a step is active, the transition is always checked and, when the transition becomes TRUE, the next step becomes active.
- This type of step works as a normal step if an action is added to it.

Initial step

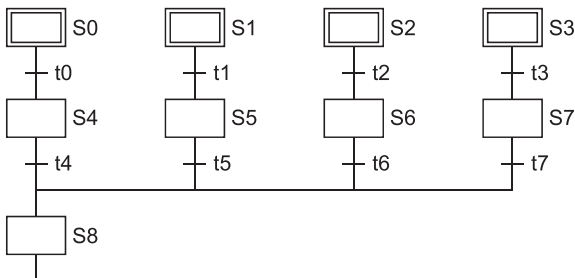
The initial step represents the beginning of a block. Up to 32 initial steps per block can be described. ( Page 89 Specifications) When there are more than one initial step, the convergence enabled is only a selective convergence. Execute the initial steps in the same way as executing other steps.

Active steps at block START

When multi-initial steps are used, the active steps change depending on the starting method as described below.

Operation of active step	Method
All initial steps become active.	When a start is made using the block start step
	When a start is made using the block START instruction of the SFC control instructions
	When a forced start is made using the block start/end bit of the SFC information devices
	When block 0 is started using the auto-start setting of block 0
Only the specified step becomes active.	When any of the initial steps is specified using the step control instruction of the SFC control instructions

Transition processing performed when multi-initial steps become active



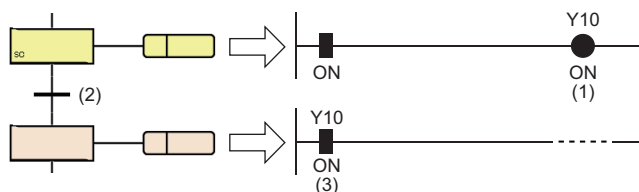
If steps are selectively connected in the block that has more than one active initial steps, the step immediately after the convergence becomes active when any of the transition conditions immediately before the convergence is satisfied. In the above program example, step 8 (S8) becomes active when any of transition conditions t4 to t7 is satisfied. When the step immediately after the convergence (S8 in the above program example) becomes active and another transition condition immediately before the convergence (any of t4 to t7 in the above program example) is satisfied, the step immediately after the convergence becomes active again.

Operation of the initial steps with step attributes

An attribute of SC (coil HOLD step), SE (operation HOLD step (without transition check)), ST (operation HOLD step (with transition check)), or R (reset step) can be added to the initial step. When an attribute is added, the operation other than automatic activation when block is started is the same as the operation of other steps. This step can also be used without an action.

Coil HOLD step [SC]

Coil HOLD step [SC] is a step that holds the outputs of a coil that has been turned on by the action even after the active state transitions.



Y10 (1) that has been turned on by using the OUT instruction remains on (3) even when the transition (2) becomes TRUE.

No operation in the action is performed after a transition becomes TRUE and the next step is activated. Therefore, the coil output status will remain unchanged even if the input condition in the action is changed.

■Timing of when coil output turns off

The coil output holding ON state is turned off in the coil HOLD step [SC] after transition when:

- The end step of a block is executed (other than the case where SM327 is on).
- A block is forcibly terminated by using the RST instruction (Ending a block).
- A step is reset by using the RST instruction (Deactivating a step).
- The device specified as the block START/END bit of the SFC information devices is reset.
- A reset step [R] for resetting the coil HOLD step [SC] becomes active.
- SM321 (SFC program start/stop) is turned off.
- The coil is reset by the program.
- The stop instruction is executed with the output mode at block stop set off.
- S999 is specified at a reset step [R] within a block.

■Operation when the block is paused or restarted

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. (Page 132 Operation when the block is paused or restarted)

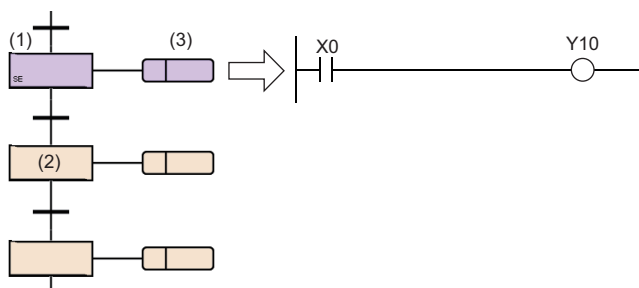
Operation HOLD step (without transition check) [SE]

Operation HOLD step (without transition check) [SE] is a step which continues the operation of the action even after the active state transitions.

This step continues the operation in the action even after a transition becomes TRUE and the next step is activated.

Therefore, when the input condition changes, the coil status also changes.

After the transition becomes TRUE and the next step is activated, the transition is not checked and the transition to the next step does not occur.



When step (2) is activated, step (1) holds the operation.
While holding the operation, the transition is not checked but the action (3) is kept executed.
In this case, Y10 turns on or off accordingly as X0 turns on or off.

■Deactivation timing

An operation HOLD step (without transition check) [SE] becomes inactive when:

- The end step of a block is executed.
- A block is forcibly terminated by using the RST instruction (Ending a block).
- A step is reset by using the RST instruction (Deactivating a step).
- The device specified as the block START/END bit of the SFC information devices is reset.
- A reset step [R] for resetting the operation HOLD step (without transition check) [SE] becomes active.
- SM321 (SFC program start/stop) is turned off.
- S999 is specified at a reset step [R] within a block.

■Operation when the block is paused or restarted

Operation when the block is paused or restart depends on the combination of SM325 (Output mode at block stop), block stop mode bit setting of SFC information device, and step hold status. (☞ Page 132 Operation when the block is paused or restarted)

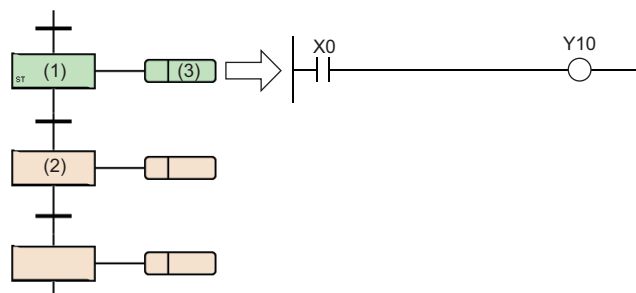
Operation HOLD step (with transition check) [ST]

Operation HOLD step (with transition check) [ST] is a step which continues the operation of the action even after the active state transitions.

This step continues the operation in the action even after a transition becomes TRUE and the next step is activated.

Therefore, when the input condition changes, the coil status also changes.

Even after the transition becomes TRUE and the next step is activated the transition is checked repeatedly. When the transition becomes TRUE again, the operation in the action is continued while activating the next step again.



When step (2) is activated, step (1) holds the operation.

The action (3) is kept executed the same as a normal active step while the step holds the operation.

In this case, Y10 turns on or off accordingly as X0 turns on or off.

The transition is also checked and, when it becomes TRUE, the next step becomes active.

■Deactivation timing

An operation HOLD step (with transition check) [ST] becomes inactive when:

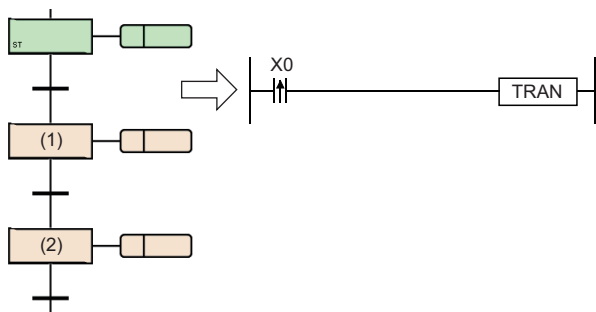
- The end step of a block is executed.
- A block is forcibly terminated by using the RST instruction (Ending a block).
- A step is reset by using the RST instruction (Deactivating a step).
- The device specified as the block start/end bit of the SFC information devices is reset.
- A reset step [R] for resetting the operation HOLD step (with transition check) [ST] becomes active.
- SM321 (SFC program start/stop) is turned off.
- S999 is specified at a reset step [R] within a block.

■Operation when the block is paused or restarted

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. (☞ Page 132 Operation when the block is paused or restarted)

■Precautions

- For the operation HOLD step (with transition check) [ST], the next step is activated every scan while the transition immediate after the operation HOLD step becomes TRUE. To prevent transition every scan, use instructions executed on the rising edge such as the PLS instruction for the transition.



By setting the start of rising edge pulse operation as the transition, step (1) is activated during only one scan caused when X0 is turned on.

Even when step (2) is activated and becomes inactive, step (1) is not activated unless X0 is turned off and on again.

- When SM328 (Clear processing mode when the sequence reaches the end step) is on, prevent the transition immediately after the operation HOLD step (with transition check) [ST] from becoming always TRUE. Otherwise, the next step is kept activating and holding no operation, therefore the block cannot be ended.

Reset step [R]

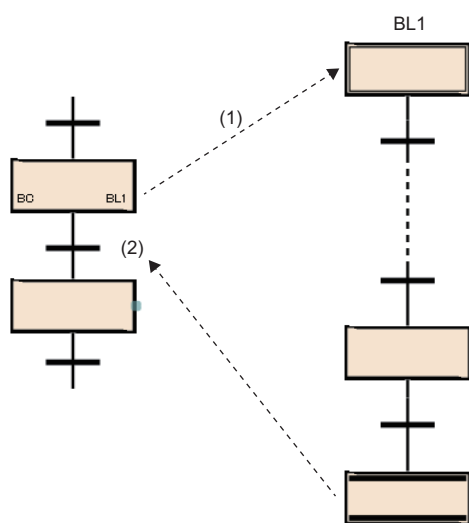
Reset step [R] is a step that deactivates the specified step.

- The reset step [R] deactivates the specified step in the current block before execution of the action every scan. Except for resetting the specified step, the reset step is the same as a normal step (without step attributes).
- When the specified step No. is S999, the HOLD steps [SC, SE, ST] that hold operations in the current block are all deactivated. In this case, only the HOLD steps [SC, SE, ST] that hold operations can be deactivated. However, any operation HOLD step [SE, ST] is not deactivated when operating with the state that does not hold an operation.
- The current step No. cannot be specified as specified step No.

Block start step (with END check) [BC]

Block start step (with END check) [BC] is a step that activates the specified block.

When the specified block becomes inactive and the transition becomes TRUE, the active state transitions to the next step.



When this step is activated, the block start step (with END check) [BC] starts block (BL1). No processing is performed until the execution of the start destination block (BL1) ends and becomes inactive and the transition (2) is not checked. When the execution of block (BL1) ends and becomes inactive, only the transition (2) check is performed and, when the condition becomes TRUE, the transition to the next step occurs.

The operation to be performed if multiple attempts to start one block are performed simultaneously or if an attempt to start an already started block is performed follows the operation setting applicable to the block double start. (Page 134 Act at block multi-activated)

Only one block can be specified. To start multiple blocks simultaneously, use parallel branches and multiple block start steps.

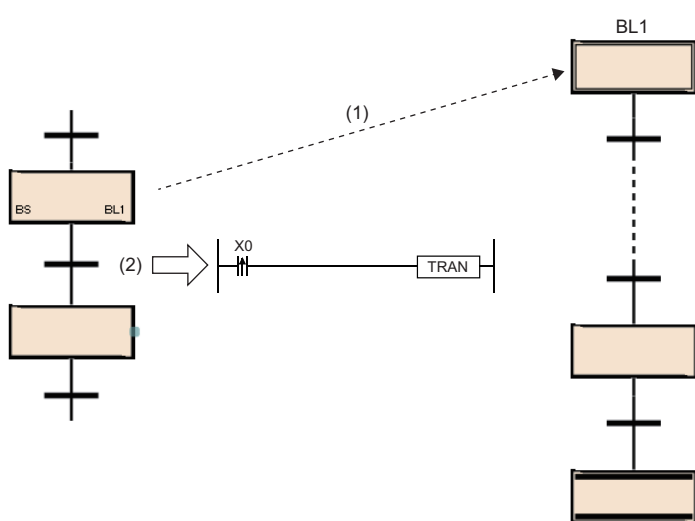
■Precautions

- An action cannot be created to the block start step (with end check) [BC].
- The block start step (with END check) [BC] cannot be created immediately before convergence of a parallel convergence. To create the step immediately before the convergence of a parallel convergence, use a block start step (without END check) [BS].

Block start step (without END check) [BS]

Block start step (without END check) [BS] is a step that activates the specified block.

When the transition becomes TRUE, the active state transitions to the next step.



After this step starts block (BL1), only the transition (2) is checked and, when the transition becomes TRUE, execution proceeds to the next step without waiting for the start destination block to end.

The operation to be performed if multiple attempts to start one block are performed simultaneously or if an attempt to start an already started block is performed follows the operation setting applicable to the block double start. (Page 134 Act at block multi-activated)

Only one block can be specified. To start multiple blocks simultaneously, use parallel branches and multiple block start steps.

■Precautions

- An action cannot be created to the block start step (without END check) [BS].

End step

End step is a step that ends a block.

- When the active state transitions to the end step and no active step exists other than steps that hold operations in the block, all the HOLD steps [SC, SE, ST] that hold operations in the block are deactivated and the block is ended.
- When a block contains any active steps other than steps that hold operations in a block, the following processing is performed depending on the status of SM328 (Clear processing mode when the sequence reaches the end step).

Status of SM328	Description
Off (default)	Clear processing is performed. The active steps remaining in the block are all terminated forcibly to end the block.
On	Clear processing is not performed. The execution of the block is continued as is and the block is not ended.

- When clear processing is performed, the coil outputs turned on by using the OUT instruction are all turned off. However, for the coil output of the HOLD steps [SC, SE, ST] that hold operations, the following processing is performed depending on the status of SM327 (Output mode at execution of the end step).

Status of SM327	Description
Off (default)	All the HOLD steps [SC, SE, ST] that hold operations are turned off.
On	All the outputs of the HOLD steps [SC, SE, ST] that hold operations are held. The setting of SM327 is valid for only the HOLD steps [SC, SE, ST] that hold operation. All the outputs of the HOLD steps [SC, SE, ST] that do not hold operations and the transition does not become TRUE are turned off. Also, when SM327 is on, the steps become inactive. However, when a forced end is performed such as by the block end instruction, the coil outputs of all steps are turned off.

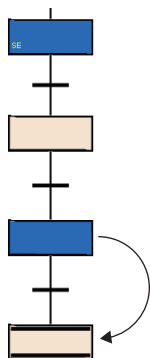
- The following shows how to restart the block once ended.

Item	Description
Block 0	The start condition of block 0 is set to "Auto-start block 0" in the SFC setting of parameters.
	The start condition of block 0 is set to "Do not auto-start block 0" in the SFC setting of parameters.
All blocks other than block 0	The block is restarted when a start request is issued for the specified block in the following methods. <ul style="list-style-type: none"> • The block start step is activated by another block. • The SET (Starting a block) instruction is executed. • The block START/END bit of the SFC information device is turned on.

■Precautions

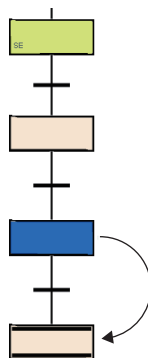
- An action cannot be created to the end step.
- The setting of SM327 (Output mode at execution of the end step) is valid only when the end step becomes active. When a forced termination is performed such as by using the RST instruction (Ending a block), the coil outputs of all steps are turned off.
- If the HOLD steps [SC, SE, ST] holding operations remain when the end step becomes active, those steps [SC, SE, ST] are deactivated even though SM328 (Clear processing mode when the sequence reaches the end step) is on. If turning off the coil outputs of the HOLD steps [SC, SE, ST] that hold operations is not required, turn on SM327. The following figure shows the operational relationships between SM328 and HOLD steps [SC, SE, ST].

When a normal active step remains or when a HOLD step [SC, SE, ST] whose transition has not become TRUE remains (the step does not hold an operation)



- When SM328 is off, the block is ended by clearing the step.
- When SM328 is on, processing is continued without clearing the step.

When an active step that holds an operation remains



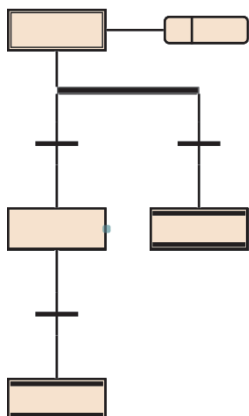
- The block is ended by clearing the step regardless of the setting of SM328.

- If a block is started at the block start step when SM328 is on, execution returns to the source as soon as there are no active step that does not hold the operation in the block.
- Prevent the transition after the operation HOLD step (with transition check) [ST] from becoming always TRUE. When the transition immediately after the operation HOLD step (with transition check) [ST] always becomes TRUE, the next step is kept active and, therefore, the block can no longer be ended when SM328 is on.

Point

Multiple end steps can be created in the SFC diagram.

To do so, select a step in the selection branch and select [Edit] ⇒ [Modify] ⇒ [End Step/Jump] from the menu.



Assigning the step relay (S) areas to steps

The step relay is a device corresponding to each step in the SFC program. It is on when the relevant step is active (including stop and hold state), and is off when the relevant step is inactive.

Step relays are assigned as follows.

- Step relays are assigned sequentially in order of block No. starting from block 0 in an SFC program and in order of step No. within a block.
- No step relay is assigned to any non-existing block No.
- Similarly, no step relay is assigned to any missing step No. within a block. The relevant bit is always off.
- All bits after the step relays assigned in the last block are off.

Ex.

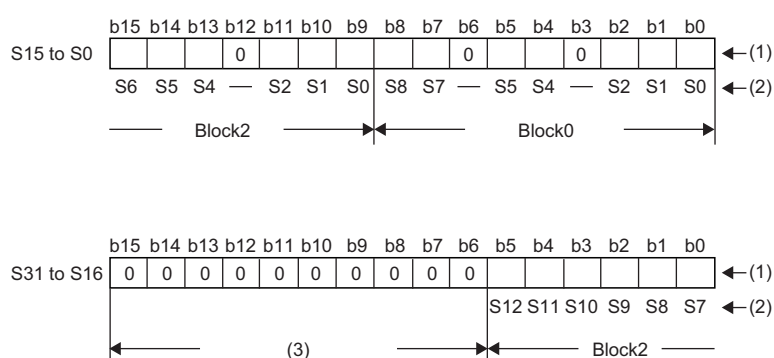
The following example shows the step relay assignments of the following block configuration.

Block0: The largest step No. is 8, and step Nos. 3 and 6 are missing.

Block1: Missing

Block2: The largest step No. is 12, and step No. 3 is missing.

Block3 and after: Missing



(1) Stored data

(2) Step numbers in a block

(3) All 0s for missing blocks

Point

Any step No. can be assigned to each step (except end step).

- Assign step numbers in ascending order wherever possible because any missing step No. will decrease the maximum number of steps that can be created.
- The step No. other than step No.0 (S0) cannot be used for the initial step of the top line and left end.

Step No. 0 is assigned to the first initial step in a block.

Step numbers that can be assigned in a block range from 0 to 511. Any step No. exceeding the upper limit cannot be assigned. Any step No. must be unique within a block. Same step numbers can be used between different blocks.

To specify a step relay in another block, use the following format.

Ex.

Specifying step No. 23 in block No. 12

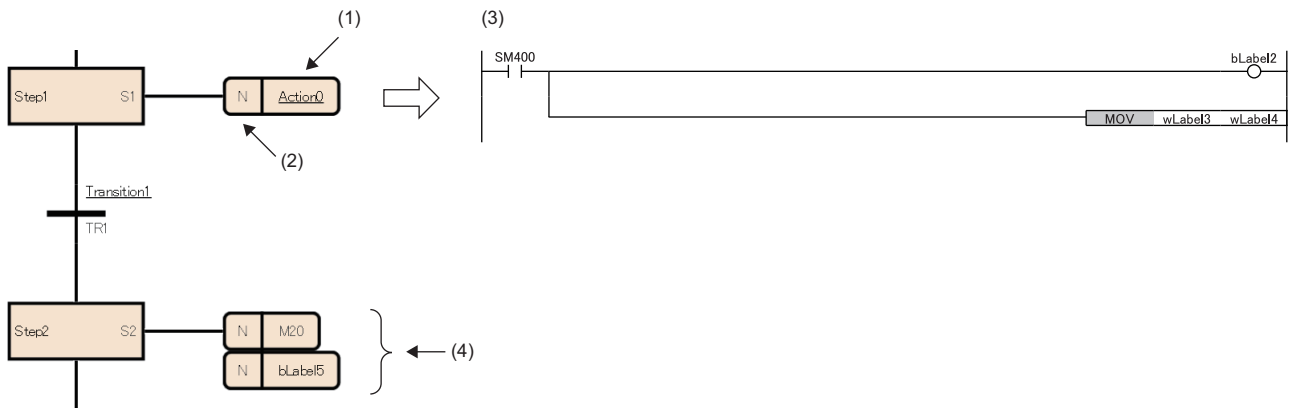
Program type		Device notation	Description
SFC program	In the same block	S23	The block name can be omitted when specifying a step in the same block
	Other than block 12	BL12\S23	Specify the target block No. and step No.
Sequence program other than SFC program	Specifying the current target block	S23	The block name can be omitted when specifying a step in the target block
	Specifying a block different from the current target block	BL12\S23	Specify the target block No. and step No.

■Precautions

- Even if "Output Mode at Block Stop" of the SFC setting is off, the step relay is on when the step is stopped the operation.

Action

An action is a program which is executed while a step is active.



(1) Action name

(2) Qualifier^{*1}

(3) Detailed expression of the action

(4) Action label/device

^{*1} N indicates that the action is executed while a step is active. Nothing but N can be set.

When the step becomes active, the action is executed every scan. When the step becomes inactive, the action is ended and not executed until next time the step becomes active.

Up to 4 actions can be created in one step. When multiple actions are created, they are executed in order from the top.

Detailed expression of an action can be created in ladder diagrams, ST language, or FBD/LD. In ladder diagrams, the description method can be switched between detailed expression and MELSAP-L (instruction format). (📖 Page 104 Action in MELSAP-L (instruction format))

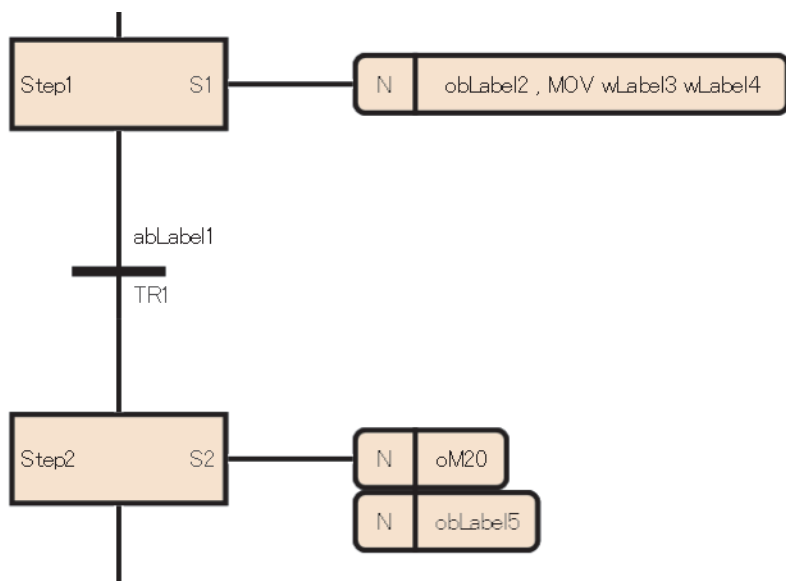
Point

For details on detailed expression or labels/devices, refer to the following.

📖 GX Works3 Operating Manual

Action in MELSAP-L (instruction format)

In MELSAP-L (instruction format), instructions for actions are described in text format in a SFC diagram.



Point

To switch from detailed expression in ladder diagrams to MELSAP-L (instruction format), select [View] ⇒ [Switch Ladder Display] ⇒ [MELSAP-L (Instruction Format)] from the menu. (GX Works3 Operating Manual)

For actions in MELSAP-L (instruction format), instructions and coils to output are described without contacts to be input conditions of each instruction.

In MELSAP-L (instruction format), programs are described in the following format.

□: Applicable label/device, Kn: Setting value of timer/counter

Item	MELSAP-L (instruction format)	Example
Coil output (OUT instruction)	o□	oY0
Setting devices (SET instruction)	s□	sM0
Resetting devices (RST instruction)	r□	rM0
Low-speed timer (OUT T instruction) ^{*1}	o□ Kn	oT0 K100
High-speed timer (OUTH T instruction)	h□ Kn	hT1 K10
Counter (OUT C instruction) ^{*1}	o□ Kn	oC0 K10
Instructions other than listed above ^{*2}	Describe instructions in the same way as in ladder diagrams.	MOV D10 D120

*1 Specify a long timer or a long counter in the same way.

*2 Some instructions cannot be used. (Page 105 Instructions that cannot be used)

To describe multiple instructions, delimit them by a comma (.). When using the IMASK or NOPLF instruction, describe them in the end of the action.

Instructions that cannot be used

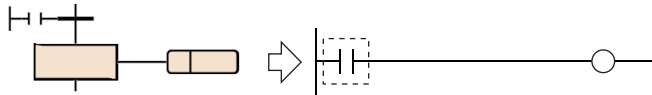
Some instructions cannot be used in actions. The following table lists the instructions that cannot be used.

Classification	Instruction symbol
Master control instruction	MC* ¹
	MCR* ¹
Termination instruction	FEND
	END
Program branch instruction	CJ* ¹
	SCJ* ¹
	JMP* ¹
	GOEND
Program execution control instruction	IRET
Structure creation instruction	BREAK* ¹
	RET
Creating a dummy transition condition	TRAN

*¹ This instruction can be used in a function or a function block in the action.



Create a contact to be input condition of each instruction in the ladder of detailed expression.



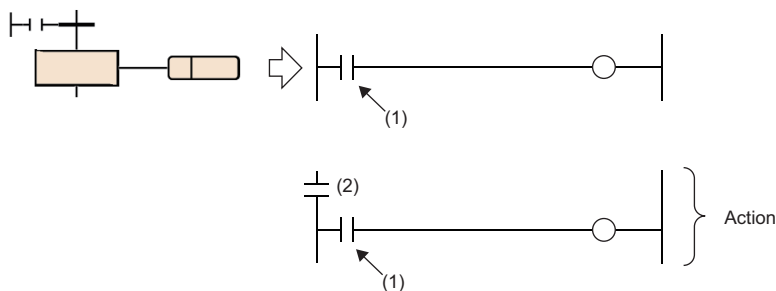
Restrictions

The following table lists the restrictions on individual programming languages used to create an action.

Language		Description
Ladder diagram	Detailed expression	<p>A pointer and an interrupt pointer cannot be input in the pointer input area.</p> <p>■ Functions/function blocks that cannot be used</p> <ul style="list-style-type: none"> Function/function block that includes an instruction that cannot be used in an action Function/function block that includes a pointer A macro type function block for which "Use MC/MCR in EN Control" is set to "Yes" and "Use EN/ENO" is set to "No"
	MELSAP-L (instruction format)	The instructions corresponding to contacts (including comparison operation instructions such as LD<), the NOP, MPS, MRD, and MPP instructions, pointers, interrupt pointers, functions, and function blocks cannot be described.
Structured text language		☞ Page 58 STRUCTURED TEXT LANGUAGE
FBD/LD		☞ Page 74 FBD/LD

Precautions

- The step operation is almost the same as the following circuit.



(1)Input condition of each instruction

(2)Contact (on when active, off when inactive) indicating the step status

- If the CALL instruction is used to issue a subroutine call in an action of the step, the output of the call destination is not turned off even when the step becomes inactive after the transition becomes TRUE. To turn off the output of the call destination when the step becomes inactive after the transition becomes TRUE, write the FCALL instruction after the CALL instruction or use the XCALL instruction. When using a subroutine call in an action of the step, using the XCALL instruction can reduce the number of steps.
- Even when the input condition in the action is always on, it is assumed to be off when the action is inactive. Therefore, immediately after the step becomes active, the instruction is executed when the output is turned on. For example, when the input condition is set to be always on by using the instructions executed at the rising edge such as the PLS or INCP instruction, the instruction is executed every time the step becomes active.
- The device that turned on by the OUT C instruction, the SET instruction, a basic instruction, or an application instruction in the action is not turned off even when the step is deactivated and the action is ended. To turn off the device, execute the RST instruction separately.
- With the PLS or PLF instruction, the specified device is normally turned on for only one scan and thereafter becomes off. However, the specified device holds the ON state if it is turned on at the same time when the transition of the coil HOLD step [SC] becomes TRUE. In this case, it is turned off by changing the condition to the one where the coil output of the coil HOLD step [SC] turns off or activating the step again. For the conditions where the coil output turns off, refer to the following.

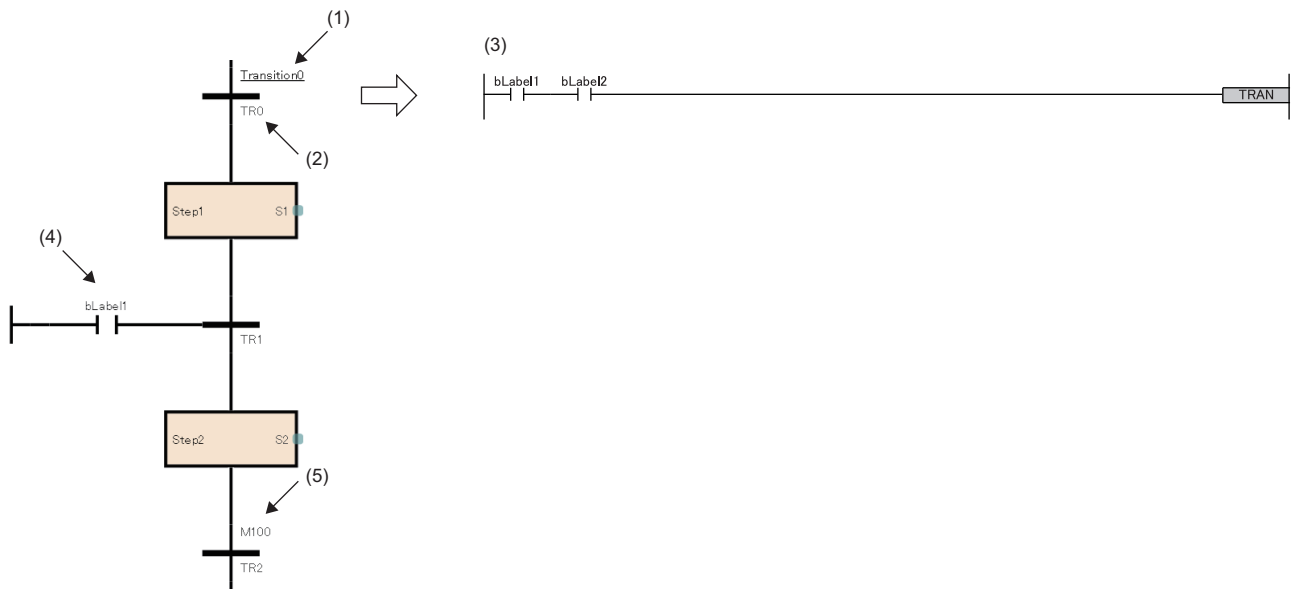
Page 96 Timing of when coil output turns off

- If the step is deactivated and the action is ended while the input condition of the PLF instruction is on, the specified device remains on.
- When the transition becomes TRUE in the coil HOLD step [SC] or the step is stopped by SM325 (Output mode at block stop) which is set to hold, operation may not be performed just holding the coil output. This case means the non-execution status, and therefore the operation of each instruction at the time of operation resumption depends on the execution condition before the no-execution status is entered.
- When a program which cannot be described in MELSAP-L (instruction format) is created in detailed expression in ladder diagrams and switched to MELSAP-L (instruction format), the program will be displayed "????????". When the definition of the label used in a program is deleted, the program will be also displayed in the same way. To check and modify the program, switch to detailed expression.
- When a program which has been created in MELSAP-L (instruction format) is switched to detailed expression in ladder diagrams, a contact SM400 (Always ON) will be added as an execution condition of an instruction.

MELSAP-L (instruction format)	Detailed expression in ladder diagrams

Transition

A transition is the basic unit for comprising a block and transfers the active state to the next step when the condition becomes TRUE.


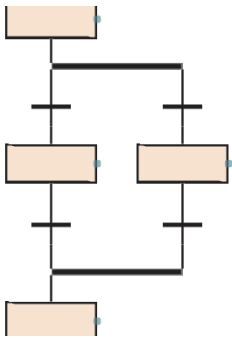
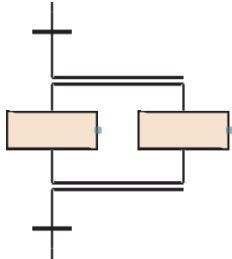
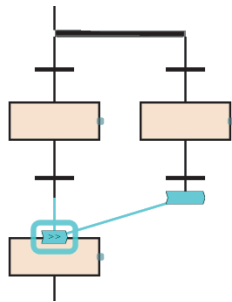


- (1) Transition name
- (2) Transition No.
- (3) Detailed expression of the transition ([Page 113 Detailed expression of transitions](#))
- (4) Direct expression of a transition ([Page 117 Direct expression of transitions](#))
- (5) Transition label/device ([Page 117 Transition label/device](#))

Detailed expression of a transition can be created in ladder diagrams, ST language, or FBD/LD. In ladder diagrams, the description method can be switched between detailed expression and MELSAP-L (instruction format). ([Page 115 Transition in MELSAP-L \(instruction format\)](#))

Transition types

The following table lists the types of transition.

Item		Description
Series sequence		When the transition becomes TRUE, the active state transitions from the preceding step to the subsequent step.
Selective sequence (divergence/convergence)		Divergence: A step branches to multiple transitions, and only the step in the line where the transition becomes TRUE first is activated. Convergence: The next step is activated when the transition immediately before convergence, which is in the line where the transition becomes TRUE first, becomes TRUE.
Simultaneous sequence (divergence/convergence)		Divergence: All the steps branched from one step are activated simultaneously. Convergence: When all the steps immediately before convergence are activated and the common transition becomes TRUE, the active state transitions to the next step.
Jump sequence		When the transition becomes TRUE, the active state transitions to the specified step in the same block.

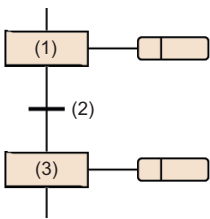
Point

For the operation of transition to the step which is already activated, refer to the following.

☞ Page 144 Behavior when an active step is activated

Series sequence

When the transition becomes TRUE, the active state transitions from the preceding step to the subsequent step.



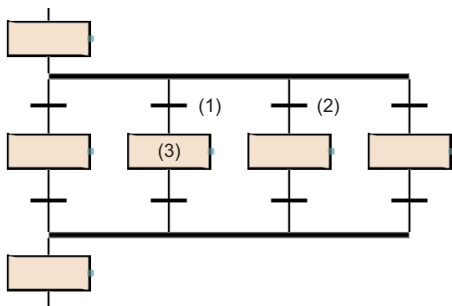
When the transition (2) becomes TRUE while the step (1) is active, the step (1) is deactivated and the step (3) is activated.

Selective sequence (divergence/convergence)

A step branches to multiple transitions, and only the step in the line where the transition becomes TRUE first is activated. The next step is activated when the transition immediately before convergence, which is in the line where the transition becomes TRUE first, becomes TRUE.

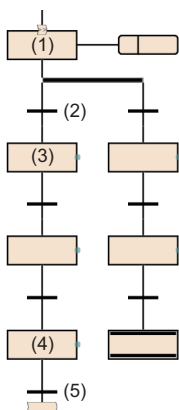
Item	Description	
Divergence	<p>The diagram shows a vertical line starting from a box labeled (1). From the bottom of box (1), the line splits into two parallel vertical paths. The left path has a horizontal bar labeled (2) above a box labeled (4). The right path has a horizontal bar labeled (3) above a box labeled (5). Both boxes (4) and (5) have small horizontal bars on their right sides. Below each of these boxes is another horizontal bar, and from each of these bars, a vertical line descends to a final box at the bottom.</p>	<p>When the step (1) is active, the step (4) or (5) is activated depending on which of the transition (2) and transition (3) becomes TRUE first.</p> <p>The step (1) becomes inactive. However, if it is a HOLD step [SC, SE, ST], the step holds the coil output or action according to its attribute.</p> <ul style="list-style-type: none">• If multiple transitions become TRUE simultaneously, the condition to the left will take precedence.• Subsequent processing will proceed from step to step in the selected column until another convergence occurs.
Convergence	<p>The diagram shows two parallel vertical paths at the top. The left path starts with a box labeled (3), followed by a horizontal bar labeled (1), and then a box labeled (5). The right path starts with a box labeled (4), followed by a horizontal bar labeled (2), and then a box labeled (5). Both boxes (3) and (4) have small horizontal bars on their right sides. From the bottom of box (5) in the left path, a vertical line descends to a final box at the bottom. From the bottom of box (5) in the right path, a vertical line descends to the same final box at the bottom.</p>	<p>When the transition (1) or transition (2) on the activated branch becomes TRUE, the step (5) is activated.</p> <p>The activated step (3) or step (4) becomes inactive. However, if it is a HOLD step [SC, SE, ST], the step holds the coil output or action according to its attribute.</p>

- The selective sequence allows branching to up to 32 transition.
- If multiple transitions become TRUE simultaneously, the condition to the left will take precedence.



If transition (1) and (2) become TRUE simultaneously, the action of step (3) will be executed.

- An SFC diagram in which the numbers of branches and convergences of a selective sequence do not match can also be created. However, in an SFC diagram, a selection branch and parallel convergence or a parallel branch and selective convergence cannot be combined.
- In a selective transition, a convergence can be omitted by a jump transition or end transition.



When transition (2) becomes TRUE during action of step (1), step (3) and step (4) are sequentially executed. When the transition (5) becomes TRUE, a jump sequence to step (1) occurs.

Point

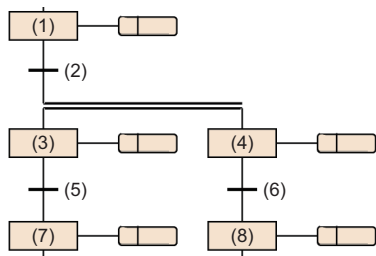
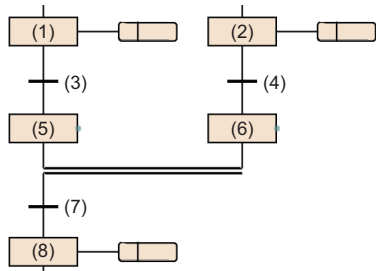
The above program can be created by changing the step other than those at the left end of selective branches to the end step and changing the end step at the left end of the selective branch to a jump sequence.

For the operation method for changing steps, refer to the following.

GX Works3 Operating Manual

Simultaneous sequence (divergence/convergence)

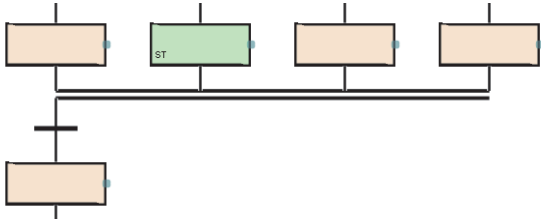
All the steps branched from one step are activated simultaneously. When all the steps immediately before convergence are activated and the common transition becomes TRUE, the active state transitions to the next step.

Item	Description
Divergence	 <p>When the transition (2) becomes TRUE while the step (1) is active, both of the step (3) and step (4) are activated at the same time. The step (1) becomes inactive. However, if it is a HOLD step [SC, SE, ST], the step holds the coil output or action according to its attribute. Processing will proceed to step (7) when transition (5) becomes TRUE, and to step (8) when transition (6) becomes TRUE.</p>
Convergence	 <p>When the transition (3) and transition (4) become TRUE while the step (1) and step (2) are active, the step (5) and step (6) are activated. After the step (5) and step (6) immediately before the convergence become active, the transition (7) is checked and then becomes TRUE, the step (8) is activated. The step (5) and step (6) become inactive. However, if it is a HOLD step [SC, SE, ST], the step holds the coil output or action according to its attribute.</p>

- The simultaneous sequence allows transitions to up to 32 steps.
- If another block is started by the simultaneous sequence, the START source block and START destination block will be executed simultaneously.
- A simultaneous convergence is always performed after a simultaneous branch.

■Precautions

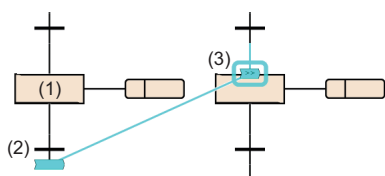
- When the steps connected by a convergence include HOLD steps [SC, SE, ST] that hold operations, the operation is performed as follows.

Item	Description
Coil HOLD step [SC]	A transition to the next step does not occur the same as an inactive step.
Operation HOLD step (without transition check) [SE]	
Operation HOLD step (with transition check) [ST]	<p>A transitions to the next step occurs if another connected step is active.</p> 

- In the simultaneous convergence, a block start step (with END check) [BC] cannot be created immediately before the convergence. Use a block start step (without END check) [BS].

Jump sequence

When the transition becomes TRUE, the active state transitions to the specified step in the same block.



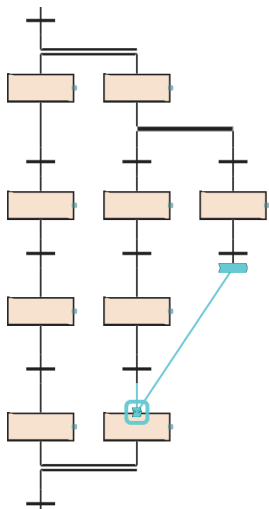
When the transition (2) becomes TRUE while the step (1) is active, the step (3) is activated.

The step (1) becomes inactive. However, if it is a HOLD step [SC, SE, ST], the step holds the coil output or action according to its attribute.

- There are no restrictions regarding the number of jump sequences.
- A jump sequence in the simultaneous sequence is possible only in the same branch. A jump sequence to another branch within a simultaneous branch, a jump sequence for exiting from a simultaneous branch, or a jump sequence to a simultaneous branch from outside a simultaneous branch cannot be created.

Ex.

Example of jump sequence that can be specified in the simultaneous branch



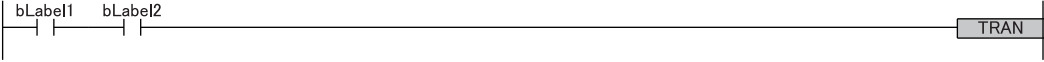
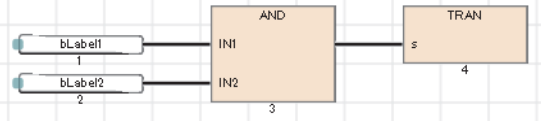
■Precautions

Under the following conditions, a step cannot be specified as the destination of jump sequence.

- When a step at the position escaping from a simultaneous sequence is specified
- When a step at the position entering a simultaneous sequence is specified
- When a step immediately before the preceding transition is specified
- When current step is specified

Detailed expression of transitions

Create detailed expression of transitions in the Zoom editor. The condition can be created in following programming languages.

Type		Description
Ladder diagram	Detailed expression	<p>Used to create a transition program consisting of a contact circuit and the TRAN instruction (Creating a dummy transition condition) in a single circuit block. The transition becomes TRUE when the TRAN instruction is executed.</p>  <p>■Restrictions</p> <ul style="list-style-type: none">• Inline ST cannot be used.• Only a TRAN instruction can be input to the coil.
	MELSAP-L (instruction format)	<p>📖 Page 115 Transition in MELSAP-L (instruction format)</p>
Structured text language		<p>Used to create the following transition program.</p> <p>■Method of writing a TRAN function (Creating a dummy transition condition) call statement</p> <pre>TRAN(bLabel1 & bLabel2);</pre> <p>//The transition becomes TRUE when the Boolean expression of the input argument is true.</p> <p>■Method of writing an assignment statement of Boolean expression for reserved word "TRAN"</p> <pre>TRAN := bLabel1 & bLabel2;</pre> <p>//The transition becomes TRUE when the Boolean expression of the right-hand side is true.</p> <p>■Method of writing an assignment statement of Boolean expression for the transition name</p> <pre>Transition1 := bLabel1 & bLabel2;</pre> <p>//Transition1 indicates the transition name input on the SFC editor. The transition becomes TRUE. when the Boolean expression of the right-hand side is true.</p>
FBD/LD		<p>Used to create a transition program ending with the TRAN instruction (Creating a dummy transition condition) in a single network.</p>  <p>■Restrictions</p> <ul style="list-style-type: none">• Only one TRAN instruction can be used.• A program to be assigned to the device/label cannot be created.• Coil, function block, function (except some), jump, jump label, and return program elements cannot be used. <p>For the available instructions other than TRAN, refer to the following.</p> <p>📖 Page 114 Detailed expression of transitions</p>

Point

- The detailed expression of the same transition can be used for multiple transitions.
- The created detailed expression of a transition can be checked from the Zoom list. (📖 GX Works3 Operating Manual)

■Usable instructions

The following table lists the instructions that can be used in transition programs.

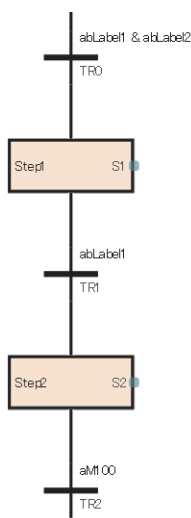
Classification	Instruction symbol
Contact instruction	LD, LDI, AND, ANI, OR, ORI
	LDP, LDF, ANDP, ANDF, ORP, ORF
	LDPI, LDFI, ANDPI, ANDFI, ORPI, ORFI* ²
Association instruction	ANB, ORB
	INV
	MEP, MEF
	EGP, EGF* ¹
Comparison operation instruction	LD□, LD□_U, AND□, AND□_U, OR□, OR□_U
	LDD□, LDD□_U, ANDD□, ANDD□_U, ORD□, ORD□_U
Real number instruction	LDE□, ANDE□, ORE□
	LDED□, ANDED□, ORED□
Character string processing instruction	LD\$□, AND\$□, OR\$□
Creating a dummy transition condition	TRAN

*1 The EGP and EGF instructions cannot be used in a transition program created in ST or FBD/LD.

*2 The LDPI, LDFI, ANDPI, ANDFI, ORPI, ORFI, and TRAN instructions cannot be used in a transition program created in MELSAP-L (instruction format).

Transition in MELSAP-L (instruction format)

In MELSAP-L (instruction format), transitions are described in text format in a SFC diagram.



Point

To switch from detailed expression in ladder diagrams to MELSAP-L (instruction format), select [View] ⇒ [Switch Ladder Display] ⇒ [MELSAP-L (Instruction Format)] from the menu. (GX Works3 Operating Manual)

In MELSAP-L (instruction format), transitions are described using the instructions corresponding to contacts. If a Boolean expression of a transition is TRUE, the transition becomes TRUE.

In MELSAP-L (instruction format), programs are described in the following format.

□: applicable label/device

Item	MELSAP-L (instruction format)	Example
Normally open contact (LD instruction)	a□	aX0
Normally closed contact (LDI instruction)	b□	bX1
Rising edge pulse (LDP instruction)	p□	pM2
Falling edge pulse (LDF instruction)	f□	fM3
Inverting the operation result (INV instruction)	& INV	aM0 & INV
Converting the operation result into a pulse (rising edge) (MEP instruction)	& MEP	aM1 & MEP
Converting the operation result into a pulse (falling edge) (MEF instruction)	& MEF	aM2 & MEF
Converting the edge relay operation result into a pulse (rising edge) (EGP instruction)	& EGP □	aM3 & EGP V0
Converting the edge relay operation result into a pulse (falling edge) (EGF instruction)	& EGF □	aM4 & EGF V1

Item	MELSAP-L (instruction format)	Example
Comparison operation instruction corresponding to contacts	Describe instructions in the same way as in ladder diagrams. The following comparison operation instructions can be used. Comparing 16-bit binary data (signed): <, <=, <>, =, >, >= Comparing 32-bit binary data (signed): D<, D<=, D<>, D=, D>, D>= Comparing single-precision real numbers: E<, E<=, E<>, E=, E>, E>= Comparing double-precision real numbers: ED<, ED<=, ED<>, ED=, ED>, ED>= Comparing 16-bit binary data (unsigned): <_U, <=_U, <>_U, =_U, >_U, >=_U Comparing 32-bit binary data (unsigned): D<_U, D<=_U, D<>_U, D=_U, D>_U, D>=_U Comparing string data: \$<, \$<=, \$<>, \$=, \$>, \$>=	< D10 D20
Parallel connection (OR)		aX0 aM0
Series connection (AND)	&	aX0 & aM0
Parentheses	()	(aX0 aM0) & aX1

When "&" and "|" are used in a single expression, "&" takes a priority. Use "(")" to alter priorities.

Direct expression of transitions

The transition which transfers an active state to the next step can be created directly on the SFC diagram. A contact of FBD/LD element is connected to it.



Coil, function block, function, jump, jump label, and return elements cannot be used.

Point

Select a transition and select [Edit] ⇒ [Modify] ⇒ [Direct Expression for Transition] from the menu. This can connect the FBD/LD element to the left side of the transition. (GX Works3 Operating Manual)

Transition label/device

Bit type label, bit device or Boolean value can be specified as a condition which transfer an active state to the next step.

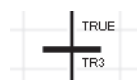
Bit type label



Bit device



Boolean value



Point

Select a transition name, select [Edit] ⇒ [Modify] ⇒ [Name] from the menu, and input the bit type label, bit device, or Boolean value to be specified. (GX Works3 Operating Manual)

Precautions

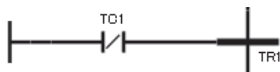
- When a device (T, ST, LT, LST, C, LC) of timer or counter is used as a transition, the device operates as a contact (TS, STS, LTS, LSTS, CS, LCS). Also, when a coil (TC, STC, LTC, LSTC, CC, LCC) of timer or counter is used, the coil operates as a contact.
- To use a coil of timer or counter for transition, use a timer type or counter type label.

Ex.

Timer device and timer type label



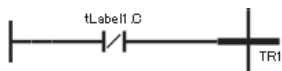
When the contact (TS0) is on, the transition becomes TRUE.



When the contact (TS1) is off, the transition becomes TRUE.



When the coil of the timer type label (tLabel0) is on, the transition becomes TRUE.



When the coil of the timer type label (tLabel1) is off, the transition becomes TRUE.

8.3 SFC Control Instructions

SFC control instructions are used to check a block or step operation status (active/inactive), or to execute a forced start, end or others. If SFC control instructions are used, SFC programs can be controlled from the actions of sequence programs and SFC programs.

Instruction List

The following table lists the SFC control instructions.

Instruction name	Instruction symbol	Processing
Checking the status of a step	LD, LDI, AND, ANI, OR, ORI [S□] ^{*1}	Checks whether a specified step is active or inactive.
	LD, LDI, AND, ANI, OR, ORI [BL□\S□]	
Checking the status of a block	LD, LDI, AND, ANI, OR, ORI [BL□]	Checks whether a specified block is active or inactive.
Batch-reading the status of steps	MOV(P) [K4S□] ^{*1}	Batch-reads (in units of 16-bit binary data) the status (active or inactive) of steps in a specified block, and stores the read data in a specified device.
	MOV(P) [BL□\K4S□]	
	DMOV(P) [K8S□] ^{*1}	Batch-reads (in units of 32-bit binary data) the status (active or inactive) of steps in a specified block, and stores the read data in a specified device.
	DMOV(P) [BL□\K8S□]	
	BMOV(P) [K4S□] ^{*1}	Batch-reads (in units of the specified number of words starting from a specified step) the status (active or inactive) of steps in a specified block.
	BMOV(P) [BL□\K4S□]	
Starting a block	SET [BL□]	Activates a specified block and executes a step sequence starting from an initial step.
Ending a block	RST [BL□]	Deactivates a specified block.
Pausing a block	PAUSE [BL□]	Temporarily stops a step sequence in a specified block.
Restarting a block	RSTART [BL□]	Releases the temporary stop and restarts the sequence from the step where the sequence was stopped in the specified block.
Activating a step	SET [S□] ^{*1}	Activates a specified step.
	SET [BL□\S□]	
Deactivating a step	RST [S□] ^{*1}	Deactivates the specified step.
	RST [BL□\S□]	
Switching a block	BRSET	Specifies a target block No. of SFC control instruction.

^{*1} When using in a sequence program, block 0 is the target block. When using in a SFC program, current block is the target block. For details on the SFC control instructions, refer to the following.

 MELSEC iQ-R Programming Manual (Instructions, Standard Functions/Function Blocks)

■Precautions

- Do not use the SFC control instructions in interrupt programs.
- Execute the SFC control instruction only when SM321 (SFC program start/stop) is on.

Index modification

The step relays and SFC block devices specified by SFC control instructions can be index-modified.

Device	Index modification target part
S□□	Step relay
BL□\S□□	Step of step relay with block specification
BL□□\S□	Block of step relay with block specification
BL□□\S□□	Block and step of step relay with block specification
BL□□	SFC block device

The step relays and SFC block devices can be specified within the following range, including the case of index modification.

Device	Range
S□	0 to 16383 (the maximum value is set by a CPU parameter)
BL□\S□	BL□ 0 to 319
	S□ 0 to 511
BL□	0 to 319



For details on index modification, refer to the following.

MELSEC iQ-R CPU Module User's Manual (Application)

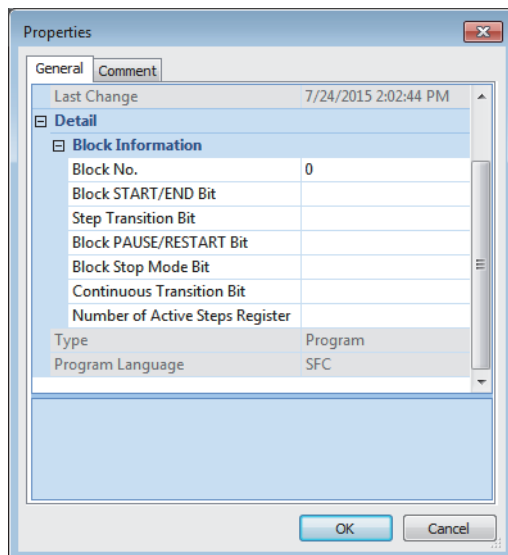
8.4 SFC Information Devices

SFC information device is the device or label which operates the forced start/termination and pause/restart direction to a block, check of the status of transition and the number of active steps, or direction of continuous transition operation of a transition.

SFC information device is set every blocks.

 [Navigation window] ⇒ [Program] ⇒ SFC program file⇒properties of block to be set

Window



Displayed items

Item	Description	Available data	
		Device	Data type (label)
Block START/END Bit	Sets the device or label to check whether the block is active. Setting the bit to on can start the block and setting it to off can end the block.	Bit: Y, M, L, F, V, B Word: Bit specification of D, W, RD	Bool, Boolean array, INT bit specification, Word bit specification
Step Transition Bit	Sets the device or label to check whether the transition of the step being executed becomes TRUE. This bit turns on when the transition to the next step becomes TRUE after execution of the action of each step.		
Block PAUSE/RESTART Bit	Sets the device or label to pause or restart an active block. Setting the bit to on stops the block at the step in execution and setting it to off restarts executing the block from the step where the block was stopped previously.		
Block Stop Mode Bit	Sets the device or label that decides the timing for stopping a block. Setting the bit to on stops the block after transition of each step and setting it to off stops all steps immediately.		
Continuous Transition Bit	Sets the device or label that decides the continuous transition action when the transition becomes TRUE. Setting the bit to on enables continuous transition and accordingly the action of the next step is executed in the same scan. Setting the bit to OFF disables continuous transition and accordingly one step is executed every scan.		
Number of Active Steps Register	Sets the device or label in which the number of currently active steps of a block is to be stored.	D, W, R, ZR, RD	INT, WORD

Not only global devices and local devices but also global labels or local labels can be specified for SFC information devices. Indirect specification, digit specification, and index modification (Z, LZ) cannot be performed.

Point

The settings of SFC information device are required only when an SFC information device is used. If such device is not used, the settings of SFC information device are not required.

Block START/END bit

This bit is a device or label to check whether the block is active.

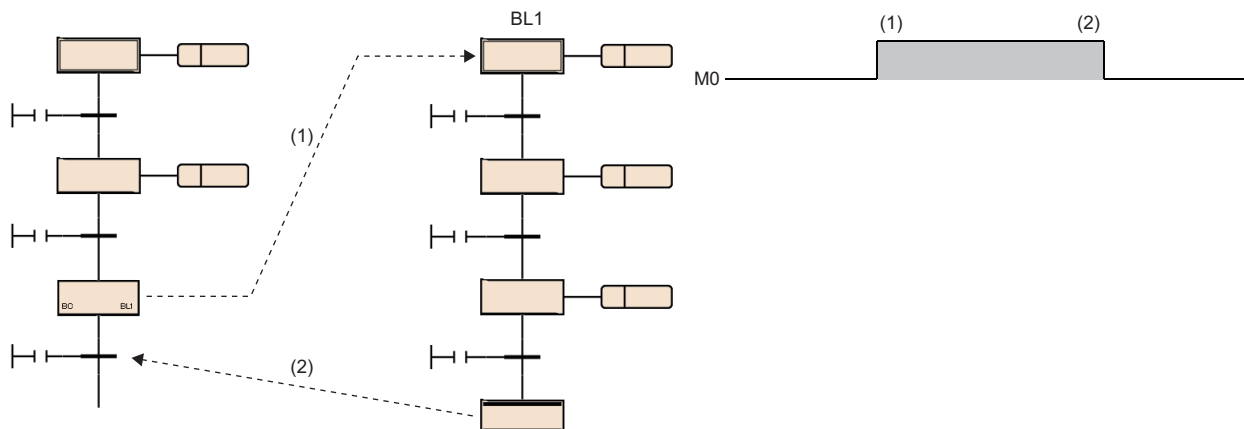
Setting the bit to on can start the block and setting it to off can end the block.

If a program to start a block is not available or because the START/END of a block can also be controlled from the engineering tool, this device or label can be used for debugging or test operation in units of block.

- When the set block starts, the block START/END bit is automatically turned on. While the set block is active, the block START/END bit stays on.
- When the set block becomes inactive, the block START/END bit is automatically turned off. While the set block is inactive, the block START/END bit stays off.

Ex.

M0 is specified in the block START/END bit of Block 1 (BL1).



(1) Block 1 (BL1) starts and M0 turns on.

(2) Block 1 becomes inactive and M0 turns off.

- When the block START/END bit is turned on while the set block is inactive, the block is started independently.
- When the block START/END bit is turned off while the set block is active, the block is ended.

The block START/END bit can also be turned on or off by the test operation of the engineering tool. (GX Works3 Operating Manual)

When the block START/END bit is turned off to make the set block inactive, processing will occur as follows:

- Execution of the set block is stopped and the outputs of the step being executed are all turned off. However, the devices turned on by using the SET instruction will not be turned off.
- If another block has been started by a block start step in the set block, the set block ends but the start destination block will remain active and continue processing.

Point

By changing the current value of BL□ or BL□/S□ from watch window of the engineering tool, the status of a block (START/END) or a step (active/inactive) can be changed.

Also, the status of the specified step (active/inactive) is changed from the menu [Debug] ⇒ [Control SFC Steps]. (GX Works3 Operating Manual)

■Precautions

- The following table shows the restart operation after the set block is deactivated.

Set block		Description
Block 0	When the start conditions setting of is "Auto-start block 0" in the SFC setting of the CPU parameter.	Operation is restarted from the initial step following end step processing.
	When the start conditions setting of is "Do not auto-start block 0" in the SFC setting of the CPU parameter.	The block is deactivated after end step processing, and processing is restarted from the initial step when another start request occurs for that block.
Other than block 0		

- When the SFC program ends, all block START/END bits that have been set in the SFC information devices are turned off. However, only when a resume start is enabled with the resume start setting, all block START/END bits are restored when the SFC program starts.

Step transition bit

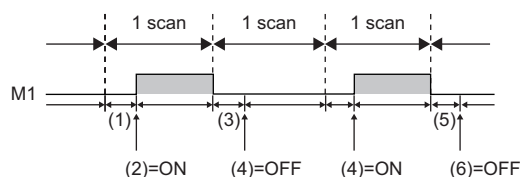
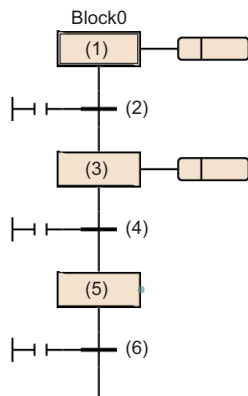
This bit is a device or label to check whether the transition of the step being executed becomes TRUE.

This bit turns on when the transition to the next step becomes TRUE after execution of the action of each step.

A step transition bit which is on is automatically turned off when processing of the specified block is performed again.

Ex.

M1 is specified in the step transition bit of Block0



If transition (2) becomes TRUE after execution of step (1), M1 is on during execution of another block.

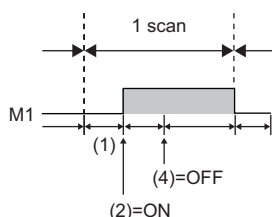
M1 is turned off at the time of Block0 processing in the next scan.

If transition (4) does not become TRUE after execution of step (3), M1 stays off.

If transition (4) becomes TRUE, M1 is on during execution of another block.

If transition (6) does not become TRUE after execution of step (5), M1 stays off.

If the continuous transition bit is turned on and set to "Continuous transition", the step transition bit will remain on during the action of the next step after the transition becomes TRUE. It will also remain on following the execution of multiple steps, even if the transition becomes FALSE. In these cases, the step transition bit will be turned off when the specified block is executed in the next scan.



If transition (2) becomes TRUE after execution of step (1), M1 is turned on.

Even if transition (4) does not become TRUE, M1 stays on.

M1 is turned off at the time of Block0 processing in the next scan.

When multiple active steps exist in the block, the step transition bit turns on when one of the transition becomes TRUE.

■Precautions

- When the end step is executed, the step transition bit of the block is turned on. The step transition bit remains on until the block is reactivated next.
- The step transition bit is not turned off when the SFC program starts or ends.


Block PAUSE/RESTART bit

This bit is a device or label to pause or restart an active block.

Setting the bit to on stops the block at the step in execution and setting it to off restarts executing the block from the step where the block was stopped previously.

Setting	Description
OFF→ON	When this bit is turned on, the specified block stops at the step being executed.
ON→OFF	When this bit is turned off, the specified block restarts execution from the action of the step that has been stopped previously. <ul style="list-style-type: none">• An operation HOLD step (without transition check) [SE] or an operation HOLD step (with transition check) [ST] which has been stopped in operation hold state is restarted with the state in effect.• The coil HOLD step [SC] cannot be restarted in hold state since the step is deactivated when it stops with the coil output is set to off (SM325 is off). If the step stops with the coil output hold setting (SM325 is on), it keeps the hold state even after it restarts.

- If another block has been started by a block start step, turning on the block PAUSE/RESTART bit stops the specified block, but the start destination block will remain active and continue processing. To stop the start destination block at the same time, the start destination's block PAUSE/RESTART bit must also be turned off.
- When the block PAUSE/RESTART bit specified in an inactive block is turned on, the block does not operate in inactive state and is put in the stopped state immediately when it becomes active.
- Even after the specified block is forcibly terminated, the state of the block PAUSE/RESTART bit remains held. If the block is forcibly terminated while it is stopped and the status of the block PAUSE/RESTART bit is not changed, the block is put in stopped state immediately after the restart.

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. ( Page 132 Operation when the block is paused or restarted)

■Precautions


- The block PAUSE/RESTART bit is not turned off when the SFC program starts or ends.

Block stop mode bit

This bit is a device or label that determines the timing for stopping a block.

Setting the bit to on stops the block after transition of each step and setting it to off stops all steps immediately.

Setting	Description
Off (immediate stop)	The block is put in stopped state immediately when a stop request is issued.
On (stop after transition)	When a stop request is issued, the block is stopped after the transition for the step being executed becomes TRUE and a transition occurs. The action of the step is not executed after the transition. When the block has multiple active steps, the steps are stopped in order from the one for which the transition becomes TRUE. A step that holds an operation stops immediately after a stop request is issued regardless of the setting of the block stop mode bit.

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. ( Page 132 Operation when the block is paused or restarted)

■Precautions

- The block stop mode bit is not turned off when the SFC program starts or ends.

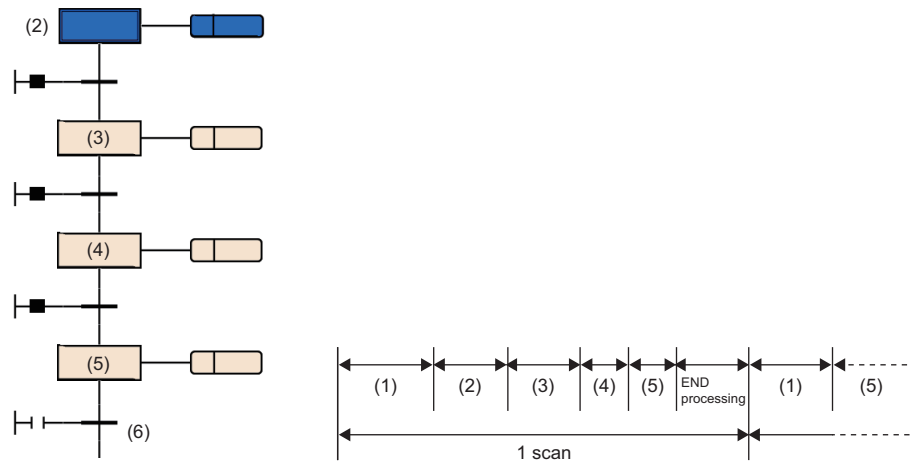
Continuous transition bit

This bit is a device or label that determines the continuous transition action when the transition becomes TRUE. Setting the bit to on enables continuous transition and accordingly the action of the next step is executed in the same scan. Setting the bit to OFF disables continuous transition and accordingly one step is executed every scan.

Setting	Description
Off (no continuous transition)	When the transition becomes TRUE, the action of the transition destination step is executed in the next scan.
On (continuous transition)	When the transition becomes TRUE, the action of the transition destination step is executed within the same scan. When the transition of the steps become TRUE continuously, the actions are executed within the same scan until the transition becomes FALSE or reaches the end step.

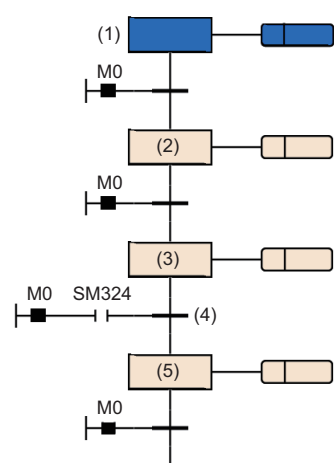
Ex.

The continuous transition bit of an SFC information device is specified.



Scan	Description
Scan 1	After execution of the sequence program (1), steps (2) to (5) of the SFC program are executed continuously.
Scan 2 and after	After execution of the sequence program (1), the action of step (2) is executed until the transition (6) becomes TRUE.

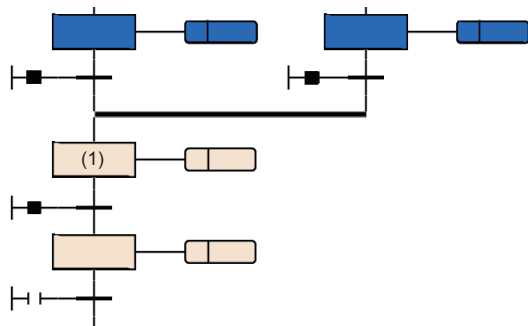
- When the continuous transition bit is set, a continuous transition is disabled while the set bit device is off and is enabled when the bit device is on, regardless of the on/off state of SM323 (All-blocks continuous transition status). When the continuous transition bit is not set, a continuous transition is disabled while SM323 is off and is enabled when it is on. (Page 139 Continuous transition ON/OFF operation)
- SM324 (Continuous transition disable flag) is turned on automatically by the system at SFC program execution, but is off during continuous transition. Use of SM324 under the AND condition in a transition disables a continuous transition.



When M0 is on, one scan causes continuous transitions from steps (1) to (3). Since SM324 is added as the AND condition to the transition (4), the transition (4) after execution of step (3) does not become TRUE. In the next scan, SM324 is turned on after step (3) and therefore a transition to step (5) occurs within the scan.

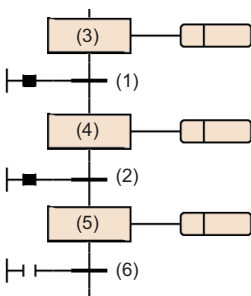
■Precautions

- If the continuous transition bit is turned on, execution of actions (from a transition becoming TRUE to destination step) takes priority over the other processing. allows to shorten a takt time. In this case, however, the operations of the other blocks and sequence program may become slower.
- The continuous transition bit is not turned off when the SFC program starts or ends.
- When a jump transition or selective convergence causes the active state to transition from multiple steps to one step, the action of one step may be executed twice in a single scan.



When the setting is "Continuous transition", step (1) is executed twice in a single scan.

- If the transition after the step becomes TRUE with the setting of "Continuous transition", a step is started or ended within one scan. In this case, since the END processing has not been executed, the input/output refresh of coil output by using the OUT instruction in the action is not reflected and therefore other programs cannot detect ON of the coil. In the case of output (Y), for example, output (Y) is not output while END processing is unexecuted and other programs cannot detect output (Y) ON. Accordingly, ON of the step relay cannot be detected, either. To reflect the I/O refresh of the OUT instruction, create a program so that one step is executed in multiple scans.



When the transition (1) and (2) become TRUE, the following is executed in one scan.

- The action of step (3) is executed.
- As the transition (1) becomes TRUE, the action of step (3) is turned off.
- Step (3) becomes inactive and step (4) becomes active.
- As continuous transitions are enabled, the action of step (4) is executed.
- As the transition (2) becomes TRUE, the action of step (4) is turned off.
- Step (4) becomes inactive and step (5) becomes active.
- As continuous transitions are enabled, the action of step (5) is executed.
- As the transition (6) does not become TRUE, the action of step (5) is not turned off.

- When creating a program that uses a jump sequence for looping, eliminate continuous transitions or prevent all transitions in the loop from becoming TRUE during execution. If all transitions in the loop become TRUE during execution with continuous transitions enabled, an infinite loop occurs in a single scan.

Number of Active Steps Register

This register is a device or label in which the number of active steps of a block is to be stored.

The number of active steps stored in the number of active steps register includes the following steps.

- Normal active step
- Coil HOLD step [SC] that holds the operation
- Operation HOLD step (with transition check) [ST] that holds the operation
- Operation HOLD step (without transition check) [SE] that holds the operation
- Steps that stop each operation

■Precautions

- When a block ends, the number of active steps register becomes 0.
- The register does not become 0 when the SFC program ends but becomes 0 when the program starts.

8.5 SFC Setting

Set start conditions and others of SFC program in CPU parameter or SFC block setting.

CPU parameter

The following table lists the SFC settings.

Type	Item	Description
SFC Setting	SFC Program Start Mode Setting	Set whether to start with initial status (Initial Start) or to start holding the previous execution status (Resume Start) at the start-up of SFC program.
	Start Conditions Setting	Set whether to automatically start and activate block 0 or to keep it inactive until a start request is issued, when starting the SFC program.
	Output Mode Setting at Block Stop	Set whether to turn off the coil output or to hold it when stopping a block.


Point

Keep the certain number of step relay (S) points before using the SFC program. (Default number of step relay (S) points is 0.)

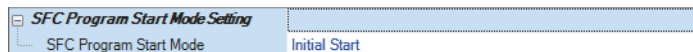
Set the number of step relay (S) points within the range of 1024 to 16384 points (in units of 1024 points) in [CPU Parameter] ⇒ [Memory/Device Setting] ⇒ [Device/Label Memory Area Setting] ⇒ [Device Setting].

SFC program start mode setting


Set whether to start with initial status (Initial Start) or to start holding the previous execution status (Resume Start) at the start-up of SFC program.

 [CPU Parameter]⇒[SFC Setting]⇒[SFC Program Start Mode Setting]

Window



Displayed items

Setting	Description
Initial Start (default)	The program is started after the active state at a previous stop is cleared. The operation after a start is performed according to the start condition setting of the SFC setting. ( Page 131 Start condition setting)
Resume Start	The program starts while holding the active state at a previous stop.

Whether to start an SFC program with initial status or the previous execution status is determined by the combination of the SFC program start mode setting and the SM322 (SFC program start mode) status.

Operation		SFC program start mode setting: Initial Start		SFC program start mode setting: Resume Start	
		SM322: OFF (Initial status) ^{*1}	SM322: ON (When the setting is changed)	SM322: ON (Initial status) ^{*1}	SM322: OFF (When the setting is changed)
(1)	SM321 is turned off and on.	Initial Start		Resume Start	Initial Start
(2)	CPU module is powered off and on.			Resume Start/Initial Start ^{*4}	
(3)	SM321 is turned on and off, or CPU module is powered off and on after changing the operating status from RUN to STOP.			Resume Start	
(4)	CPU module is reset and the operating status is changed to RUN.			Resume Start/Initial Start ^{*4}	
(5)	SM321 is turned on and off, or CPU module is reset and the operating status is changed to RUN after RUN to STOP.			Resume Start	
(6)	Operating status is changed from STOP to RUN.	Resume Start ^{*3}			
(7)	Operating status is STOP, write a program, and the status is changed to RUN.	Initial Start ^{*2}			

*1 The initial status of SM322 is determined when the operating status of the CPU module is changed from STOP to RUN according to the setting of the SFC program start mode.

*2 When the Resume Start is set for the SFC program start mode, a program is resumed unless there is any change before and after program writing.

*3 The on/off state of an action is determined according to the setting of "Output Mode at STOP to RUN" of parameter setting.

*4 Depending on the timing, a program cannot be resumed and starts with initial status.

■Precautions

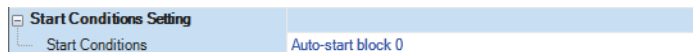
- When a program is resumed, the SFC program stop position is held but the status of the label or device used for an action is not held. Therefore, if labels or devices are required to be held to start with previous status, set them to be latched.
- When a program is resumed with conditions other than the ones ((1), (3), (5) in the table) where the coil output of the coil HOLD step [SC] is turned off, the coil HOLD step [SC] that holds the operation is restarted but the output is not turned on. To continue the output, set the labels and devices required to be held to be latched. The on/off state of the output at the time of changing STOP to RUN is determined according to the setting of "Output Mode Setting at STOP to RUN" of CPU parameter setting. (📖 MELSEC iQ-R CPU Module User's Manual (Application))
- At power-off or reset, the intelligent function module is initialized. To resume a program, creating an initial program for the intelligent function module in the block which is always active or in a sequence program is recommended.
- At power-off or reset, labels and devices are also cleared. When the SFC information device is set, the values are held only when latch setting is performed.
- Depending on the timing, a program may be resumed after power-off or reset. If a program is started with initial status while the start mode is set to Resume Start, an event indicates that a program cannot be resumed is stored in the event history. To resume a program without fail, turn off SM321 or switch the operating status of the CPU module from RUN to STOP, and then power off or reset the CPU module.

Start condition setting

Set whether to automatically start and activate block 0 or to keep it inactive until a start request is issued, when starting the SFC program.

 [CPU parameter]⇒[SFC Setting]⇒[Start Conditions Setting]

Window



Displayed items

Setting	Description	
	At SFC Program START	At the end of block 0
Auto-start block 0 (default)	Block 0 is started automatically and starts execution from its initial step.	Block 0 is restarted automatically and restarts execution from its initial step.
Do not auto-start block 0	Block 0 is activated by a start request resulting from the SET (Starting a block) instruction or a block start step, in the same manner as other blocks.	Block 0 is not restarted automatically and remains inactive until another start request is issued.

Use the start condition setting when it is desired to specify the start block at the start of SFC program according to the product type.

"Auto-start block 0" is useful when block 0 is used as described below.


- Management block
- Preprocessing block
- Continuous monitoring block

■Precautions

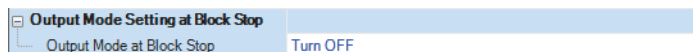
- To execute the SFC program when "Do not auto-start block 0" is set, execute the SET instruction (Starting a block) from the sequence program or turn on the block START/END bit that is set in the SFC information device.
- When "Auto-start block 0" is set, be sure to create block 0.

Output mode setting at block stop

Set whether to turn off the coil output or to hold it when stopping a block.

 [CPU parameter]⇒[SFC Setting]⇒[Output Mode Setting at Block Stop]

Window



Displayed items

Setting	Description
Turn OFF (default)	Coil output is turned off.
Keep ON	Coil output is held in the state immediately before stop.

- The settings made are reflected to the initial value of SM325 (Output mode at block stop) at power-on, reset, or switching from STOP to RUN, and follow the settings of SM325 when the SFC program operates. CPU parameter settings are ignored.

■Operation when the block is paused or restarted

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status.

The following table lists the operations at block PAUSE/RESTART.

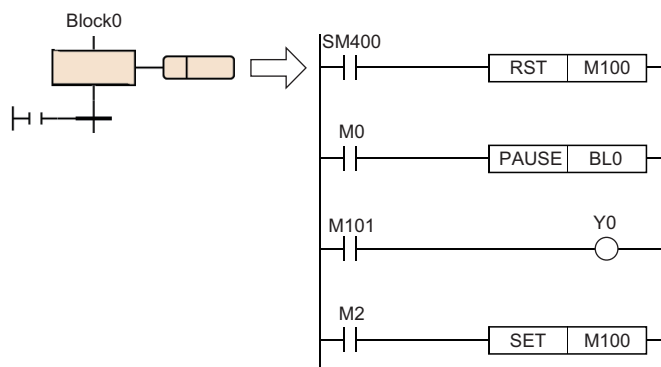
Output mode setting at block stop	Setting of block stop mode bit	Operation			
		Active step other than step that holds operation (including SC, SE, and ST whose transition does not become TRUE)	Step that holds operation		
			Coil HOLD step [SC]	Operation HOLD step (without transition check) [SE]	Operation HOLD step (with transition check) [ST]
SM325=OFF (coil output OFF)	OFF or no setting (immediate stop)	Immediately after a stop request is made, the coil output of the action is turned off and the block is stopped. The status remains active.	Immediately after a stop request is made, the coil output of the action is turned off and the block is deactivated.	Immediately after a stop request is made, the coil output of the action is turned off and the block is stopped. The status remains active.	
	On (stop after transition)	After the transition becomes TRUE, step end processing is performed and simultaneously the transition destination step becomes active and the block is stopped before execution of the action.			
SM325=ON (coil output held)	OFF or no setting (immediate stop)	Immediately after a stop request is made, the block is stopped with the coil output of the action being held. The status remains active.	Immediately after a stop request is made, the block is stopped with the coil output of the action being held. The status remains active.		
	On (stop after transition)	Normal operation is performed until the transition becomes TRUE. After the transition becomes TRUE, step end processing is performed and simultaneously the transition destination step becomes active and the block is stopped before execution of the action.			
At restart		Returns to normal operation.	Coil output is off: Becomes inactive and restart is disabled Coil output is held: Restarts with the hold state	Restarts the execution of the action in a HOLD status.	In the hold status, the action is restarted and the transition is also checked.

■Precautions

- When the block specified with the LD instruction (Checking the status of a block) has stopped, the coil output is turned on. Also, when the step specified with the LD instruction (Checking the status of a step) has stopped, the coil output is turned on.
- If the block is started while the PAUSE/RESTART bit of the SFC information device is on, the initial step stops before it becomes active. If the SET instruction (Activating a step) is executed for an inactive block, the specified step stops before it becomes active.
- When SM325 (Output mode at block step) is on, the block can be stopped while holding the coil output. Even when SM325 is turned on and off in stopped state, the state of the coil output does not change. When a block restart request is issued, the coil output restarts while keeping the hold state.
- If the block is stopped when SM325 is on, the coil HOLD step [SC] in the hold state keeps its state even after restart but the step operation does not restart. To make the coil HOLD step [SC] inactive, execute the RST instruction (Deactivating a step).
- When a stop request is issued in the action to the block, the step being executed currently is executed until it ends, and then the stop request is executed. Therefore, when the block stop mode bit is off (immediate stop), the step being executed does not stop even if a stop request is issued within the step. If the block stop mode bit is changed to on (stop after transition) afterwards in the same step, a stop request is executed in stop mode after transition.

Ex.

M100 is the block stop mode bit and M101 is the block PAUSE/RESART bit.



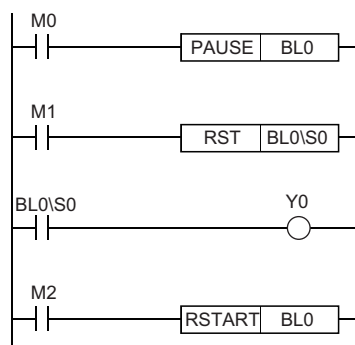
If M0 is turned on during execution of the above action, the PAUSE instruction is executed and the block PAUSE/RESTART bit (M101) of Block0 is turned on but the execution continues to the end of the action, so Y0 is turned on.

When M2 is on, the block stop mode bit is turned on even after execution of the PAUSE instruction and, after the actions are all executed, a stop request is executed in stop mode after transition.

- If the RST instruction (Deactivating a step) is executed while the block is stopped, the specified step relay is turned off. However, the monitor of the engineering tool keeps showing the active status and changes to inactive when the block is restarted. The same is also true when the instruction is executed while the block is stopped by turning on SM325 (holding the coil output when the block is stopped), but the coil output is not turned off.
- The SET instruction (Activating a step) is executed immediately even while the block is stopped and the specified step relay is turned on. The display on the monitor of the engineering tool also shows the active status. However, the action is executed only after the block is restarted.

Ex.

Block PAUSE/RESTART when the RST instruction (Deactivating a step) is used




- (1) When M0 is turned on, block 0 is stopped.
- (2) When M1 is turned on, a termination request is executed for step No. 0 and BL0\SO of the step relay is turned off but step No. 0 is left active on the monitor of the engineering tool.
- (3) BL0\SO is turned off, so Y0 is also turned off.
- (4) When M2 is turned on while M0 and M1 are off, block 0 restarts and step No. 0 ends.

- If the block stop mode bit (stop after transition mode) is turned off while a step in a state that is waiting to stop operation after transition exists, the step remains in that state. To immediately stop after clearing this state, restart the block and issue a stop request again while the block stop mode bit is off.
- When the step transition destination is an end step in stop after transition mode, end step processing is executed and therefore the step is not put in the stopped state.
- To check that a stop request has been issued, monitor the block list display of the engineering tool or monitor the bit that has been set in the block PAUSE/RESTART bit. However, whether the step is in stop state or operating to wait to stop cannot be checked from the monitor of the engineering tool.
- The stop after transition state can be cleared by turning off the block PAUSE/RESTART bit or executing the RESTART instruction before the transition becomes TRUE. If a restart request is issued while steps that is already stopped and steps that is waiting to stop operations coexist, the former starts and the latter continues the operation. The stop request is cleared.

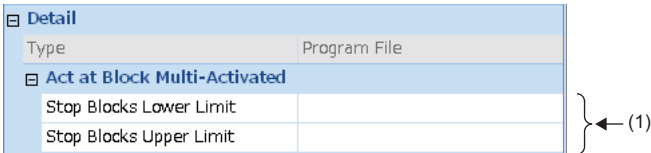
SFC block setting

Act at block multi-activated

Set the operation mode to stop the operation of the CPU module when a start request is issued by the block start step (with end check) [BC] or block start step (without end check) [BS] for an already active block. For the setting range, set the range of the block to be stopped.

 [Navigation window] ⇒ [Program] ⇒ Properties of SFC program file to be set

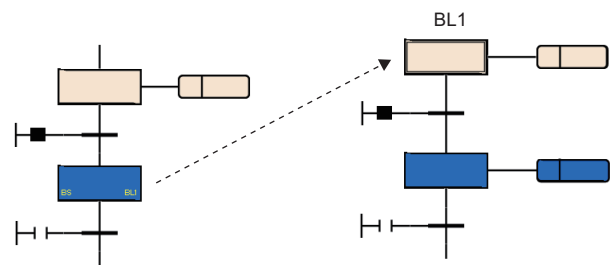
Window



(1) Set the range of the block to be stopped.

Displayed items

Setting	Description	
No setting (default)	Stand by	CPU module operation continues, and standby until the start destination block becomes inactive while the transition becomes TRUE. When the start destination block is deactivated, the block is reactivated. If a transition in standby state, the previous step is deactivated, the output is switched OFF, and the action will not be executed.
Block stop range is set.	Stop	An error results.



■Precautions

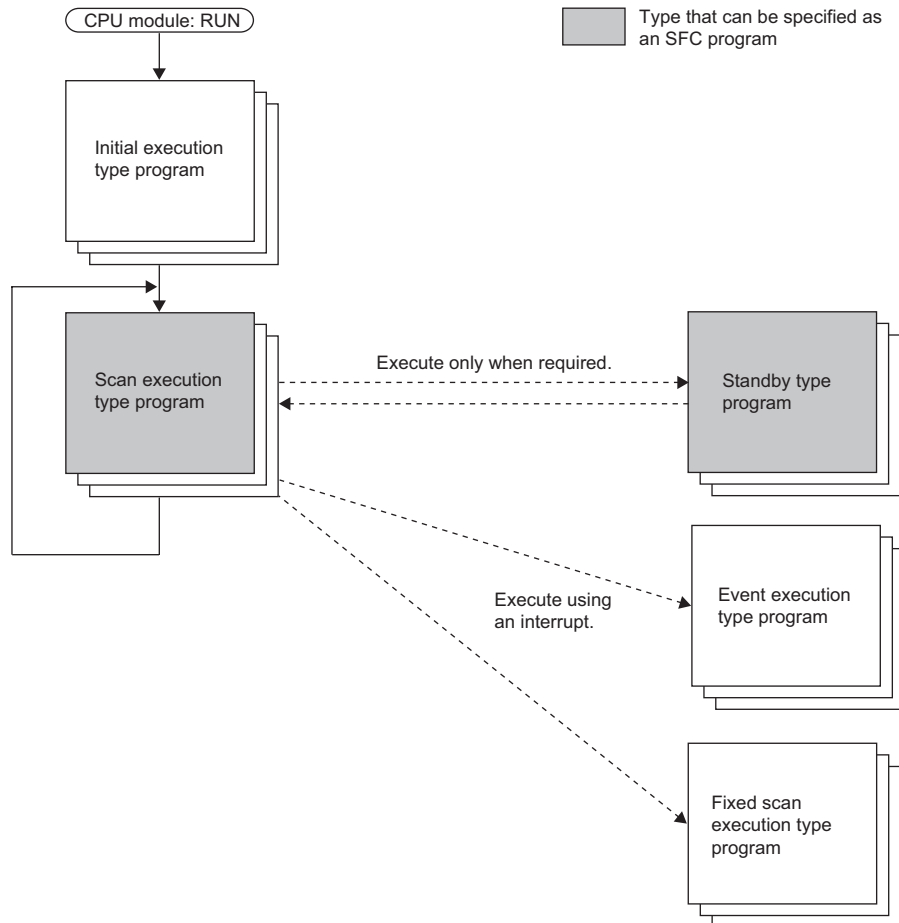
- When the SET instruction (Activating a block) is executed for the block that is already active, the start request is ignored and the processing of the SFC program is continued as is.
- If an attempt to transition to an active block start step is made, the activation of the block start step is ignored. The block is not executed again from the initial step.

8.6 SFC Program Execution Order

Whole program processing

Execution type that can be specified

This section shows whether the execution type of SFC program can be specified.



Execution Type	Specification enable/disable	Remarks
Initial execution type program	×	—
Scan execution type program	○	Only one SFC program can be executed.
Standby type program	○	By specifying the SFC program by using the PSCAN instruction, this type of program can be changed to the scan execution type.
Event execution type program	×	—
Fixed scan execution type program	×	—

■Precautions

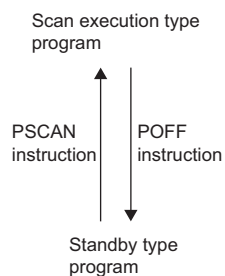
When no scan execution type SFC program exists (only standby type program), do not execute an SFC control instruction and monitoring for an SFC program.

Changing the execution type by an instruction

The execution type of a program can be changed by using a program control instruction.

The following table lists program control instructions with regard to whether an SFC program can be specified.

Instruction symbol	Specification enable/disable	Remarks
PSCAN	○	Changes the execution type of the specified SFC program to the scan execution type. If this instruction is executed with another SFC program specified while an scan execution type SFC program already exists, an error occurs.
PSTOP	×	If this instruction is executed for an SFC program, an error results.
POFF	○	Executes end processing of all blocks in the next scan and changes the execution type of the specified SFC program to the standby type in the following scan.



■Precautions

- Do not use the PSCAN instruction to read/write a file from/to the CPU module or use the data logging function. If the PSCAN instruction is executed, the scan time may extend several hundred milliseconds.
- When the SFC program, which is different program from the one operated last time, is operated by using the PSCAN instruction while specifying resume start, the specified SFC program performs the initial start. In this case, "SFC program continuous start not possible" (event code: 0430) is saved in the event history.

SFC program processing sequence

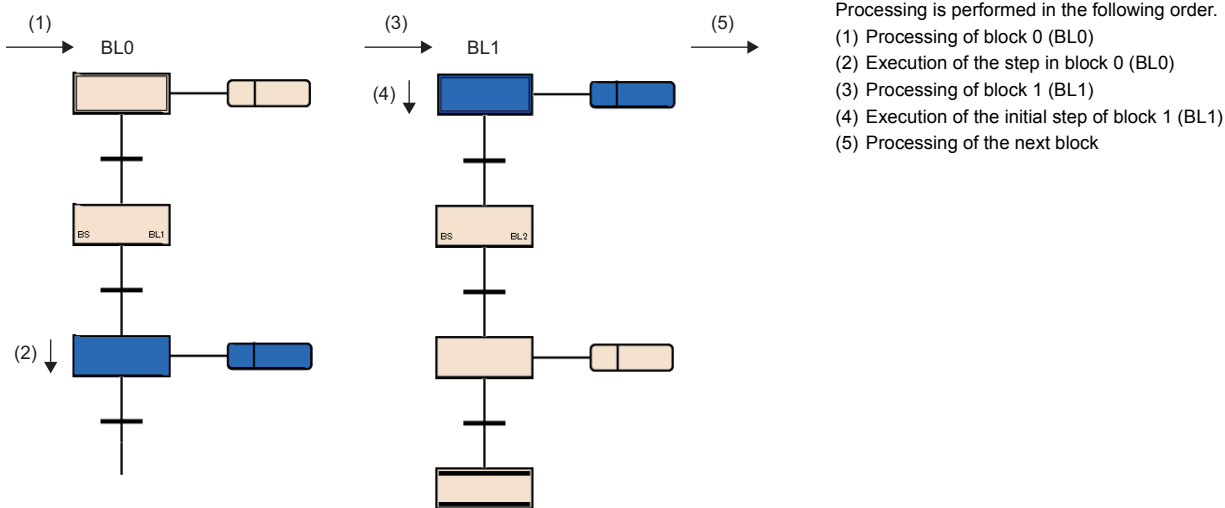
Block execution sequence

While the SFC program is running, the actions of each step are executed sequentially starting from the initial step of an active block.

An SFC program containing multiple blocks checks the state (active/inactive) of the blocks in ascending order of block numbers(block 0 → block 1 → block 2).

An active block executes the actions of active steps in the block.

An inactive block checks for existence of a start request. If a start request exists, the block is activated and the active steps in the block are executed.



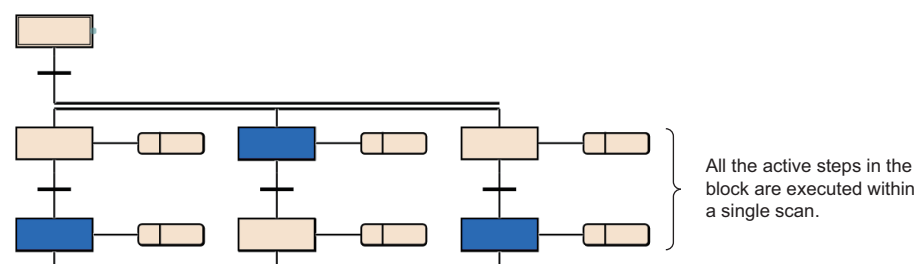
Only block 0 can be started automatically when the block 0 autostart is specified in the start condition setting of the SFC setting. With this setting, even when block 0 reaches the end step and becomes inactive, it is started again in the next scan.

(📖 Page 131 Start condition setting)

A request for END, PAUSE, RESTART of a block is processed immediately before execution processing in the block.

Step execution sequence

In the SFC program, the actions of all active steps are processed within one scan.

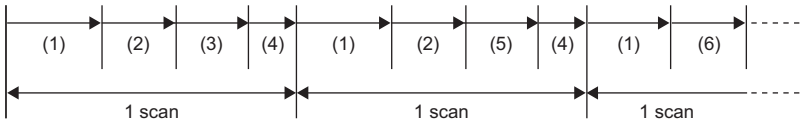


When the action of each step is finished, whether the transition to the next step becomes TRUE or not is checked.

- When the transition has not become TRUE: The action of the same step is executed again in the next scan.
- When the transition has become TRUE: The outputs of the executed actions by using the OUT instruction are all turned off. When the next scan is executed, the action of the next step is executed. The step executed previously is deactivated and the action becomes inactive.

Even when the transition becomes TRUE, if coil HOLD step [SC] is set in the step attribute, the step is not deactivated but performs processing according to the attribute. (☞ Page 96 Coil HOLD step [SC])

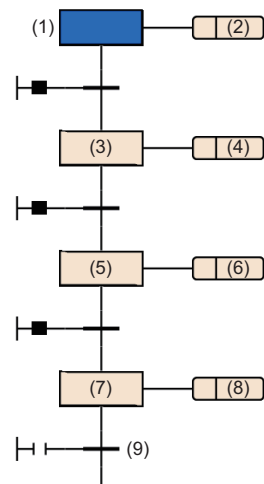
STOP→RUN
(SM321=ON)



- (1) Execution of sequence program
- (2) Execution of action
- (3) Checking the transition to the next step (FALSE)
- (4) END processing
- (5) Checking the transition to the next step (TRUE)
- (6) The next action is executed.

Ex.

The continuous transition bit of an SFC information device is not specified.

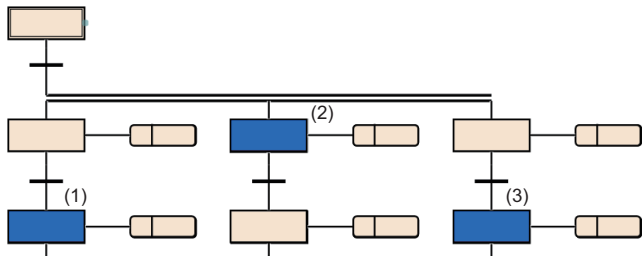


Scan	Description
Scan 1	Step (1) is activated and the action (2) is executed.
Scan 2	Step (3) is activated and the action (4) is executed.
Scan 3	Step (5) is activated and the action (6) is executed.
Scan 4	Step (7) is activated and the action (8) is executed.
Scan 5 and after	Step (7) is active until the transition (9) becomes TRUE, and the action (8) is executed.

■Precautions

- As a step for which the transition becomes TRUE at the first execution is deactivated in a single scan, the I/O refresh of coil output is not reflected and therefore other programs cannot detect that the coil output is on. To reflect the I/O refresh, create a program so that one step is executed in multiple scans.
- The actions of active steps in a block are executed simultaneously (within the same scan). For this reason, do not create SFC programs which depend on the execution sequence of actions.

The execution sequence of actions (1), (2), and (3) are undefined.



Continuous transition ON/OFF operation

There are two types of transitions in SFC program: "Continuous transition" and "No continuous transition". The setting of the type (Continuous transition or No continuous transition) is determined by the settings of the continuous transition bit of the SFC information device and SM323 (All-blocks continuous transition status).

Continuous transition bit	SM323	Description	
No setting	Off	No continuous transition	When the transition becomes TRUE, the action of the transition destination step is executed in the next scan.
	On	Continuous transition	When the transition becomes TRUE, the action of the transition destination step is executed within the same scan. When the transitions of the steps become TRUE continuously, the actions are executed within the same scan until the transition becomes FALSE or the end step is reached.
Off	On or off	No continuous transition	When the transition becomes TRUE, the action of the transition destination step is executed in the next scan.
On	On or off	Continuous transition	When the transition becomes TRUE, the action of the transition destination step is executed within the same scan. When the transitions of the steps become TRUE continuously, the actions are executed within the same scan until the transition becomes FALSE or the end step is reached.

Point

The tact time can be shortened by setting "Continuous transition". This resolves the problem of waiting time from when the transition becomes TRUE until the action of the transition destination step is executed. However, when "Continuous transition" is set, the operations of the other blocks and sequence program may become slower.

8.7 SFC Program Execution

Starting and stopping the SFC program

The SFC program can be started and stopped by either of the following methods.

- CPU parameter
- Starting and stopping the program by the special relay (SM321)
- Starting and stopping the program by using instructions

CPU parameter

Set "Auto-start block 0" to "Start Conditions Setting" in the CPU parameter. Block 0 of the SFC program starts automatically when the CPU module is powered on or reset, or the operating status is changed from STOP to RUN. (➡ Page 131 Start condition setting)

Starting and stopping the program by the special relay (SM321)

SM321 (SFC program start/stop) automatically turns on at execution of the SFC program. Once executed, the start/stop status is controlled by SM321.

- The program execution can be stopped by turning off SM321.
- The program execution can be restarted by turning on SM321.

Point

The resume start of the SFC program can be set in the CPU parameter ("SFC Program Start Mode Setting"). (➡ Page 129 SFC program start mode setting)

Starting and stopping the program by using instructions







The SFC program is started and stopped by using the program control instructions. (➡ Page 136 Changing the execution type by an instruction)

- The standby type SFC program is started by using the PSCAN instruction. The program execution type changes from the standby type to the scan execution type.
- Outputs are turned off and the SFC program is stopped by the POFF instruction. The program execution type changes from the scan execution type to the standby type.

Starting and ending a block





Starting a block

A block in the SFC program can be started by either of the following methods.

Item	Method	Remarks	Reference
CPU parameter (auto start, only for block 0)	Set "Auto-start block 0" to "Start Conditions Setting" in the CPU parameter. When the SFC program is executed, block 0 starts automatically and processing is performed sequentially from the initial step.	This method is used to use block 0 as a control block, preprocessing block, or continuous monitoring block.	 Page 131 Start condition setting
Block start step	Start another block by using a block start step [BC or BS] in a block.	This method is effective when the control sequence is clear.	 Page 98 Block start step (with END check) [BC]  Page 99 Block start step (without END check) [BS]
SFC control instruction	Start the block specified by the SFC control instruction used in the action of the SFC program or in another sequence program. <ul style="list-style-type: none"> Use the SET [BL□] (Starting a block) instruction to execute the program from the initial step of the specified block. Use the SET [S□/BL□\S□] (Activating a step) instruction to execute the program from the specified step of the specified block. 	This method is effective to restart the error processing block or execute interrupt processing.	 Page 118 SFC Control Instructions
SFC information device	Start the specified block by turning on the block START/END bit set to each block.	This method is effective for debugging (in units of blocks) and test operation because blocks can be restarted even from external devices.	 Page 121 Block START/END bit
Engineering tool	Start the specified block by turning on the SFC block device.	This method is effective for debugging and test operation.	 GX Works3 Operating Manual

Ending a block

A block in the SFC program can be ended by either of the following methods.

Item	Method	Remarks	Reference
End step	Execute the end step in a block. Processing is stopped and the block becomes inactive.	This method is effective to stop operation by stopping a cycle in automatic operation.	 Page 100 End step
SFC control instruction	End and deactivate the block specified by the RST [BL□] (Ending a block) instruction used in the action of the SFC program or in another sequence program. (The block ends when all the active steps in the specified block are deactivated by using the RST [BL□\S□] (Ending a block) instruction.)	This method is effective to end processing regardless of the operation status, such as an emergency stop.	 Page 118 SFC Control Instructions
SFC information device	End the specified block by turning off the block START/END bit set to each block.	This method is effective for debugging (in units of blocks) and test operation because blocks can be ended even from external devices.	 Page 121 Block START/END bit
Engineering tool	End the specified block by turning off the SFC block device.	This method is effective for debugging and test operation.	 GX Works3 Operating Manual

Pausing and restarting a block

Pausing a block

The specified block in the SFC program being executed can be paused by either of the following methods.

Item	Method	Remarks	Reference
SFC control instruction	Pause the block specified by the PAUSE [BL□] (Pausing a block) instruction used in the action of the SFC program or in another sequence program.	This method is effective to clear the error by temporarily stopping the machine and operating it manually.	☞ Page 118 SFC Control Instructions
SFC information device	Pause the specified block by turning on the block PAUSE/RESTART bit set to each block.	This method is effective for debugging and test operation because blocks can be paused even from external devices.	☞ Page 124 Block PAUSE/RESTART bit

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. (☞ Page 132 CPU parameter)

Restarting a block

The paused block in the SFC program can be restarted by either of the following methods.

Item	Method	Remarks	Reference
SFC control instruction	Restart the block specified by the RSTART [BL□] (Restarting a block) instruction used in the action of blocks other than the paused block in the SFC program or in another sequence program.	This method is effective in operating the machine automatically again after it is stopped temporarily and operated manually.	☞ Page 118 SFC Control Instructions
SFC information device	Restart the specified block by turning off the block PAUSE/RESTART bit set to each block.	This method is effective for debugging (in units of blocks) and test operation because blocks can be restarted even from external devices.	☞ Page 124 Block PAUSE/RESTART bit

Operation when the block is paused or restart depends on the combination of the SM325 (Output mode at block stop) status, block stop mode bit setting of the SFC information device, and step hold status. (☞ Page 132 CPU parameter)

Activating and deactivating a step

Activating a step

A step in the SFC program can be activated by either of the following methods.

Item	Method	Remarks	Reference
Transition condition	The transition is checked at the end of the step. If it is TRUE, the next step is automatically activated.	—	Page 107 Transition
SFC control instruction	Activate the step specified by the SET [S□/BL□\S□] (Activating a step) instruction used in the action of the SFC program or in another sequence program.	—	Page 118 SFC Control Instructions
Engineering tool	<ul style="list-style-type: none">• Activate the specified step by turning on the step relay.• Activate the selected step from the menu [Debug] ⇒ [Control SFC Steps].	This method is effective for debugging and test operation.	GX Works3 Operating Manual

Deactivating a step

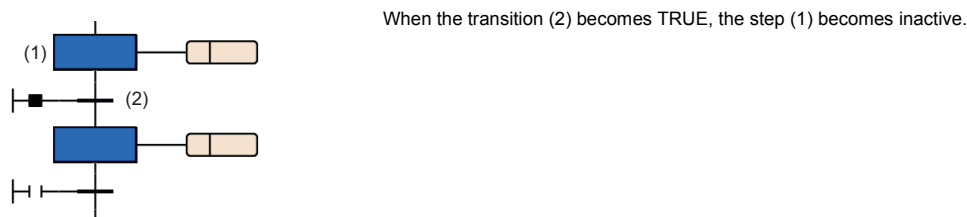
A step in the SFC program can be deactivated by either of the following methods.

Item	Method	Remarks	Reference
Transition condition	The transition is checked at the end of the step. If it is TRUE, the current step is automatically deactivated.	—	Page 107 Transition
Reset step [R]	Activating this step deactivates the step specified for attribute target.	This method is effective to deactivate HOLD steps [SC, SE, ST] when the sequence for error processing is selected in the selection branch.	Page 98 Reset step [R]
SFC control instruction	Deactivate the step specified by the RST [S□/BL□\S□] (Deactivating a step) instruction used in the action of the SFC program or in another sequence program.	When all the active steps in the specified block are deactivated by using the RST instruction, the block also ends.	Page 118 SFC Control Instructions
Engineering tool	<ul style="list-style-type: none">• Deactivate the specified step by turning off the step relay.• Deactivate the selected step from the menu [Debug] ⇒ [Control SFC Steps].	This method is effective for debugging and test operation.	GX Works3 Operating Manual

Behavior when an active step is activated

When an active step is activated, the step behaves as follows.

Series sequence



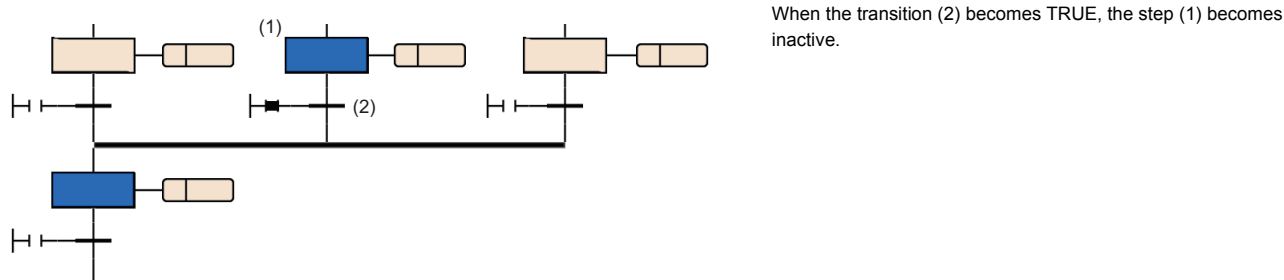
Selective sequence

■Divergence

Transitions are checked from left to right. If the step connected to the transition having a TRUE value is active, the steps behave in the same way as in the series sequence. After the first TRUE path is taken, transitions are no longer checked.

■Convergence

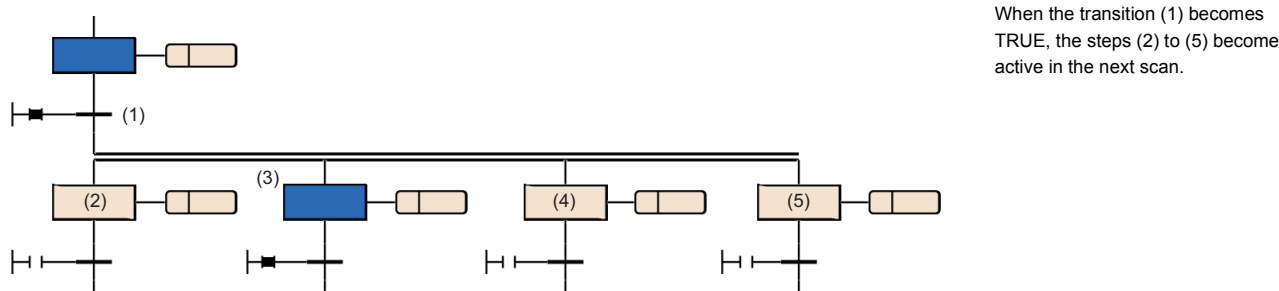
The steps behave in the same way as in the series sequence.



Simultaneous sequence

■Divergence

If any one of the steps in divergence of the simultaneous sequence is active, all steps below the transition become active in the next scan.



■Convergence

All steps above the transition become inactive. The HOLD steps [SC, SE, ST] hold operations.

Operation when a program is modified

When an SFC program is modified by writing data to the programmable controller or online change, the SFC program operates as follows.

When data is written to the programmable controller

The SFC program operates as follows.

SM322 (SFC program start mode)	Program modification status	
	Modified	Not modified
Off: Initial start	Initial start	Initial start
On: Continue start	Initial start	Continue start

Values of devices and labels used in the SFC program will be as follows depending on the setting of SM326 (SFC device/label clear mode).

SM326 (SFC device/label clear mode)*1	Description
Off	Values of devices and labels, excluding the following, are cleared, and the SFC program is executed. <ul style="list-style-type: none">• Step relay (S)• File register (R/ZR)*2• Latched labels
On	Values of devices and labels, excluding the step relay (S), are held, and the SFC program is executed.

*1 The setting of SM326 is valid only when an SFC program exists after data is written to the programmable controller. It is also valid when a program file or parameter file is written to the programmable controller. It will be invalid when only the common device comment file, device memory file, or device initial value file is written.

*2 Even when the device is not latched, data is not cleared.

■Precautions

- After an SFC program is modified by writing data to the programmable controller, reset the CPU module, and execute the SFC program.
- If “Resume Start” is set to “SFC Program Start Mode Setting” in the CPU parameter, turn off SM322 (SFC program start mode) first, and modify the program by writing data to the programmable controller. Thereafter, initial-start the SFC program and then turn on SM322 (resume start) again.

When online change is executed

After an SFC program is modified by online change, the resume start is performed regardless of the CPU parameter setting (“SFC Program Start Mode Setting”).

When the operating status is changed from STOP to RUN

If the operating status of the CPU module is changed from RUN to STOP during execution of the SFC program, the device values and active/inactive state of the SFC program immediately before the stop are held and restored after the operating status is changed back to RUN. the resume start is performed regardless of the CPU parameter setting (“SFC Program Start Mode Setting”).

If any of the sequence program file (including an SFC program), FB file, or parameter file (such as CPU parameter file and system parameter file) is written to the CPU module while it is in the STOP state, the SFC program will be executed initially when the operating status is changed back to RUN. Note that the resume start may be performed if there is no change in the SFC program after the program file is written. (👉 Page 129 SFC program start mode setting)


Checking SFC program operation

Use the following functions of the engineering tool to check SFC program operation.

- Monitor
- Watch
- Device/buffer memory batch monitor
- Control SFC steps
- SFC block list
- SFC all blocks batch monitor
- Active step monitor



For details on each functions and operation check methods, refer to the following.

 GX Works3 Operating Manual

INDEX

Symbols

-	60
*	60
**	60
/	60
&	60
+	60
<	60
<=	60
<>	60
=	60
>	60
>=	60
\$	47

A

AND	60
ASCII	39,70,82
Assignment statement	61

B

BC	93
Bit	39
Block PAUSE/RESTART Bit	120
Block start step (with END check)	93
Block start step (without END check)	93
Block START/END Bit	120
Block Stop Mode Bit	120
BOOL	39
BS	93
Buffer memory	8

C

CASE	64
Class	41
Coil HOLD step	93
Constants	47
Continuous Transition Bit	120
Conversion of data type	62,75
COUNTER	39,40
Counter	39,40

D

Data type	39,40,41
Detailed expression	107
Device	8
Device assignment	37
DINT	39
Direct expression	107
Double word [signed]	39
Double word [unsigned]/bit string [32 bits]	39
Double-precision real number	39
DWORD	39

E

EN	16,26
----	-------

End step	93
ENO	16,26
EXIT	65
External variable	21

F

FB/FUN file	16,17,26,27
FBD/LD	9
FOR...DO	65,69
Function (FUN)	13,15,38
Function block (FB)	13,20,38
Function block call statement	63
Function call statement	63

G

Generic data type (ANY type)	41
Global label	37,38,41

I

IF THEN	64
IF...ELSE	64
IF...ELSEIF	64
Initial step	93
Input variable	15,21
Input/output variable	21
Instance	22
INT	39
Internal variable	21
Interrupt program	14

J

Jump sequence	108
---------------	-----

L

Label	8
Label/device	107
Ladder diagram	9,53
LCOUNTER	39,40
Local label	37,38,41
Long counter	39,40
Long retentive timer	39,40
Long timer	39,40
LREAL	39
LRETENTIVETIMER	39,40
LTIMER	39,40

M

Macro type function block	27,28
Main routine program	14
MOD	60
Module label	37

N

Normal step	93
-------------	----

NOT	60
Note	57
Number of Active Steps Register	120
Number of array elements	43
Number of steps	18

O

Operation HOLD step (with transition check)	93
Operation HOLD step (without transition check)	93
OR	60
Output Mode Setting at Block Stop	128
Output variable	15,21

P

POINTER.	39
Pointer.	39
Program.	11,16,26
Program block	14,38
Program file	11
Programming language	9
Project	11

R

R.	93
REAL.	39
REPEAT...UNTIL	65
Reserved word	59
Reset step	93
Retentive timer	39,40
RETENTIVETIMER.	39,40
RETURN	64

S

Safety communications	8
Safety control	8
Safety device	8
Safety function (Safety FUN)	35
Safety function block (Safety FB)	36
Safety global label.	50
Safety local label.	50
Safety program.	8
SC.	93
SE.	93
Selective sequence (divergence/convergence).	108
Series sequence	108
SFC program	9
SFC Program Start Mode Setting	128
Shift JIS	39,70,82
Simultaneous sequence (divergence/convergence)	108
Single-precision real number	39
ST.	93
Standard communications	8
Standard control	8
Standard CPU	8
Standard device	8
Standard program	8
Standard/safety shared label	50
Start Conditions Setting	128
Statement	57
Step Transition Bit.	120
STRING.	39,70,82

String	39,70,82
String [Unicode]	39,70,82
Structure	40,45
Structure array	46
Structured text (ST)	9
Subroutine program	14
Subroutine type function block	27,28
System label	37

T

TIME	39
Time	39
TIMER.	39,40
Timer	39,40
Transition name	107
Transition No	107
Type specifier.	71,82

U

Unicode.	39,70
----------	-------

W

WHILE...DO.	65
WORD	39
Word [signed].	39
Word [unsigned]/bit string [16 bits]	39
WSTRING	39,70,82

X

XOR	60
-----	----

REVISIONS

*The manual number is given on the bottom left of the back cover.

Revision date	*Manual number	Description
June 2014	SH(NA)-081265ENG-A	First edition
February 2015	SH(NA)-081265ENG-B	■Added or modified parts Chapter 1, Section 4.7, 6.1, Chapter 7
August 2015	SH(NA)-081265ENG-C	■Added or modified parts CONDITIONS OF USE FOR THE PRODUCT, TERMS, Chapter 1, Section 2.1, 3.1, 3.2, 3.3, 3.4, 4.3, 4.8, 6.1, Chapter 8, WARRANTY
January 2016	SH(NA)-081265ENG-D	■Added or modified parts Section 8.1, 8.2
August 2016	SH(NA)-081265ENG-E	■Added or modified parts Chapter 2, Section 3.1, 3.4, Chapter 5, 6, 7, 8
October 2016	SH(NA)-081265ENG-F	■Added or modified part Chapter 8

Japanese manual number: SH-081225-G

This manual confers no industrial property rights of any other kind, nor does it confer any patent licenses. Mitsubishi Electric Corporation cannot be held responsible for any problems involving industrial property rights which may occur as a result of using the contents noted in this manual.

© 2014 MITSUBISHI ELECTRIC CORPORATION

WARRANTY

Please confirm the following product warranty details before using this product.

1. Gratis Warranty Term and Gratis Warranty Range

If any faults or defects (hereinafter "Failure") found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company.

However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing on-site that involves replacement of the failed module.

[Gratis Warranty Term]

The gratis warranty term of the product shall be for one year after the date of purchase or delivery to a designated place. Note that after manufacture and shipment from Mitsubishi, the maximum distribution period shall be six (6) months, and the longest gratis warranty term after manufacturing shall be eighteen (18) months. The gratis warranty term of repair parts shall not exceed the gratis warranty term before repairs.

[Gratis Warranty Range]

- (1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.
- (2) Even within the gratis warranty term, repairs shall be charged for in the following cases.
 1. Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
 2. Failure caused by unapproved modifications, etc., to the product by the user.
 3. When the Mitsubishi product is assembled into a user's device, Failure that could have been avoided if functions or structures, judged as necessary in the legal safety measures the user's device is subject to or as necessary by industry standards, had been provided.
 4. Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
 5. Failure caused by external irresistible forces such as fires or abnormal voltages, and Failure caused by force majeure such as earthquakes, lightning, wind and water damage.
 6. Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
 7. Any other failure found not to be the responsibility of Mitsubishi or that admitted not to be so by the user.

2. Onerous repair term after discontinuation of production

- (1) Mitsubishi shall accept onerous product repairs for seven (7) years after production of the product is discontinued. Discontinuation of production shall be notified with Mitsubishi Technical Bulletins, etc.
- (2) Product supply (including repair parts) is not available after production is discontinued.

3. Overseas service

Overseas, repairs shall be accepted by Mitsubishi's local overseas FA Center. Note that the repair conditions at each FA Center may differ.

4. Exclusion of loss in opportunity and secondary loss from warranty liability

Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation to:

- (1) Damages caused by any cause found not to be the responsibility of Mitsubishi.
- (2) Loss in opportunity, lost profits incurred to the user by Failures of Mitsubishi products.
- (3) Special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products.
- (4) Replacement by the user, maintenance of on-site equipment, start-up test run and other tasks.

5. Changes in product specifications

The specifications given in the catalogs, manuals or technical documents are subject to change without prior notice.

TRADEMARKS

Ethernet is a registered trademark of Fuji Xerox Co., Ltd. in Japan.

The company names, system names and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.

In some cases, trademark symbols such as "™" or "®" are not specified in this manual.

SH(NA)-081265ENG-F(1610)

MODEL: R-P-PS-E

mitsubishi electric corporation

HEAD OFFICE : TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS : 1-14, YADA-MINAMI 5-CHOME, HIGASHI-KU, NAGOYA, JAPAN

When exported from Japan, this manual does not require application to the
Ministry of Economy, Trade and Industry for service transaction permission.

Specifications subject to change without notice.