MITSUBISHI ELECTRIC

Programmable Controller

MELSEC iQ-F series

MELSEC iQ-F
FX5 Programming Manual (Program Design)

SAFETY PRECAUTIONS

(Read these precautions before using this product.)
Before using the FX5 PLCs, please read the manual supplied with each product and the relevant manuals introduced in that manual carefully and pay full attention to safety to handle the product correctly.
Store this manual in a safe place so that it can be taken out and read whenever necessary. Always forward it to the end user.

INTRODUCTION

This manual describes the instructions and functions required for programming of the FX5. Please read this manual and the relevant manuals and understood the functions and performance of the FX5 PLCs before attempting to use the unit.
It should be read and understood before attempting to install or use the unit. Store this manual in a safe place so that you can take it out and read it whenever necessary. Always forward it to the end user.
When utilizing the program examples introduced in this manual to the actual system, always confirm that it poses no problem for control of the target system.

Regarding use of this product

- This product has been manufactured as a general-purpose part for general industries, and has not been designed or manufactured to be incorporated in a device or system used in purposes related to human life.
- Before using the product for special purposes such as nuclear power, electric power, aerospace, medicine or passenger movement vehicles, consult with Mitsubishi Electric.
- This product has been manufactured under strict quality control. However when installing the product where major accidents or losses could occur if the product fails, install appropriate backup or failsafe functions in the system.

Note

- If in doubt at any stage during the installation of the product, always consult a professional electrical engineer who is qualified and trained to the local and national standards. If in doubt about the operation or use, please consult the nearest Mitsubishi Electric representative.
- Since the examples indicated by this manual, technical bulletin, catalog, etc. are used as a reference, please use it after confirming the function and safety of the equipment and system. Mitsubishi Electric will accept no responsibility for actual use of the product based on these illustrative examples.
- This manual content, specification etc. may be changed without a notice for improvement.
- The information in this manual has been carefully checked and is believed to be accurate; however, if you have noticed a doubtful point, a doubtful error, etc., please contact the nearest Mitsubishi Electric representative. When doing so, please provide the manual number given at the end of this manual.

CONTENTS

RELEVANT MANUALS

Manual name <manual number>	Description
MELSEC iQ-F FX5 Programming Manual (Program Design) <JY997D55701> (This manual)	Describes the specifications of ladder, ST, FBD/LD, and SFC programs, and labels.
MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks) <JY997D55801>	Describes the specifications of instructions and functions that can be used in programs.
GX Works3 Operating Manual <SH-081215ENG>	Describes the system configuration, parameter settings, and online operations of GX Works3.

TERMS

Unless otherwise specified, this manual uses the following terms.

Terms	Description
Buffer memory	A memory in an intelligent function module, where data (such as setting values and monitoring values) are stored.
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
POU	Defined unit of a program. Use of POUs enables a program to be divided into units according to process or function, and each unit to be programmed individually.
Signal flow	The execution status that the last time an operation of a program or an FB is executed in each step.

GENERIC TERMS AND ABBREVIATIONS

Unless otherwise specified, this manual uses the following generic terms and abbreviations.

Generic term/abbreviation	Description
FX3 intelligent function module	A generic term for FX3U-4AD, FX3U-4DA, FX3U-4LC, FX3U-1PG, FX3U-2HC, FX3U-16CCL-M, FX3U-64CCL, FX3U-128ASL-M, and FX3U-32DP
FX5	A generic term for FX5S, FX5UJ, FX5U, and FX5UC PLCs
FX5 CPU module	A generic term for FX5S CPU module, FX5UJ CPU module, FX5U CPU module, and FX5UC CPU module
FX5 intelligent function module	A generic term for FX5-4AD, FX5-4DA, FX5-8AD, FX5-4LC, FX5-20PG-P, FX5-20PG-D, FX5-40SSC-G, FX5-80SSC-G, FX5-40SSC-S, FX5-80SSC-S, FX5-ENET, FX5-ENET/IP, FX5-CCLGN-MS, FX5-CCLIEF, FX5-CCL-MS, FX5-ASL-M, and FX5-DP-M
GX Works3	The product name of the software package, SWnDND-GXW3, for the MELSEC programmable controllers (The 'n' represents a version.)
Intelligent function module	A generic term for FX5 intelligent function modules and FX3 intelligent function modules
Operand	A generic term for items, such as source data (s), destination data (d), number of devices (n), and others, used to configure instructions and functions.

1 OUTLINE

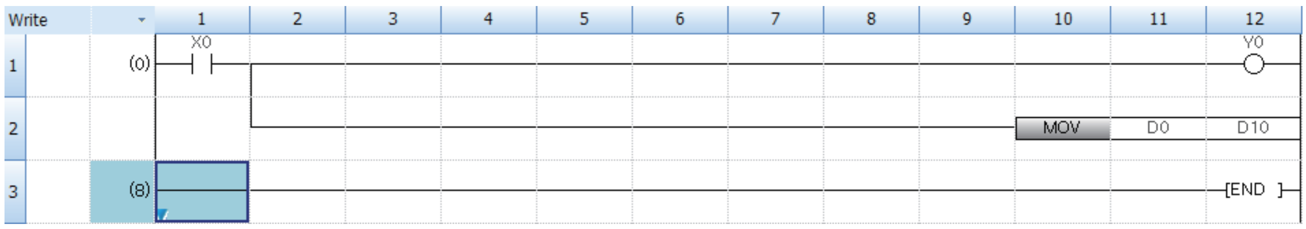
This manual describes program configurations, content, and method for creating programs.
For how to create, edit, or monitor programs using the engineering tool, refer to the following.
📖GX Works3 Operating Manual

Type of programming languages

With the FX5 series, the optimal programming language can be selected according to the application.
○: Applicable —: Inapplicable

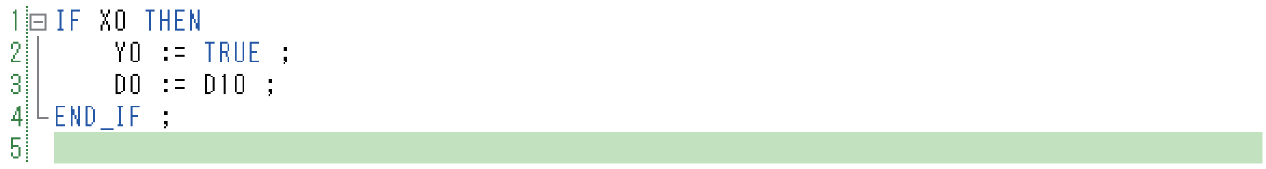
Programming language	Description	Applicability to CPU module		
		FX5S	FX5UJ	FX5U/ FX5UC
Ladder diagram	Ladder diagram is a graphic language that indicates circuits using contacts, coils, and others. The ladder diagram describes logic circuits with symbolized contacts and coils for easy-to-understand sequence control.	○	○	○
Structured text language (ST language)	ST language is a text language that describes programs with IF statements, operators, and others. Because operation processing that is difficult to describe in ladder diagram can be easily and briefly described with ST language, ST language is suitable for applications requiring complicated arithmetic operation or comparative operation. With ST language, programs can be easily described with syntax using selective branches with conditional statements and repetition by repetitive statements in the same way as C language.	○	○	○
Function block diagram/ ladder diagram (FBD/LD language)	This is a graphic language that describes a program by wiring blocks for specific processing (function elements, FB elements), variable elements, and constant elements along with the flows of data and signals. You can easily create a program that may be complicated to create by using a ladder program. So you can enhance 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



When using ladder diagram, refer to the following.
☞ Page 42 LADDER DIAGRAM

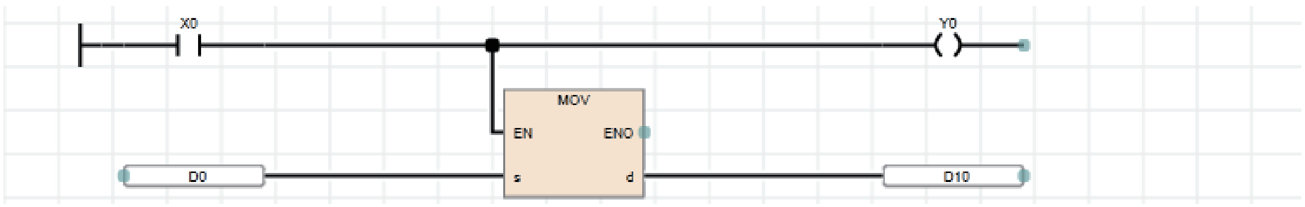
■ST language



```
IF X0 THEN
    Y0 := TRUE ;
    D0 := D10 ;
END_IF ;
```

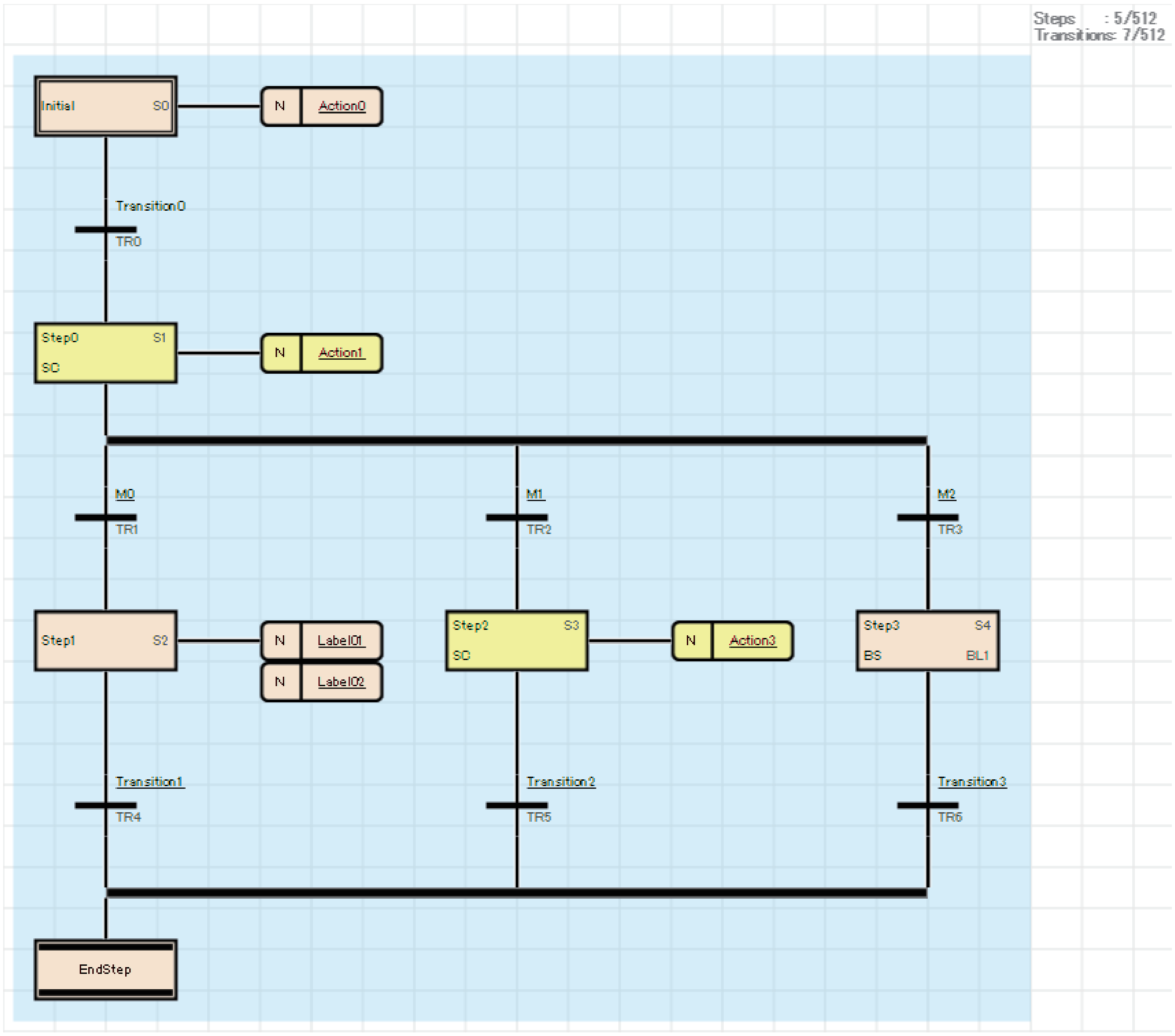
When using ST language, refer to the following.
☞ Page 47 ST LANGUAGE

■FBD/LD language



When using FBD/LD language, refer to the following.
☞ Page 61 FBD/LD LANGUAGE

■SFC program



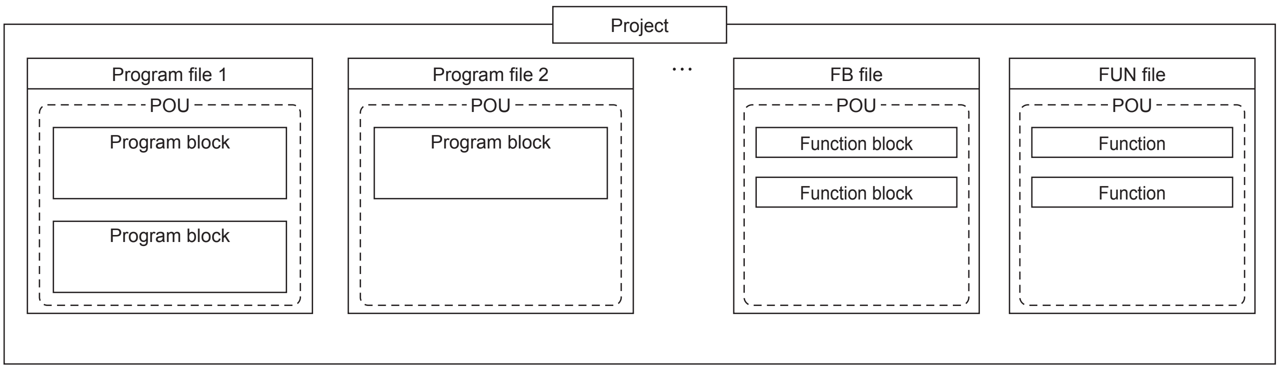
When using SFC program, refer to the following.
☞ Page 71 SFC PROGRAM

Point

- Ladder diagram and FBD/LD language are for customers who have knowledge or experience of sequence control and logic circuits.
- ST language is for customers who have knowledge or experience of the C language programming.
- SFC program is suitable for creating program blocks for each actual control of machines and controlling the transition of each operation.
- By using labels in a program, the readability of the program is improved, and activating a program for the system with a different module configuration is easy.

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 9 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 10 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.

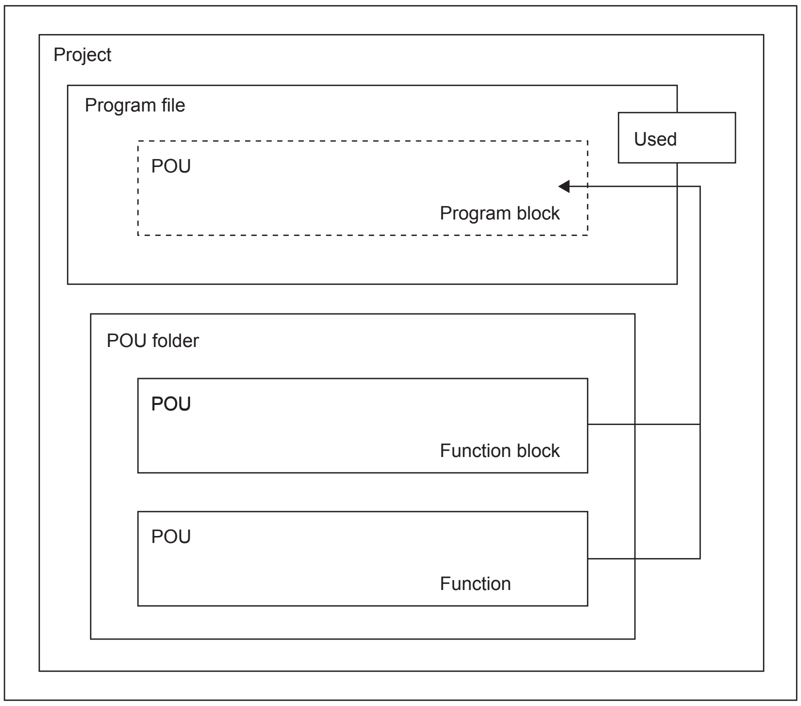
MEMO

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.



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

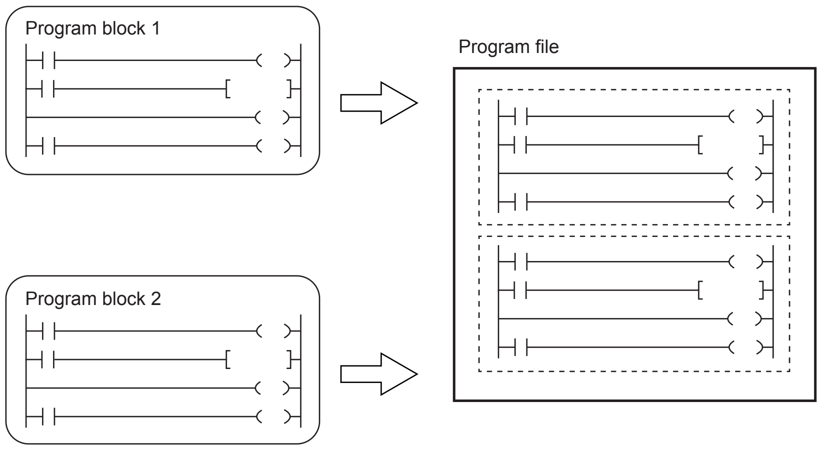
This chapter describes three types of POUs using labels.
Devices can also be used in the program (worksheet) of each POU. For details on devices, refer to the following.
📖 MELSEC iQ-F FX5 User's Manual (Application)

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.



Dividing into program blocks

A main routine program, subroutine program, and interrupt program can be created separately in individual program blocks.

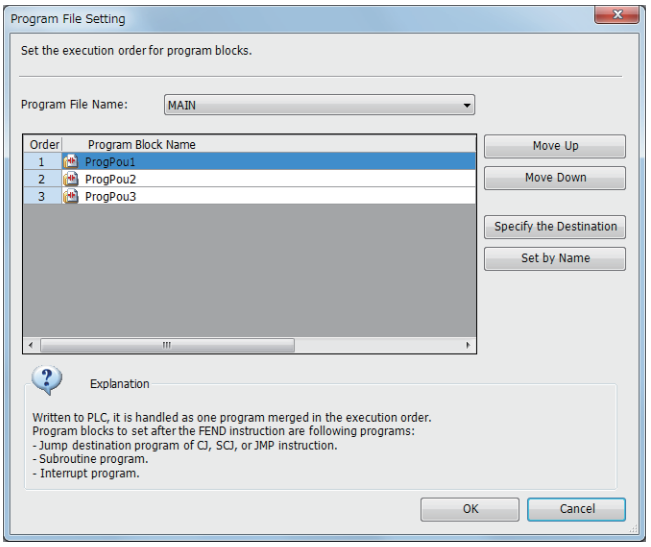
Program type	Description
Main routine program	A program beginning with step 0 and ending with the FEND instruction
Subroutine program	A program beginning with a pointer (P) and ending with the RET instruction. This program is executed only when it is called by a subroutine call instruction (CALL and XCALL 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.

■Program file setting

In the program file setting, the order of executions of program blocks in a program file can be set.

[Convert] ⇨ [Program File Setting]

[Navigation window] ⇨ Select and right-click the program file. ⇨ [Program File Setting]


Program File Setting

Set the execution order for program blocks.

Program File Name: MAIN

Order	Program Block Name
1	ProgPou1
2	ProgPou2
3	ProgPou3

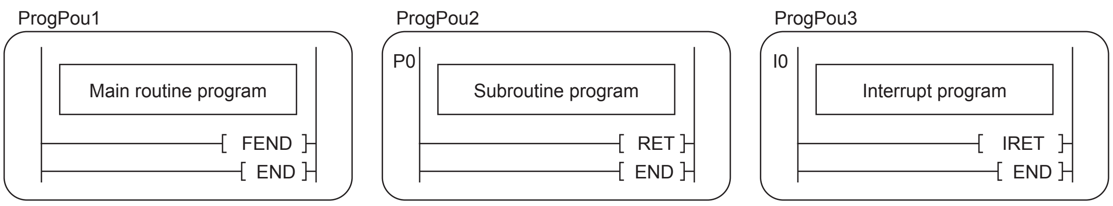
Explanation

Written to PLC, it is handled as one program merged in the execution order.
Program blocks to set after the FEND instruction are following programs:
- Jump destination program of CJ, SCJ, or JMP instruction.
- Subroutine program.
- Interrupt program.

For details of the program file setting, refer to the following.
GX Works3 Operating Manual

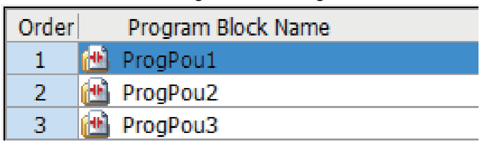
Ex.

Create a program block as shown below.

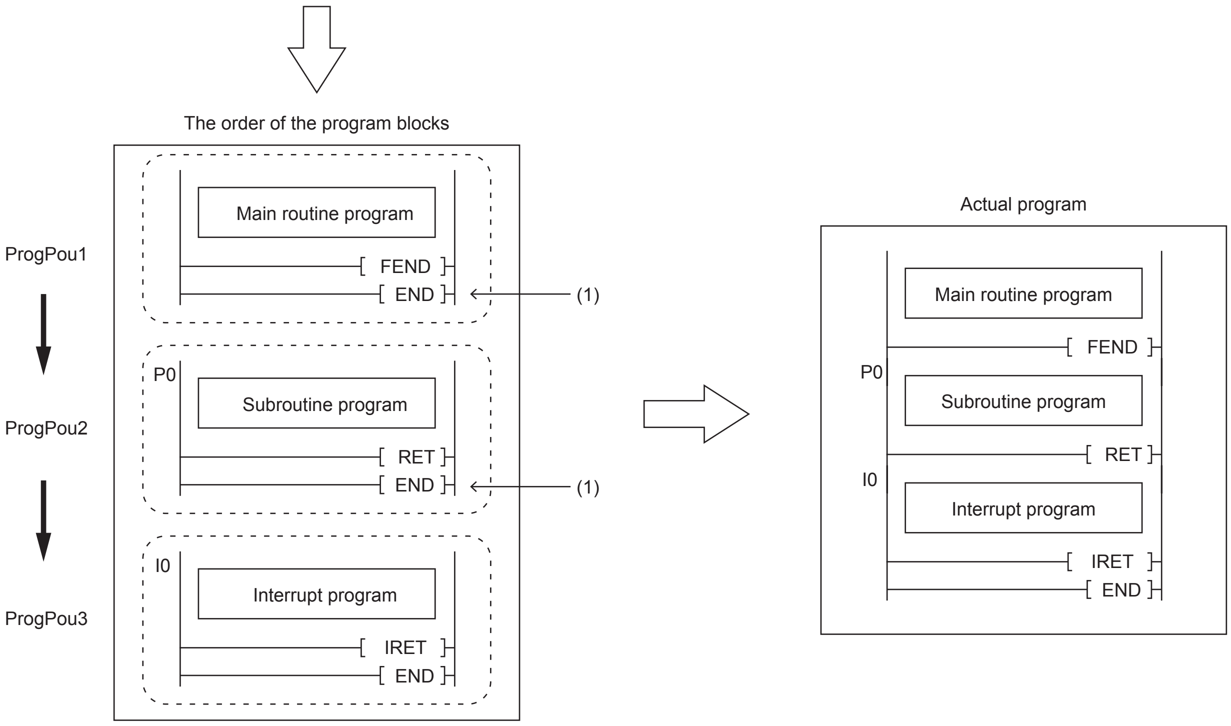


Execute the program according to the order of the execution of program file setting.

Program file setting



Order	Program Block Name
1	ProgPou1
2	ProgPou2
3	ProgPou3



(1) The END instruction in the middle of the program file is ignored.

Point

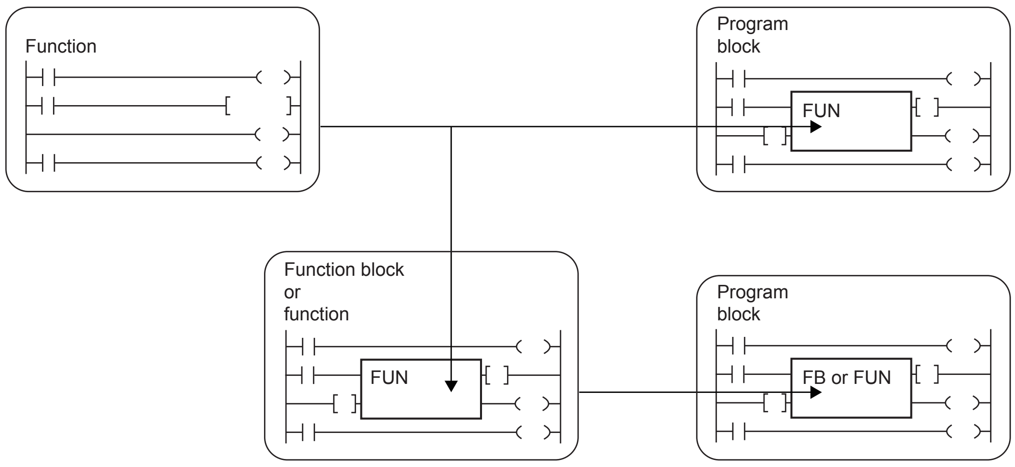
- Create a subroutine program and interrupt program after the FEND instruction of the main routine program. Any program after the FEND instruction is not executed as a main routine program. For example, when the FEND instruction is used at the end of the second program block, the third program block or later runs as a subroutine program or interrupt program. (☞ Page 27 When a subroutine program or an interrupt program is used)
- 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.

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

📖 MELSEC iQ-F FX5 User's Manual (Application)

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.

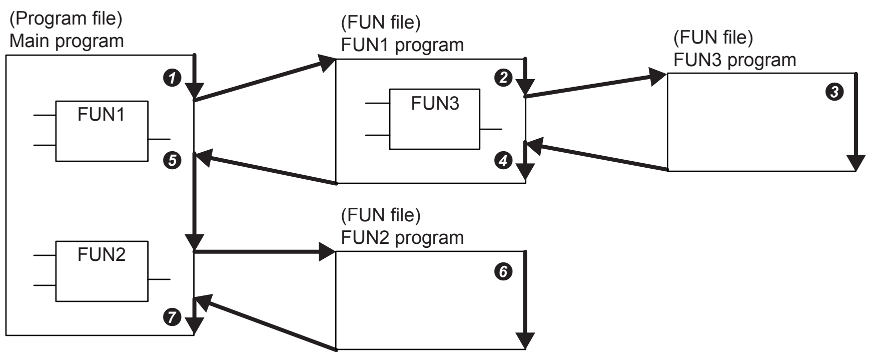


Operation overview

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

Ex.

When calling FUN1 and FUN2 from the main program, and calling FUN3 by FUN1 (Nested three times)
❶ to ❼ indicate the execution flow (order).

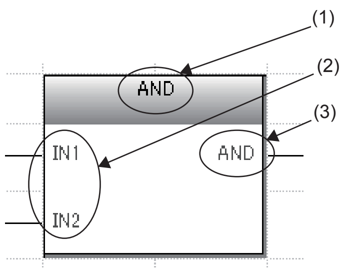


Up to 32 subroutine type function blocks, macro type function blocks, and functions in total can be nested.

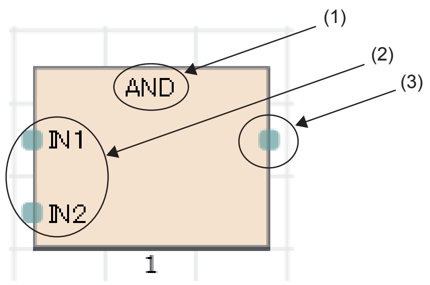
Input variables and output variables

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



The return value of the function is not displayed.

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

Input variables are set in the VAR_INPUT class and output variables are set in the VAR_OUTPUT class.

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.

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]
Select "Function" for "Data Type" in "Basic Setting".

The created program is stored in the FUN file.

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

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

The rising edge execution instruction or falling edge execution instruction cannot be used in the function.

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-F FX5S/FX5UJ/FX5U/FX5UC User's Manual (Hardware)

■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	○[*1]
Label (pointer type)	Pointer type global label	△
	Pointer type local label	○
Device	Global device	○
Pointer	Global pointer	△

*1 The following data types cannot be used.
Timer, retentive timer, counter, 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.

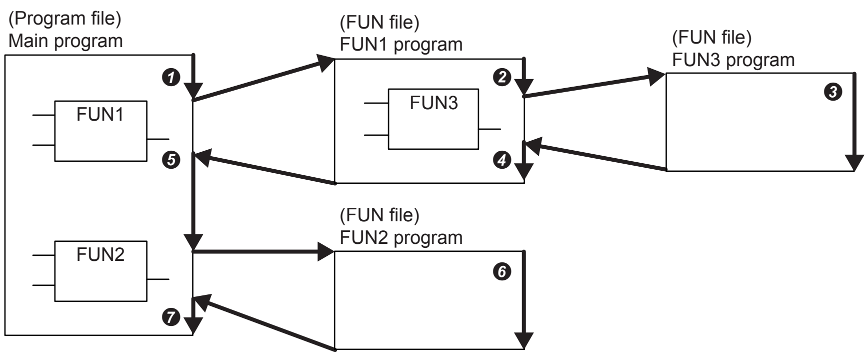
Labels defined by a function

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.

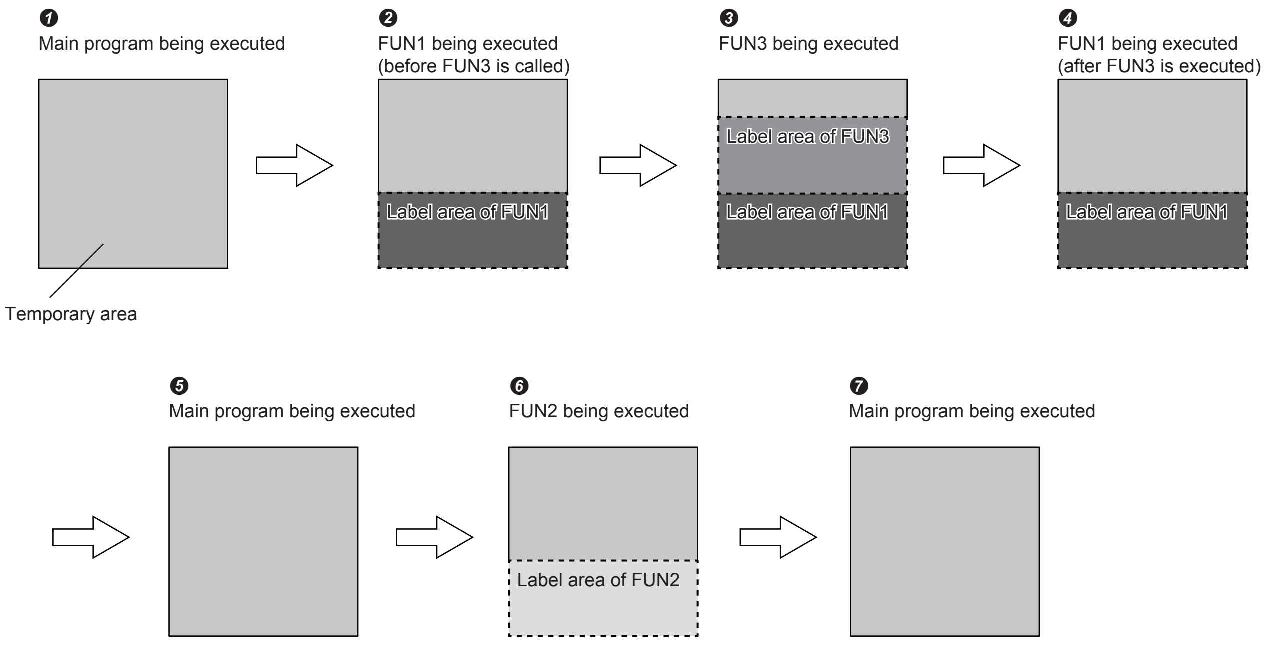
Ex.

When calling FUN1 and FUN2 from the main program, and calling FUN3 by FUN1

(❶ to ❼ indicate the execution flow (order).)



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



The class of labels that can be defined in the function are VAR, VAR_CONSTANT, VAR_INPUT, and VAR_OUTPUT.

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.

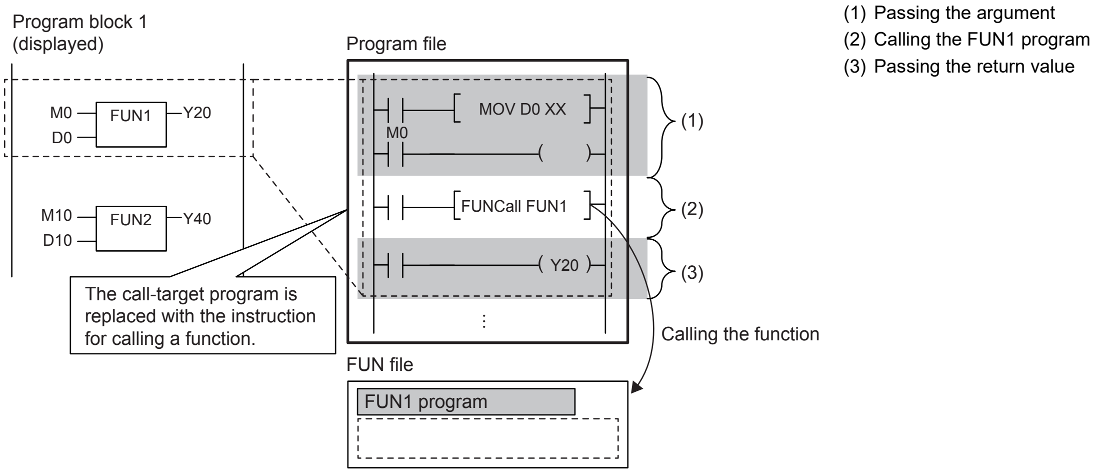
■Program

The number of steps required for a function program is the total number of instruction steps plus a minimum of additional 13 steps occupied by the system. For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-F FX5 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-F FX5 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	
	Time	LD+DMOV	
	String(32)	LD+$MOV	
	String [Unicode](32)	LD+$MOV_WS	
	Array, Structure	LD+BMOV	

- Calling the program

The following table lists the number of steps required to call the program of the function.

Item	Number of steps
With EN	10
Without EN	12

- 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

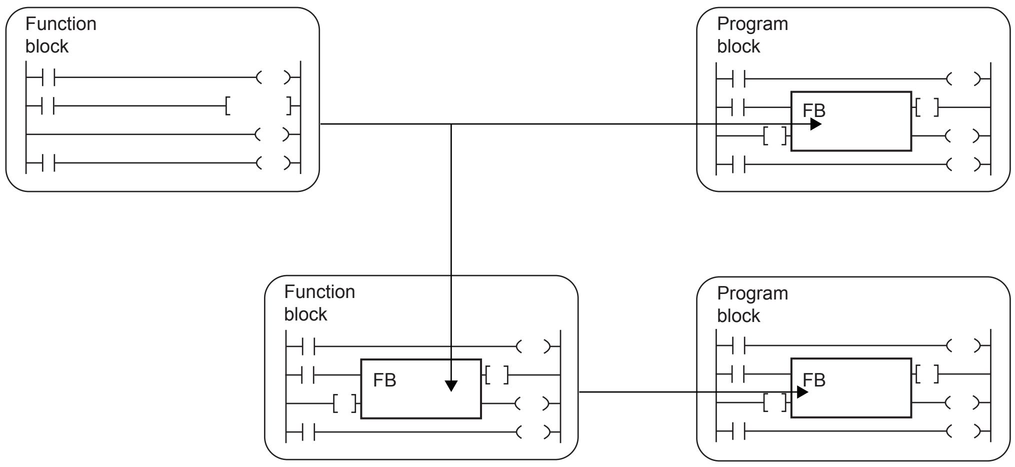
• EN/ENO

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

Item	Number of steps
EN	4 to 7 (The number of steps differs depending on the details of the program such as the type and number of the device specified as the input source of EN.)
ENO	6 to 10 (The number of steps differs depending on the details of the program such as the type and number of the device specified as the output destination of ENO.)

3.3 Function Blocks (FB)

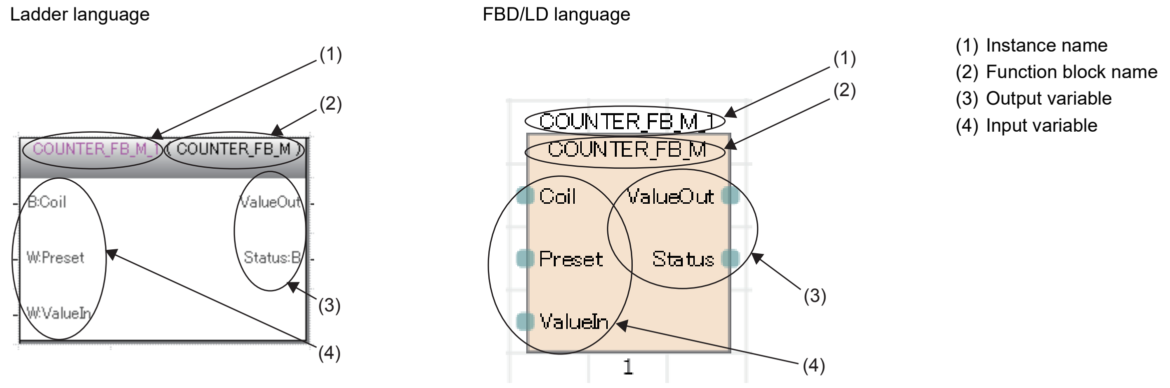
A function block is a POU called and executed by program blocks and other function blocks.



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.



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

☞ Page 20 Instances

Point

• For details on standard function blocks, refer to the following.

📖 MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

• For details on module function blocks, refer to the following.

📖 Function Block Reference for the module used

Operation overview

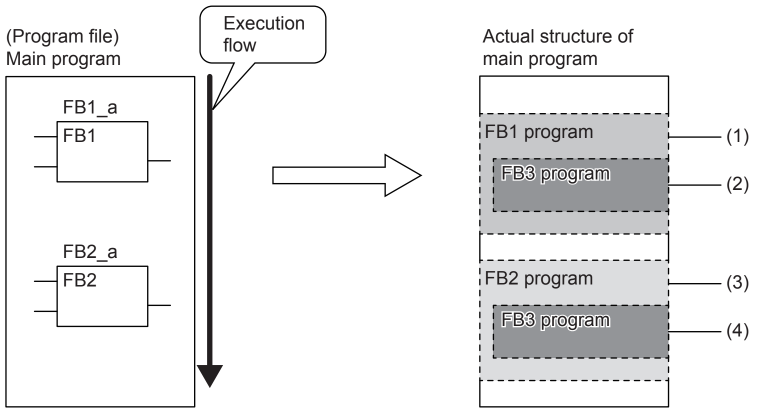
■Macro type function blocks

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.

Ex.

When calling FB1_a and FB2_a from the main program, calling FB3_a by FB1_a, and calling FB3_b by FB2_a



(1) The FB1 program is loaded into the main program and executed.
(2) The FB3 program called by FB1 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.
(4) The FB3 program called by FB2 is loaded into the FB2 program.

■Subroutine type function blocks

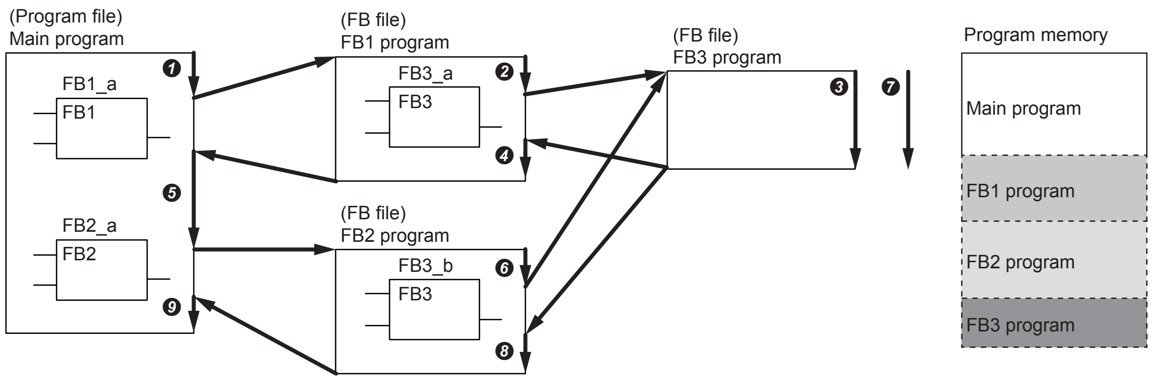
The program of a subroutine type function block is stored in the FB file and called by the calling source program when executed.

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

Ex.

When calling FB1_a and FB2_a from the main program, calling FB3_a by FB1_a, and calling FB3_b by FB2_a (Nested three times)

❶ to ❾ indicate the execution flow (order).

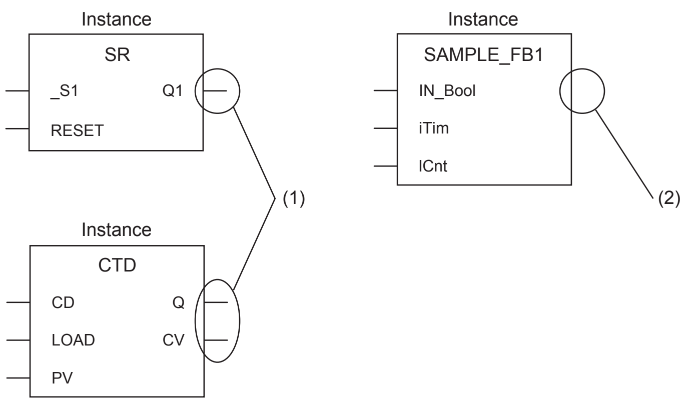


Up to 32 subroutine type function blocks, macro type function blocks, and functions in total can be nested.

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.



(1) The operation result(s) is output.

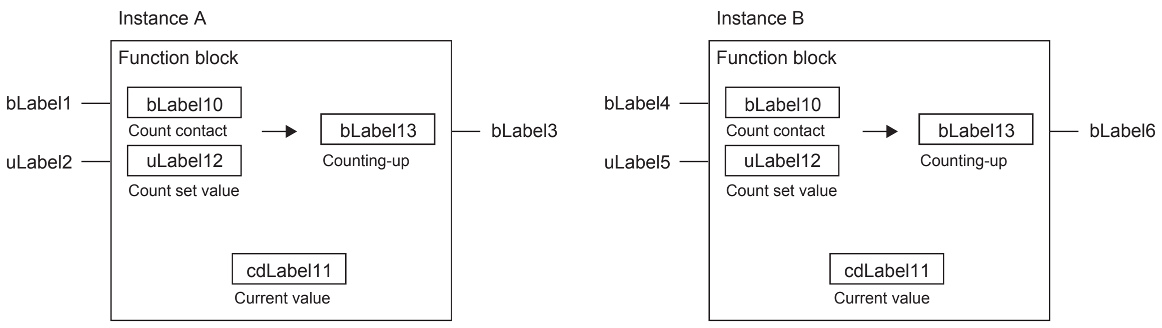
(2) No operation result is output.

Input variables are set in the VAR_INPUT class, output variables are set in the VAR_OUTPUT class and VAR_OUTPUT_RETAIN class, and input/output variables are set in the VAR_IN_OUT class.

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.

Internal variables are set in the VAR, VAR_CONSTANT and VAR_RETAIN class.

External variables and public variables

Function blocks can use external variables (global label) and public variables.

Public variables are set in the VAR_PUBLIC and VAR_PUBLIC_RETAIN class.

Instances

■Instances

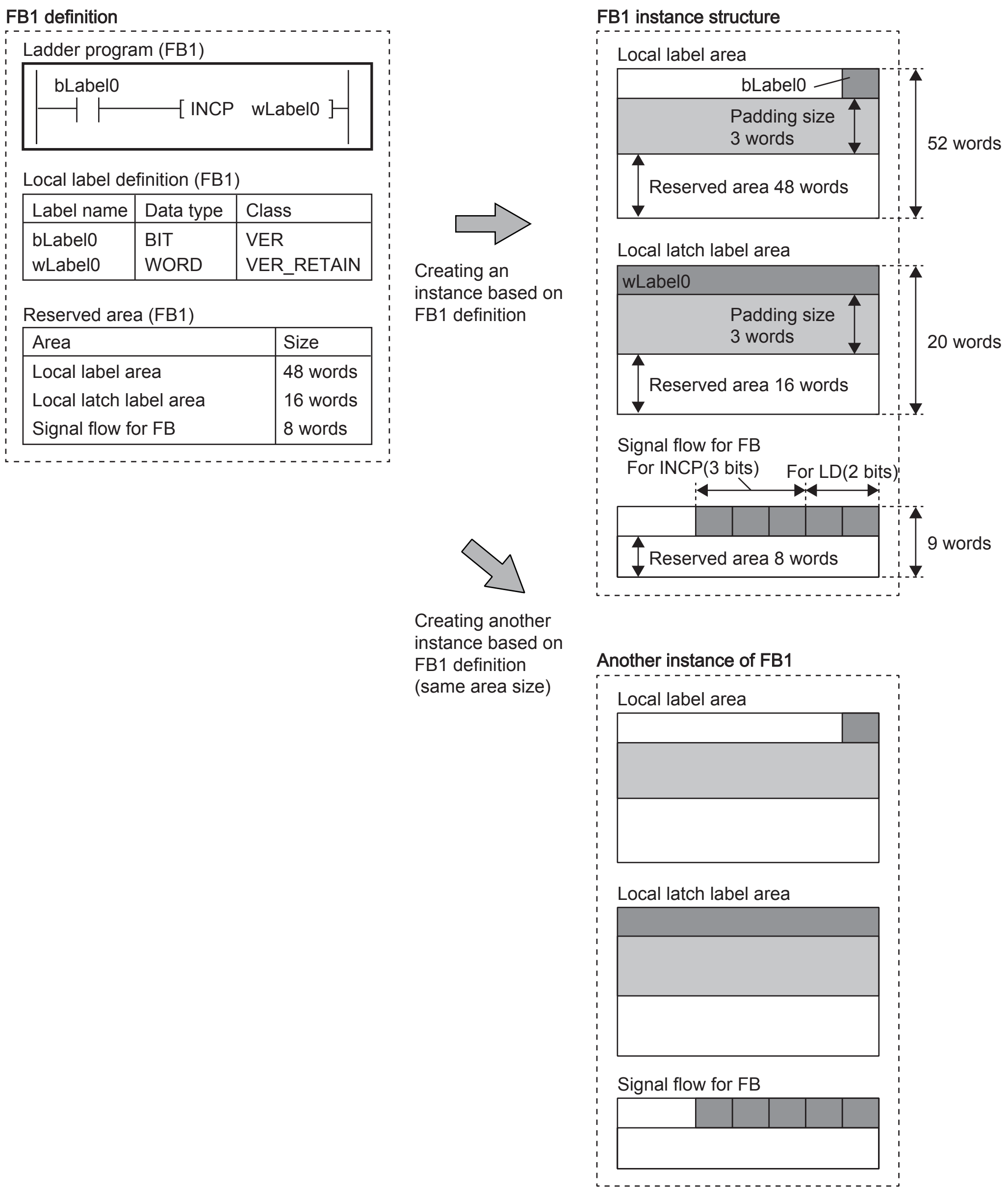
An instance is a label assigned to realize a function block definition. Multiple instances can be created from one function block definition.

An instance consists of the following items.

Item	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.
Signal flow area (Signal flow for FB)	Used to assign the signal flow for the instruction in the function block definition.

Ex.

Structure of instance (Example of subroutine type function block)



For the local label area and local latch label area, since the label area is secured in units of four words, three-words (padding size) are secured in the above example.

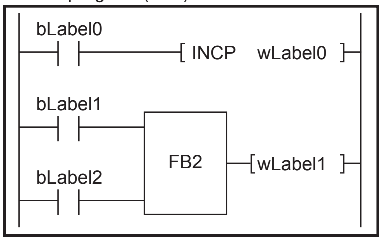
Each area occupies a reserved area. The reserved area is used to add or change the local label, instructions, or instances of the function block while keeping the label assignment by conversion or online change. If the area of the target data type to be added cannot be secured, all programs are required to be converted (reassigned).

Ex.

Structure of instance for nested function block

FB1 definition

Ladder program (FB1)



Local label definition (FB1)

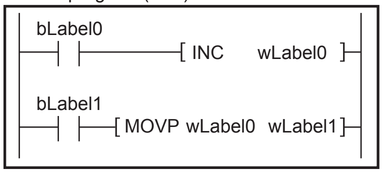
Label name	Data type	Class
bLabel0	BIT	VER
bLabel1	BIT	VER
bLabel2	BIT	VER
wLabel0	WORD	VER_RETAIN
wLabel1	WORD	VER
FB2_a	FB2	VER

Reserved area (FB1)

Area	Size
Local label area	48 words
Local latch label area	16 words
Signal flow for FB	8 words

FB2 definition

Ladder program (FB2)

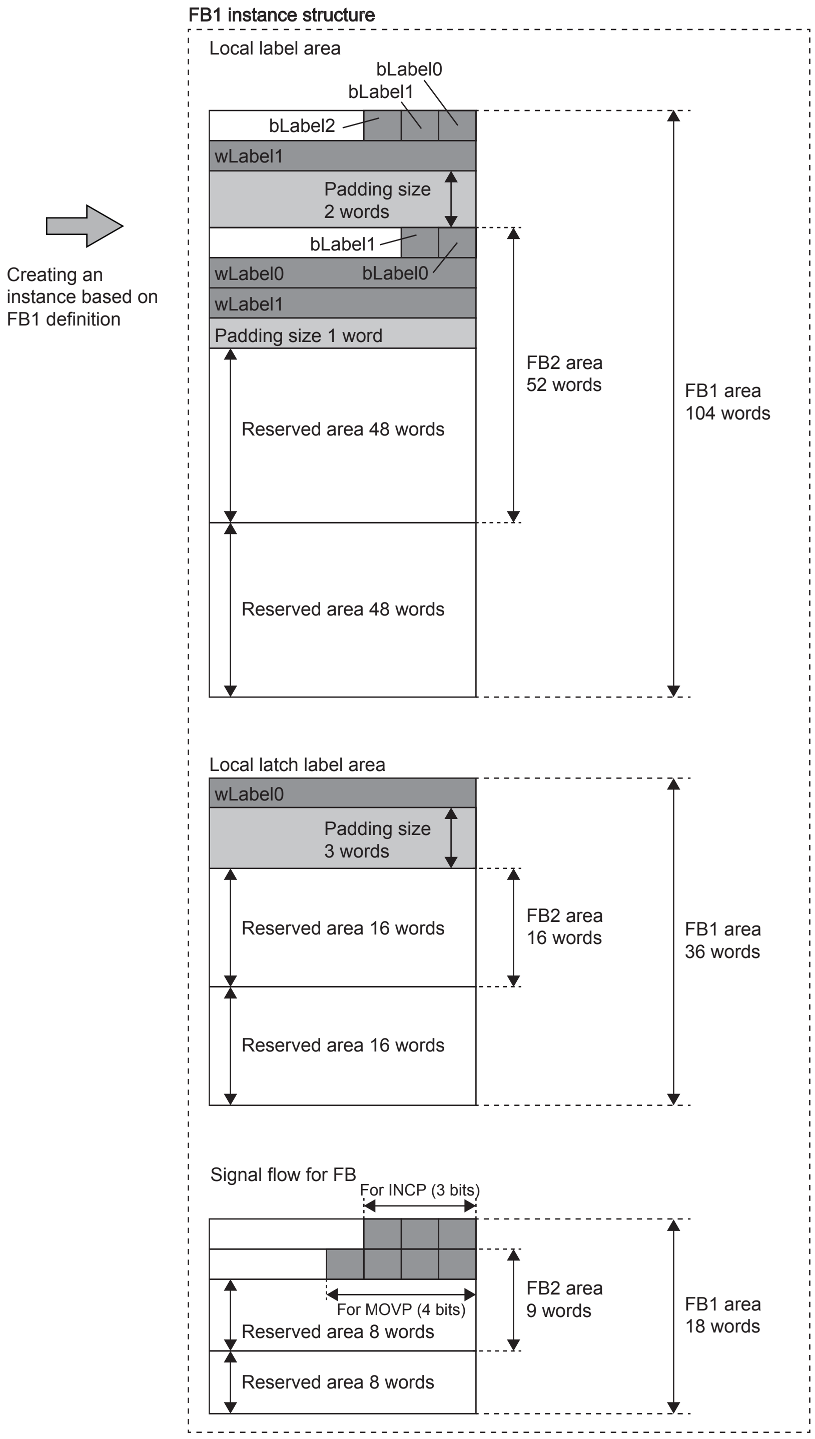


Local label definition (FB2)

Label name	Data type	Class
bLabel0	BIT	VER_INPUT
bLabel1	BIT	VER_INPUT
wLabel0	WORD	VER
wLabel1	WORD	VER_OUTPUT

Reserved area (FB2)

Area	Size
Local label area	48 words
Local latch label area	16 words
Signal flow for FB	8 word



The instance of FB2 declared as a local label is secured in the local label area, local latch label area, and signal flow for FB of FB1 which is the declared source.

When an FB type local label to FB1 is added in the above example, since the capacity of the reserved area is 48 words for local label area, 16 words for local latch label area, and 8 words for signal flow for FB, all programs are required to be converted (reassigned) to add a function block with an area exceeding the capacity.

■Creating instances

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

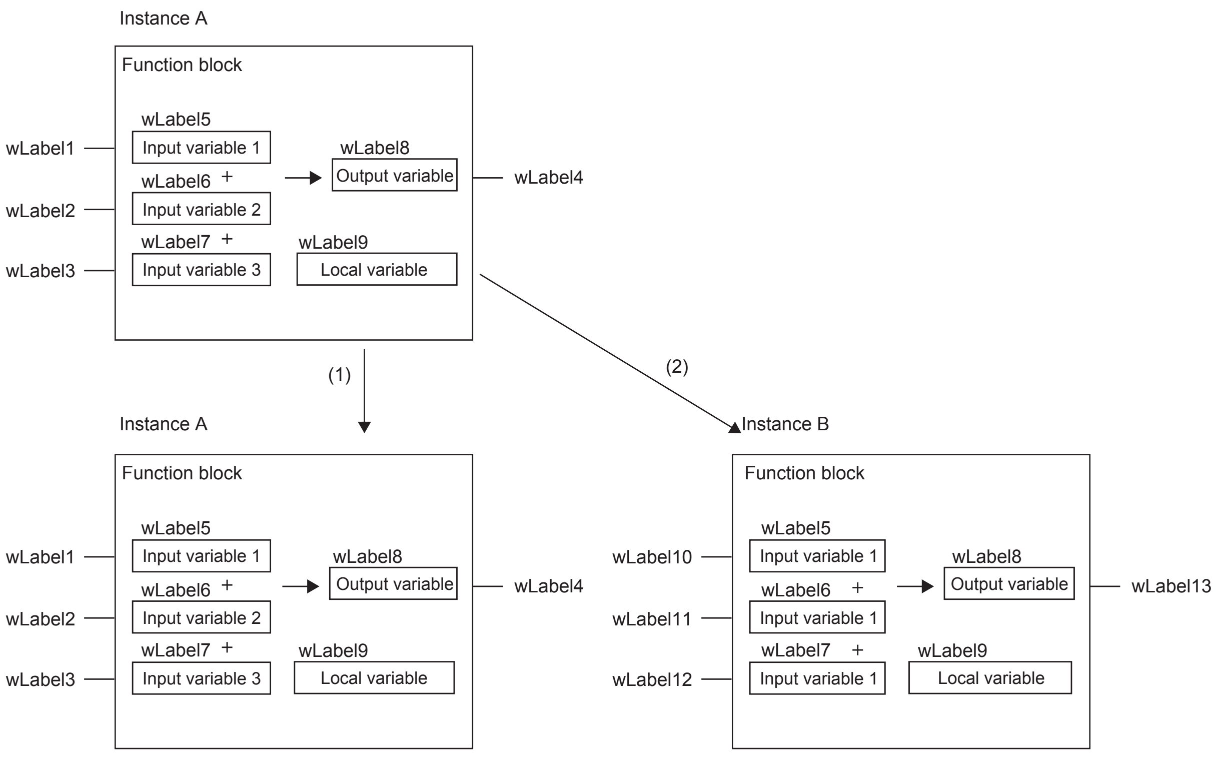
By creating instances, a function block can be called and executed by a program block or another function block.

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.



(1) Same instances use the same internal variables.

(2) Different instances use different internal variables.

■Capacity of instance

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

- Capacity of 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

Item	Description
Capacity of local labels (except the ones with latch attribute)	Total capacity of the data used for local labels. The capacity of areas to be used differs depending on the memory assignment of labels. For details on memory assignment of labels, refer to the following. 📖 GX Works3 Operating Manual
Capacity of reserved area	48 words.

- Capacity of 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

Item	Description
Capacity of latch attribute local labels	Total capacity of the data used for latch attribute local labels. The capacity of areas to be used differs depending on the memory assignment of labels. For details on memory assignment of labels, refer to the following. 📖 GX Works3 Operating Manual
Capacity of reserved area	16 words.

- Capacity of signal flow for FB

In the macro type function block, the number of steps are the same as the program.

The capacity of the subroutine type function block is as follows.

Capacity of signal flow for FB (word) = The number of program steps of the function block / 16 + Capacity of reserved area

Item	Description
Capacity of signal flow for FB	Total capacity of the signal flow for FB for the instruction in the function block definition
Capacity of reserved area	8 words.

Point

If the reserved area capacity cannot be allocated to the data to be added by online change, the online change cannot be executed and all programs are required to be converted (reassigned).

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 13 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]
Select "Function Block" for "Data Type" in "Basic Setting".

The created program is stored in the FB file.

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

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

For details on program creation, refer to the following.

Item	Reference
How to create function blocks	📖 GX Works3 Operating Manual
Number of FB/FUN files that can be written to a CPU module	📖 MELSEC iQ-F FX5S/FX5UJ/FX5U/FX5UC User's Manual (Hardware)

■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 18 Operation overview

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

■Inherent property setting

The following items can be set when a program of a function block is created. (📖 GX Works3 Operating Manual)

Item	Description
Use MC/MCR to Control EN[*1]	For "Yes", the MC/MCR instructions are used to control EN. For "No", the CJ instruction is used to control EN. Select "Yes" when instructions executed at the rising edge or falling edge are used in an FB. The operations of a timer/counter and the OUT instruction used in an FB differ depending on the selected item. For details, refer to the following. ☞ Page 114 Operations of when the MC/MCR instructions are used to control EN
Use EN/ENO	For "Yes", a function block with EN/ENO is created, and EN/ENO labels can be used in a program without registering as local labels. For "No", a function block without EN/ENO is created. For details on EN/ENO, refer to the following. ☞ Page 24 EN and ENO

*1 To select this item, select "Yes" for "Use EN/ENO". However, the item cannot be used depending on the versions of the CPU module and GX Works3 used when "Subroutine Type" is selected for "FB Type". For the versions of the CPU module and the GX Works3, refer to the following.
📖 MELSEC iQ-F FX5 User's Manual (Application)

■Applicable devices and labels

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

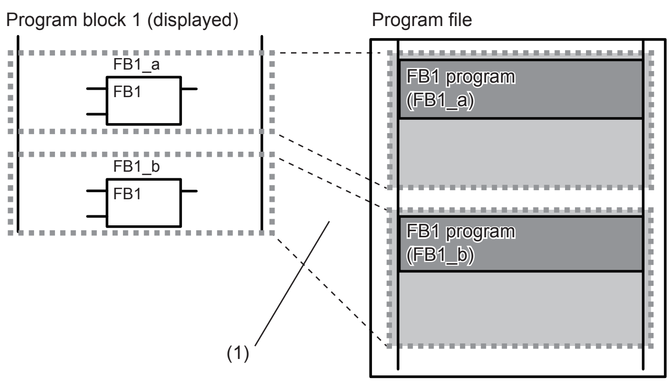
○: Applicable, △: Applicable only in instructions (Cannot be used to indicate the program step.), ×: Not applicable

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	○
Pointer	Global pointer	△

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.

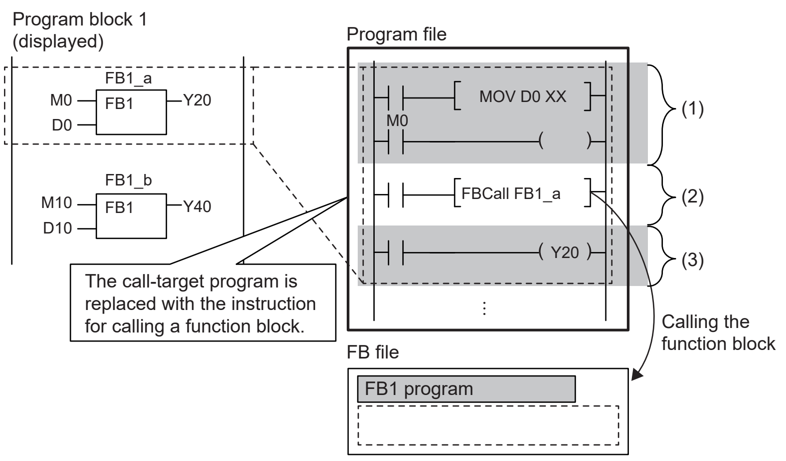
For the number of steps required for each instruction, refer to the following.

📖 MELSEC iQ-F FX5 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 before and after the call processing.



(1) Passing the argument (input argument, input/output argument)
(2) Calling the FB1 program
(3) Passing the argument (output argument, input/output argument)

- 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 VAR_OUTPUT	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-F FX5 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	
	Time	LD+DMOV	
	String(32)	LD+$MOV	
	String [Unicode](32)	LD+$MOV_WS	
	Array, Structure	LD+BMOV	

- Calling the program

The following table lists the number of steps required to call the program of the function block.

Item	Number of steps
With EN	10
Without EN	12

- EN/ENO

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

Item	Number of steps
EN	4 to 7 (The number of steps differs depending on the details of the program such as the type and number of the device specified as the input source of EN.)
ENO	6 to 10 (The number of steps differs depending on the details of the program such as the type and number of the device specified as the output destination of ENO.)

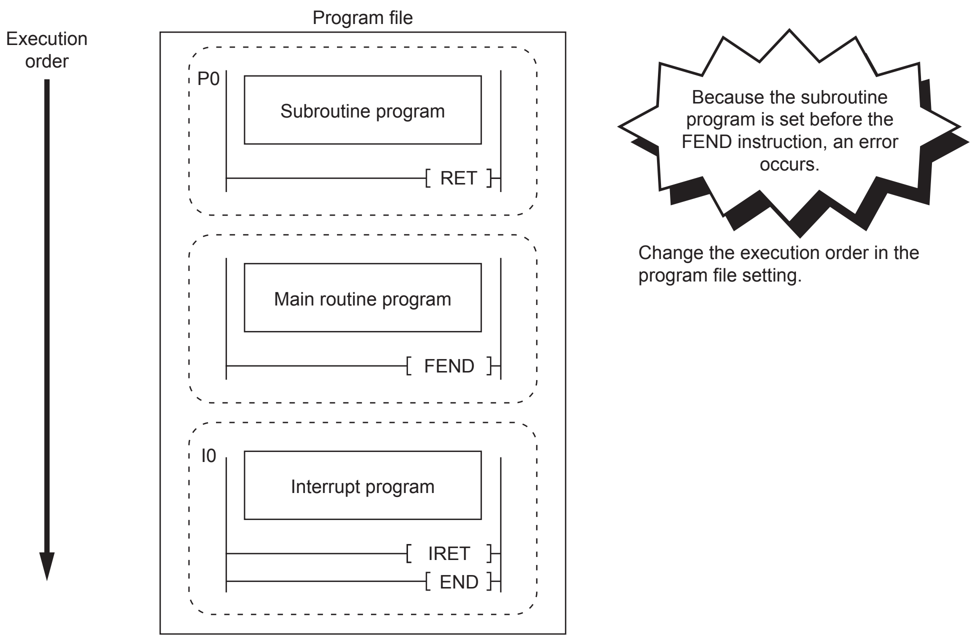
■Program

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-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

3.4 Precautions

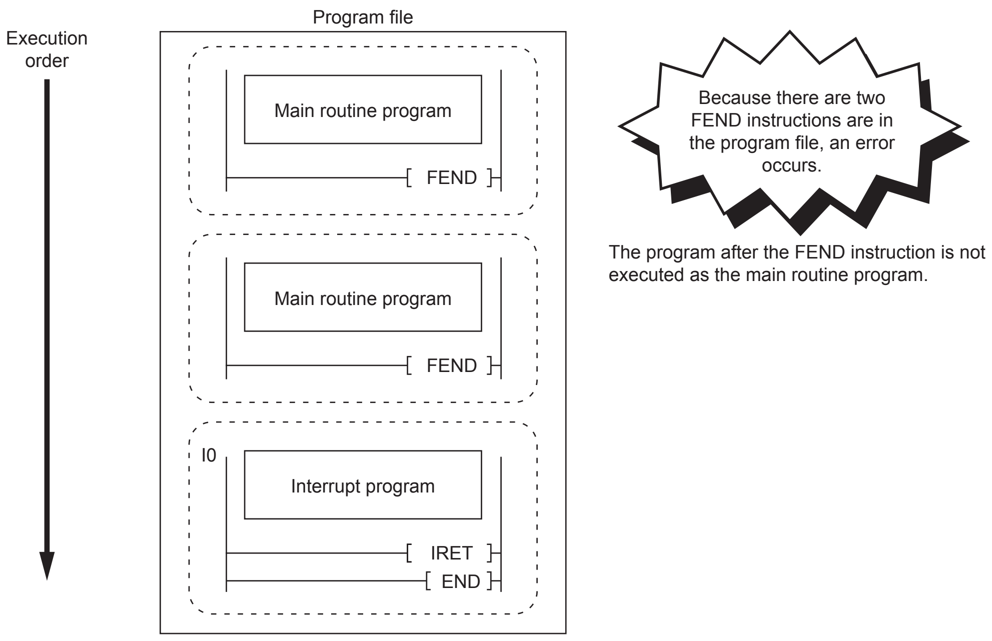
When a subroutine program or an interrupt program is used

- Set subroutine programs and interrupt programs after the FEND instruction. When the subroutine program and interrupt program are set before the FEND instruction, an error occurs.



Program file
Execution order
P0
Subroutine program
RET
Main routine program
FEND
I0
Interrupt program
IRET
END
Because the subroutine program is set before the FEND instruction, an error occurs.
Change the execution order in the program file setting.

- Only one FEND instruction can be used for a program file. When multiple FEND instructions are used, an error occurs.



Program file
Execution order
Main routine program
FEND
Main routine program
FEND
I0
Interrupt program
IRET
END
Because there are two FEND instructions are in the program file, an error occurs.
The program after the FEND instruction is not executed as the main routine program.

When a function is used

■Global pointer/pointer type global labels

The global pointer and pointer type global labels cannot be used as the labels indicating the number of program steps.

When a function block is used

■Global pointer/pointer type global labels

The global pointer and pointer type global labels cannot be used as the labels indicating the number of program steps.

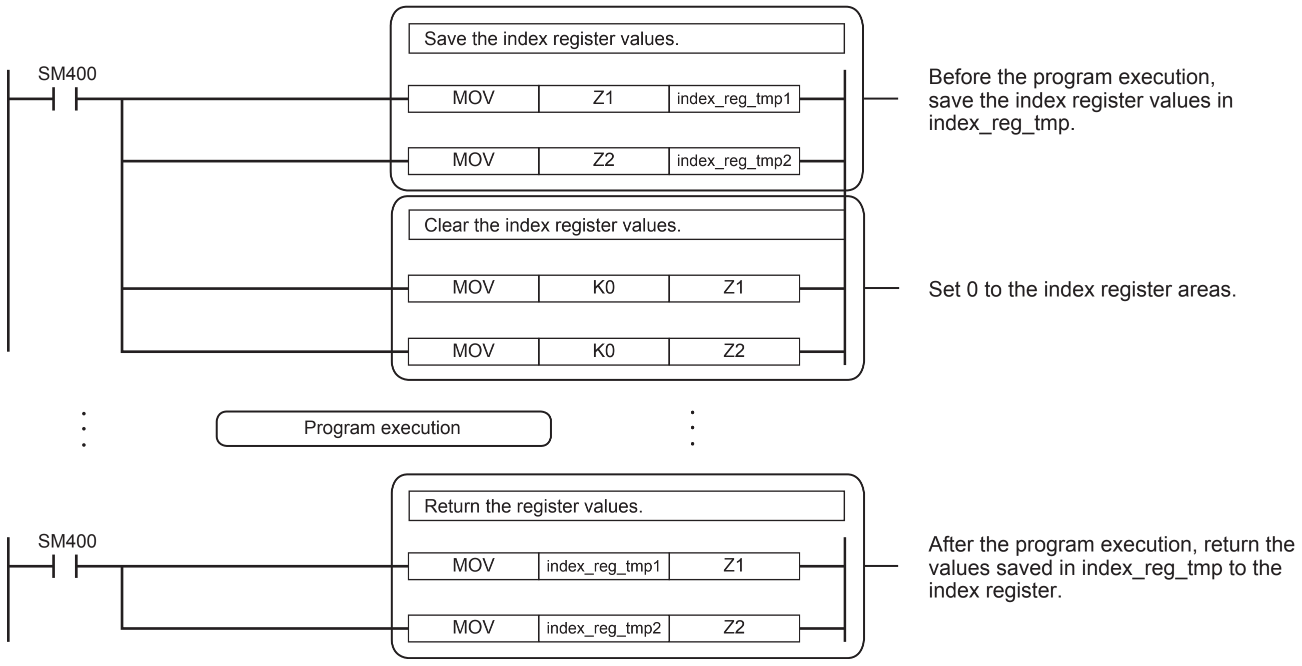
■When the index register is used

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



Save the index register values.
SM400
MOV Z1 index_reg_tmp1
MOV Z2 index_reg_tmp2
Before the program execution, save the index register values in index_reg_tmp.
Clear the index register values.
MOV K0 Z1
MOV K0 Z2
Set 0 to the index register areas.
Program execution
Return the register values.
SM400
MOV index_reg_tmp1 Z1
MOV index_reg_tmp2 Z2
After the program execution, return the values saved in index_reg_tmp to the index register.

■Argument of macro type function block

Except in the program of the macro type function block, use the device/label used for passing the argument instead of the argument of the function block.

Ex.

Device used for passing the argument

```
MacroFbPou_1 (EN := M0, ENO => M1);
M2 := M1;
```

An unintended value may be generated if the argument of the macro type function block is used in other than the program of the macro type function block.

Ex.

Unintended value

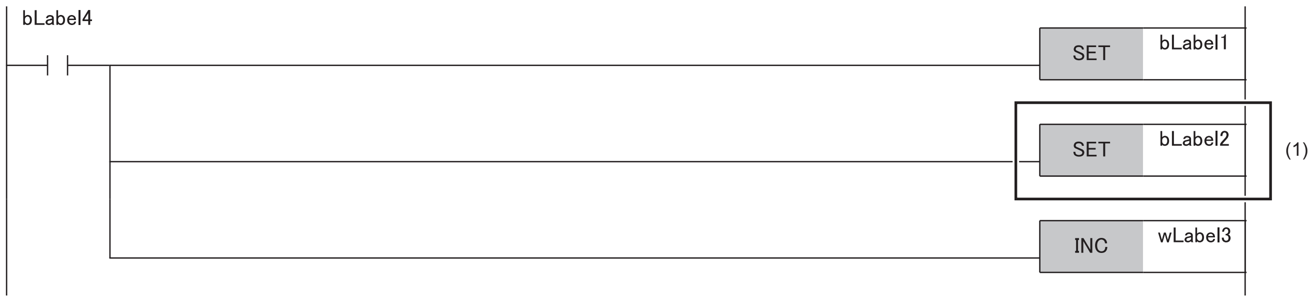
```
MacroFbPou_1 (EN := M0, ENO => M1);
M2 := MacroFbPou_1.ENO;
```

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

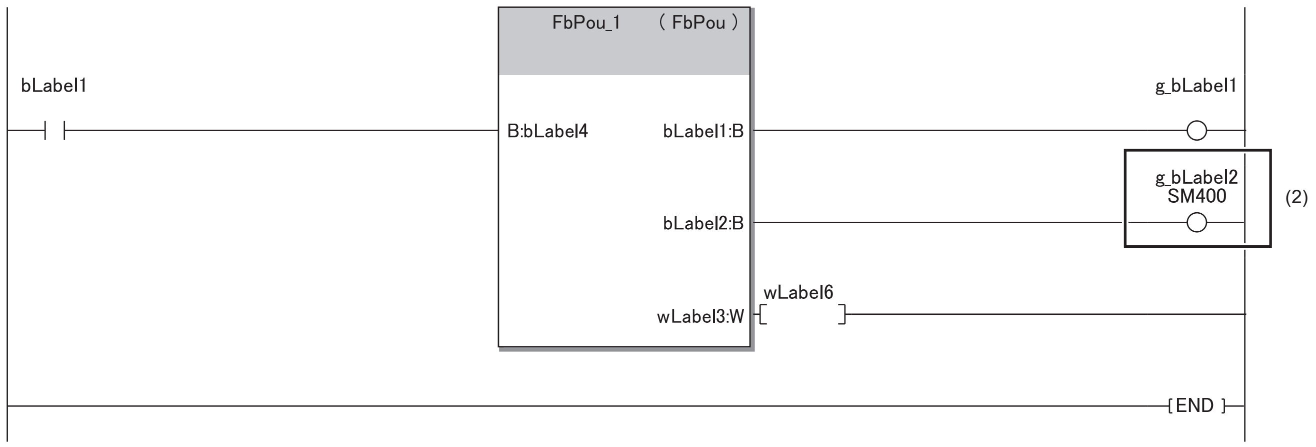
A program block that is the calling source of the function block or the function block may cause the error. In this case, check the inputs and outputs of the program block that is the calling source of the function block and the function block.

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.

■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.
- 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 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.)

■When a master control instruction is used

Shown here is the operation when the master control is OFF.

- Macro type function block

The operation in the function block is the same as contact OFF (OFF execution or no execution is made).

- Subroutine type function block and function

No execution is made in the function block.

Changing program capacity (FX5U/FX5UC CPU module only)

When the program capacity setting parameter is changed to 128000 steps from 64000 steps, the operation changes as follows.

- Signal flow area for FB is expanded from 16K bytes to 32K bytes.
- Temporary area capacity is expanded form 700 words to 32767 words.
- Execution time for each instruction is prolonged.

Do not write a program with more than 64000 steps to the CPU module firmware version earlier than 1.100. The program does not operate normally.

For the program capacity setting, refer to the following.

📖MELSEC iQ-F FX5 User's Manual (Application)

4 LABELS

Labels are variables for I/O data or internal processing, specified by a character string.
Users can create a program without considering devices or buffer memory size by using labels.
Thus, a program, where labels are used, can be reused in a system with a different module configuration easily.
When labels are used, there are some precautions on programming and functions used. For details, refer to the following.
☞ Page 40 Precautions

4.1 Type

This manual describes the following types of label.
- Global labels
- Local labels

Global labels

Global labels are labels that can be shared by programs in a project. Global labels can be used in all the programs in a project.
Global labels can be used in program blocks and function blocks.
When setting a global label, set the label name, class and data type, and assign a device.

■Device assignment

Devices can be assigned to global labels.

Item	Description
Label to which no device is assigned	• Programming without concern to 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	• If a device is to be programmed as a label referring to 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

Local labels are labels that can be used in each POU only. Local labels that are not included in POUs cannot be used.
When setting a local label, set the label name, class, and data type.

Point

There are other types of labels available in addition to global labels and local labels.
■System labels
System labels can be shared among iQ Works-compatible products and are managed by MELSOFT Navigator. Global labels registered as system labels can be monitored or accessed using the system labels on GOT.
For details, refer to the following.
📖iQ Works Beginner's Manual
■Module labels
Module labels are labels defined uniquely by each module. Module labels are automatically generated by the engineering tool from the module used, and can be used as a global label.
For details, refer to the following.
📖MELSEC iQ-F FX5 CPU Module Function Block Reference
For registration of module labels, refer to the following.
📖GX Works3 Operating Manual

4.2 Class

The label class indicates how each label can be used from which POU.

The selectable class varies depending on the POU.

Global label				
Class	**Description**	**Applicable POU**		
		Program block	**Function block**	**Function**
VAR_GLOBAL	Common label that can be used in program blocks and function blocks	○	○	×
VAR_GLOBAL_CONSTANT	Common constant that can be used in program blocks and function blocks	○	○	×
VAR_GLOBAL_RETAIN	Latch type label that can be used in program blocks and function blocks	○	○	×

Local label				
Class	**Description**	**Applicable POU**		
		Program block	**Function block**	**Function**
VAR	Label that can be used within the range of declared POUs This label cannot be used in other POUs.	○	○	○
VAR_CONSTANT	Constant that can be used within the range of declared POUs This label cannot be used in other POUs.	○	○	○
VAR_RETAIN	Latch type label that can be used within the range of declared POUs This label cannot be used in other POUs.	○	○	×
VAR_INPUT	Label that inputs to a function or a function block. This label receives a value, and cannot be changed in POUs.	×	○	○
VAR_OUTPUT	Label that outputs a value from a function or a function block	×	○	○
VAR_OUTPUT_RETAIN	Latch type label that outputs a value from a function or a function block	×	○	×
VAR_IN_OUT	Local label which receives a value, outputs it from a POU, and can be changed in POUs	×	○	×
VAR_PUBLIC	Label that can be accessed from other POUs	×	○	×
VAR_PUBLIC_RETAIN	Latch type label that can be accessed from other POUs	×	○	×

4.3 Data Type

Labels are classified into several data types according to the bit length, processing method, or value range.

The following two data types are provided.

- Elementary data type
- Generic data type (ANY)

Elementary data type

The following data types are available as the elementary data type.

Data type		Description	Value range	Bit length
Bit	BOOL	Represents binary status, such as ON or OFF	0 (FALSE), 1 (TRUE)	1-bit
Word [Unsigned]/Bit String [16-bit]	WORD	Represents 16-bit	0 to 65535	16-bit
Double Word [Unsigned]/Bit String [32-bit]	DWORD	Represents 32-bit	0 to 4294967295	32-bit
Word [Signed]	INT	Handles positive and negative integer values	-32768 to +32767	16-bit
Double Word [Signed]	DINT	Handles positive and negative double word integer values	-2147483648 to +2147483647	32-bit
FLOAT [Single Precision]	REAL	Handles the portion after the decimal point of the float (single precision) Effective digits: 7 (after the decimal point: 6)	-2^{128} to -2^{-126}, 0, 2^{-126} to 2^{128}	32-bit

Data type		Description	Value range	Bit length
Time[*1]	TIME	Handles values as d (day), h (hour), m (minute), s (second), or ms (millisecond)	T#-24d20h31m23s648 ms to T#24d20h31m23s647 ms[*2]	32-bit
String(32)	STRING	Handles a character string (ASCII, Shift JIS)	Up to 255 letters (half-width character)	Variable
String [Unicode](32)	WSTRING	Handles a Unicode character string	Up to 255 letters	Variable
Timer	TIMER	Structure that corresponds to a timer (T) of a device	☞ Page 33 Data types of timers and counters	
Retentive Timer	RETENTIVETIMER	Structure that corresponds to a retentive timer (ST) of a device		
Counter	COUNTER	Structure that corresponds to a counter (C) of a device		
Long Counter	LCOUNTER	Structure that corresponds to a long counter (LC) of a device		
Pointer	POINTER	Type that corresponds to a pointer (P) of a device (MELSEC iQ-F FX5 User's Manual (Application))		

*1 The time data is used in the time data type function of standard functions. For the standard function, refer to the following.
MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
*2 When using a constant for a label of the time data, prefix "T#" to the label.

■Data types of timers and counters

The data types of a timer, retentive timer, counter, and long counter are structures that have contacts, coils, and current values.

Data type		Member name	Data type of member	Description	Value range
Timer	TIMER	S	Bit	Indicates contacts. The operation is the same as the contact of a timer device (TS).	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates coils. The operation is the same as the coil of a timer device (TC).	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/Bit String [16-bit]	Indicates a current value. The operation is the same as the current value of a timer device (TN).	0 to 32767[*1]
Retentive Timer	RETENTIVETIMER	S	Bit	Indicates contacts. The operation is the same as the contact of a retentive timer device (STS).	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates coils. The operation is the same as the coil of a retentive timer device (STC).	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/Bit String [16-bit]	Indicates a current value. The operation is the same as the current value of a retentive timer device (STN).	0 to 32767[*1]
Counter	COUNTER	S	Bit	Indicates contacts. The operation is the same as the contact of a counter device (CS).	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates coils. The operation is the same as the coil of a counter device (CC).	0 (FALSE), 1 (TRUE)
		N	Word [unsigned]/Bit String [16-bit]	Indicates a current value. The operation is the same as the current value of a counter device (CN).	0 to 32767
Long Counter	LCOUNTER	S	Bit	Indicates contacts. The operation is the same as the contact of a long counter device (LCS).	0 (FALSE), 1 (TRUE)
		C	Bit	Indicates coils. The operation is the same as the coil of a long counter device (LCC).	0 (FALSE), 1 (TRUE)
		N	Double Word [unsigned]/ Bit string [32-bit]	Indicates a current value. The operation is the same as the current value of a long counter device (LCN).	[*2]

*1 The unit of the current value is specified by instruction name.
*2 When use a long counter in the OUT LC instruction: 0 to 4294967295
When use a long counter in the UDCNTF instruction: -2147483648 to +2147483647

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

MELSEC iQ-F FX5 User's Manual (Application)

The specification method of each member is the same as the member specification of the structure data type. (☞ Page 37 Structures)

Generic data type (ANY)

The generic data type indicates data type of a label which combines several basic data types. The data type name begins with "ANY".

The generic data type is used when multiple data types are available in arguments or return values etc. of a function of a function block.

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

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

📖MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

Definable data types

The following tables list the definable data types possibilities for each label class.

Global label

Class	Definable data type
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
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
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
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.

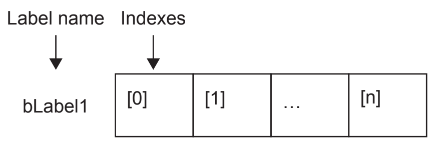
4.4 Arrays

An array represents a consecutive accumulation of the same data type labels, under the same name.

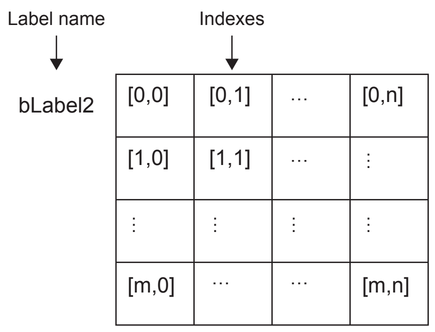
Arrays can be defined by the elementary data types or structures.

The maximum number of arrays differs depending on the data types.

■One-dimensional array



■Two-dimensional array



Definition of 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 36 Maximum number of array elements

■Definition format

The following table lists definition format examples up to three dimensions.

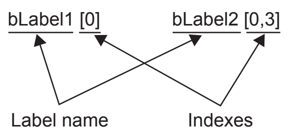
The range from the array start value to the array end value is the number of elements.

Number of array dimensions	Format	Remarks
One dimension	Array of elementary data type/structure name (array start value .. array end value)	• For elementary data types: ☞ Page 32 Elementary data type • For structured data types: ☞ Page 37 Structures
	(Definition example) Bit (0..2)	
Two dimensions	Array of elementary data type/structure name (array start value .. array end value, array start value .. array end value)	
	(Definition example) Bit (0..2, 0..1)	
Three dimensions	Array of elementary 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..3)	

How to use arrays

To identify individual labels of an array, append an index enclosed by "[ ]" after the label name.

For an array with two or more dimensions, delimit indexes in "[ ]" by using "comma (,)".



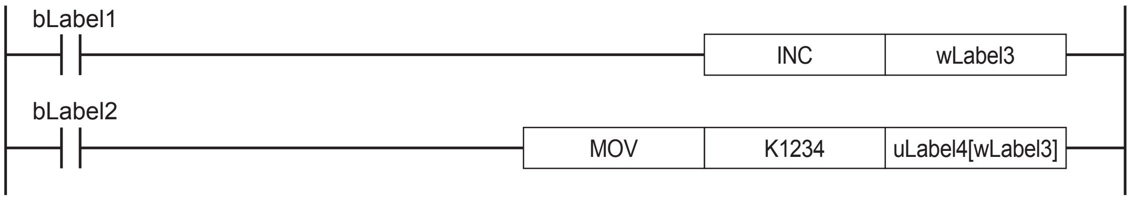
Type	Specification example	Remarks
Constant	bLabel1[0]	An integer equal to or greater than 0 can be specified. Decimal constant or hexadecimal constant can be specified.
Device	bLabel1[D0]	A word device or double-word device can be specified.
Label	bLabel1[uLabel2]	The following data types can be specified. • 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.

Precautions

When a bit of a device/label (example: D0.0) is assigned to bit array in global label, labels and devices can not be used for the array index in programming (example: bLabel1[D0] cannot be programmed).

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.



- In the case of the ladder diagram, arrays can be used with element numbers omitted. When the element number is omitted, it is converted to the starting number of the array element. For example, when the label name you define is "boolAry" and the data type is "bit (0..2,0..2)", then "boolAry[0,0]" and "boolAry" are treated in the same way.
- A multidimensional array can be specified as setting data of an instruction, function, or function block using arrays. In that case, the rightmost element in the multidimensional array is treated as the first dimension.

Maximum number of array elements

The maximum number of array elements differs depending on data types.

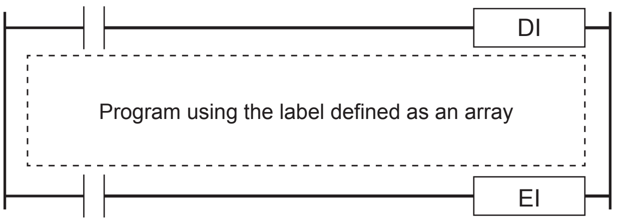
Data type	Setting range
Bit Word [Unsigned]/Bit String [16-bit] Double Word [Unsigned]/Bit String [32-bit] Word [Signed] Double Word [Signed] FLOAT [Single Precision] Time Timer Retentive Timer Counter Long Counter	1 to 32768
String(32)	1 to 32768 ÷ character string length
String [Unicode](32)	1 to 16384 ÷ character 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-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

■Array elements

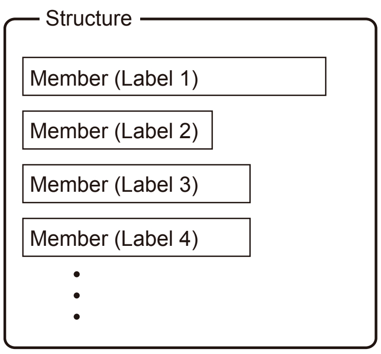
When accessing the element defined in an array, access it within the range of the number of elements.
If a constant out of the range defined for the array index is specified, a compile error will occur.
If the array index is specified with data other than a constant, a compile error will not occur. The processing will be performed by accessing another label area or latch label area.

4.5 Structures

A structure is a data type that includes different labels. Structures can be used in all POUs.
Each member (label) included in a structure can be defined even when the data types are different.

Creating structures

To create a structure, first create the configuration of the structure, and define members for the created structure.

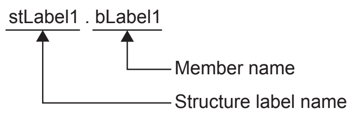


How to use structures

To use structures, register the label with the defined structure as a new data type.
To specify each member, append an element name after the structure label name with "period (.)" as a member name.

Ex.

When using the member of a structure



Point

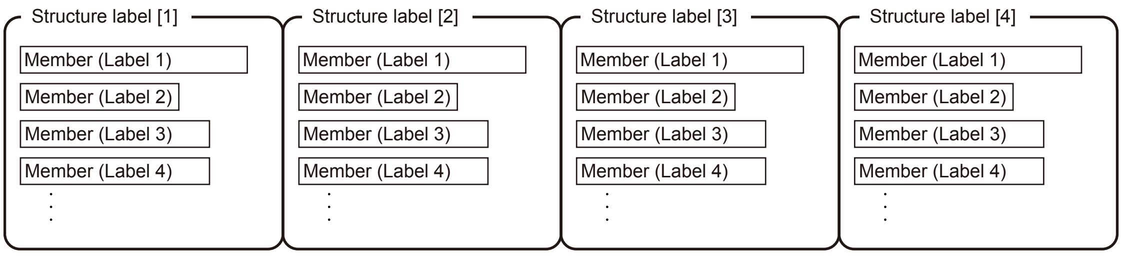
- When labels are registered by defining multiple data types in a structure and used in a program, the order the data is stored after converted is not the order the data types were defined. When programs are converted using the engineering tool, labels are classified into type and data type, and then assigned to the memory (memory assignment by packing blocks).

📖GX Works3 Operating Manual

- If a member of a structure is specified in an instruction operand that uses control data (series of consecutive devices from the operand used by the instruction), the control data is assigned to members of the structure by the order they are stored in memory, not the order the members are defined.

Arrays of structures

Structures can also be used as arrays.

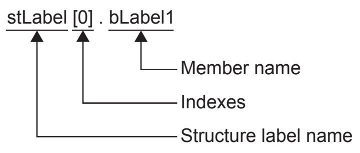


When a structure is declared as an array, append an index enclosed by "[ ]" after the structure label name.

The array of structure can be specified as arguments of functions and function blocks.

Ex.

When using an element of the structured array



Data types that can be specified

The following data types can be specified as a member of a structure.

- Elementary data type
- Pointer type
- Arrays
- Other structures

Structure types

The following data types are defined as a structure beforehand.

Type	Reference
Timer type	☞ Page 32 Data Type
Retentive Timer type	
Counter type	
Long Counter type	

4.6 Constant

Types of constants

The following table shows the expressions for setting a constant to a label.

Applicable data type	Type	Expression	Example
Bit	Boolean data	Input "TRUE" or "FALSE".	TRUE, FALSE
	Binary	Append "2#" in front of a binary number.	2#0, 2#1
	Octal	Append "8#" in front of an octal number.	8#0, 8#1
	Decimal	Directly input a decimal number, or append "K" in front of a decimal number.	0, 1, K0, K1
	Hexadecimal	Append "16#" or "H" in front of a hexadecimal number.	16#0, 16#1, H0, H1
• Word [Unsigned]/Bit String [16-bit] • Double Word [Unsigned]/Bit String [32-bit] • Word [Signed] • Double Word [Signed]	Binary*1	Append "2#" in front of a binary number.	2#0010, 2#01101010, 2#1111_1111
	Octal*1	Append "8#" in front of an octal number.	8#0, 8#337, 8#1_1
	Decimal*1	Directly input a decimal number or append "K" in front of a decimal number.	123, K123, K-123, 12_3
	Hexadecimal*1	Append "16#" in front of a hexadecimal number. Or append "H" in front of a value.	16#FF, HFF, 16#1_1
FLOAT [Single Precision]	Real number*1	Directly input a real number, or append "E" in front of a real number.	2.34, E2.34, E-2.34, 3.14_15
	Real number (exponent expression)	Append "E" in front of an exponent expression or a real number. Append "+" in front of exponent part.	1.0E6, E1.001+5
String(32)	Character string	Enclose a character string (ASCII, Shift JIS) with single quotations (').	'ABC'
String [Unicode](32)	Unicode character string	Enclose a Unicode character string in double quotation marks (").	"ABC"
Time	Time	Append "T#" in front.	T#1h, T#1d2h3m4s5ms

*1 In the binary notation, the octal notation, the decimal notation, the hexadecimal notation, and the real number notation, values can be delimited by an underscore (_) to make programs easy to read. (In the program processing, underscores are ignored.)

When "$" is used in character string type data

"$" is used as an escape sequence. Two hexadecimal numbers after "$" are recognized as an ASCII code, and characters corresponding to the ASCII code are inserted in the character string. If no ASCII code for the two hexadecimal numbers after "$" exists, a conversion error occurs. However, when any of the following characters is described after "$", no error occurs.

Expression	Symbol that is used in character string, or printer code
$$	$
$'	'
$"	"
$L or $l	Line feed
$N or $n	Newline
$P or $p	Page (form feed)
$R or $r	Return
$T or $t	Tab

4.7 Precautions

Functions with limitations

In the following functions, there is a limitation on label use.

Item	Description
Trigger of an event execution type program	Labels cannot be used. Consider taking the following measures. • Use devices. • Define a label to be used as a global label and assign devices to the global label.
Intelligent function module refresh setting	Labels cannot be used. Consider taking the following measures. • Use devices.

■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, reserve device area capacity. (The label area is not consumed.)

1. Reserve the device area to be used.

CPU Parameter ⇨ Memory/Device Setting ⇨ Device/Label Memory Area Capacity 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

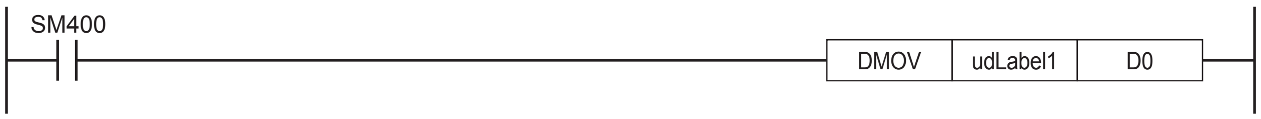
Copy the label data into a specified device following the procedure below, and use the copy-target device.

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

1. Reserve the device area to be used.

CPU Parameter ⇨ Memory/Device Setting ⇨ Device/Label Memory Area Capacity 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

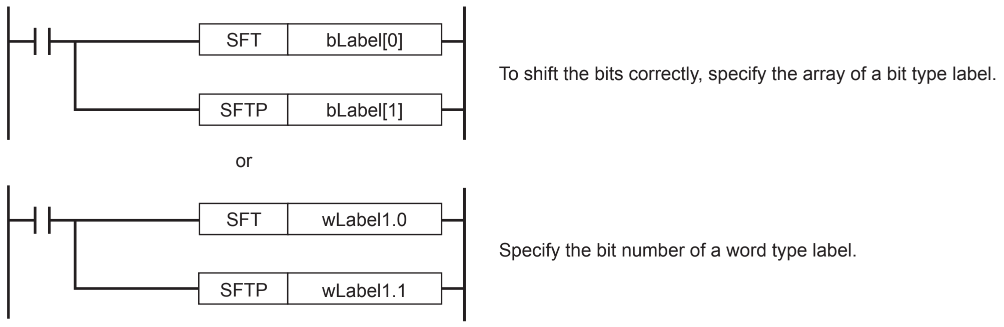
When copying a value of a label to another device by a transfer instruction, note that the number of program steps increases. In addition, when adding a transfer instruction on a program, consider execution timing of the function to be used.

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.

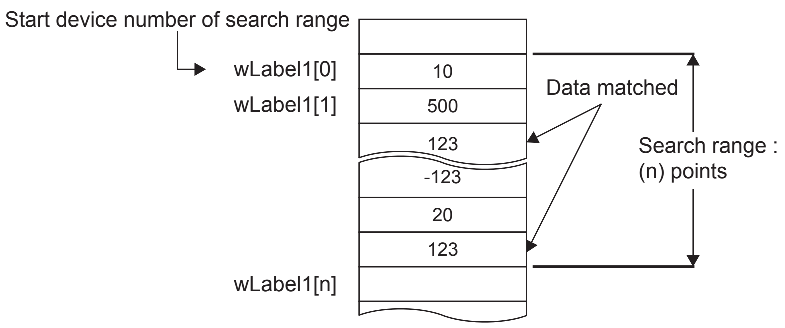
SFT(P) instruction



SFT bLabel[0]
SFTP bLabel[1]
To shift the bits correctly, specify the array of a bit type label.
or
SFT wLabel1.0
SFTP wLabel1.1
Specify the bit number of a word type label.

Ex.

SFR(P) instruction



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

Limitations on label names

Label names have the following limitations:

- A label name must start with a nonnumeric character or underscore (_). It cannot start with a number.
- Reserved words cannot be used as label names.

For details of reserved words, refer to the following.

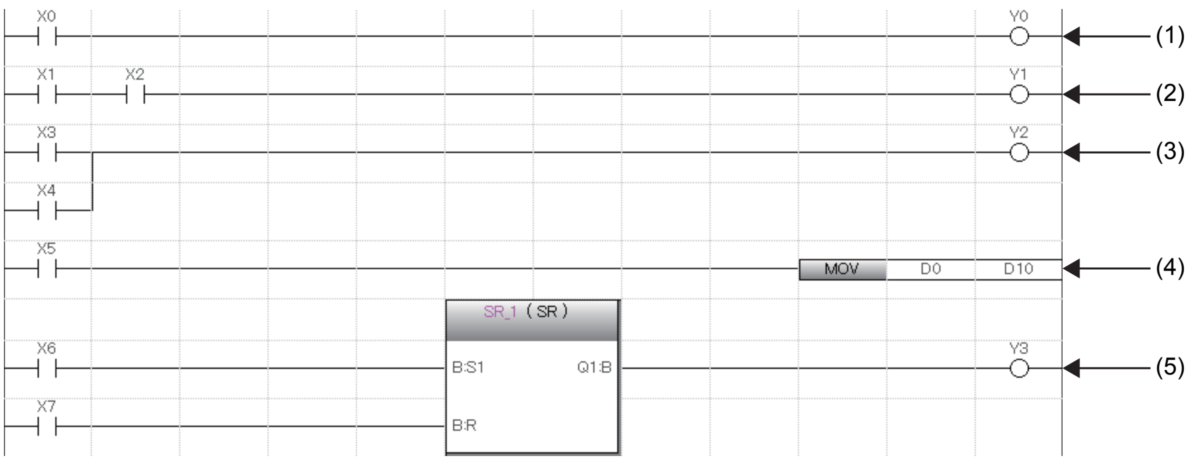
GX Works3 Operating Manual

5 LADDER DIAGRAM

Ladder diagram is a language that describes the sequence control by indicating logical operations consisting of "AND" or "OR" with combinations of series connections and parallel connections in a ladder consisting of contacts and coils.

5.1 Configuration

With the ladder diagram, the following ladder can be created.



(1) A ladder consists of contacts and coils

(2) A ladder connected in series

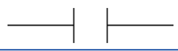
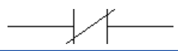
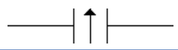
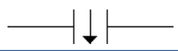
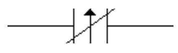
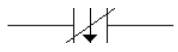
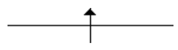
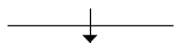
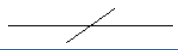
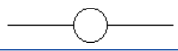
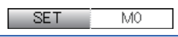
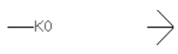
(3) A ladder connected in parallel

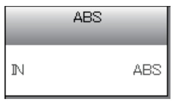
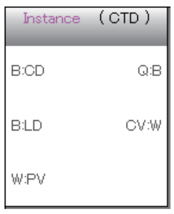
(4) A ladder where instructions are used

(5) A ladder where standard functions and function blocks are used

Ladder symbols

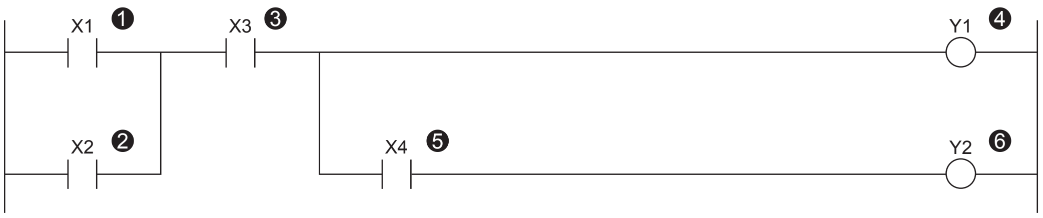
This section shows ladder symbols that can be used for programming in the ladder diagram.

Element	Symbol	Description
NO contact		Turns on when a specified device or label is ON.
NC contact		Turns on when a specified device or label is OFF.
Rising edge		Turns on at the rising edge (OFF to ON) of a specified device or label.
Falling edge		Turns on at the falling edge (ON to OFF) of a specified device or label.
Negated rising edge		Turns on when a specified device or label is OFF or ON, or at the falling edge (ON to OFF) of a specified device or label.
Negated falling edge		Turns on when a specified device or label is OFF or ON, or at the rising edge (OFF to ON) of a specified device or label.
Conversion of operation result to leading edge pulse		Turns on at the rising edge (OFF to ON) of an operation result. Turns off when the operation result is other than the rising edge.
Conversion of operation result to trailing edge pulse		Turns on at the falling edge (ON to OFF) of an operation result. Turns off when the operation result is other than the falling edge.
Inverting the operation result		Inverts the operation just before this instruction.
Coil		Outputs an operation result to a specified device or a label.
Instruction		Executes an instruction specified in [ ].
Turn-back		Turns back a circuit by creating a turn source symbol and a turn destination symbol when the number of contacts exceeds the number of contacts that can be created in one line.

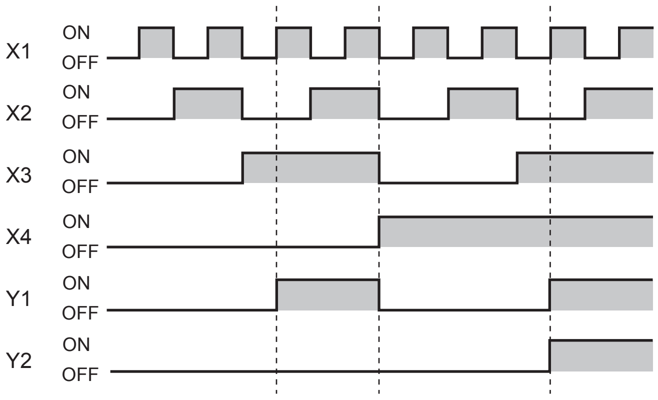
Element	Symbol	Description
Function		Executes a function. • How to create functions (GX Works3 Operating Manual) • Standard function (MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks))
Function block		Executes a function block. • How to create function blocks (GX Works3 Operating Manual) • Standard function blocks (MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)) • Module function blocks (MELSEC iQ-F FX5 CPU Module Function Block Reference)

Program execution order

The program is executed in order of the following numbers.



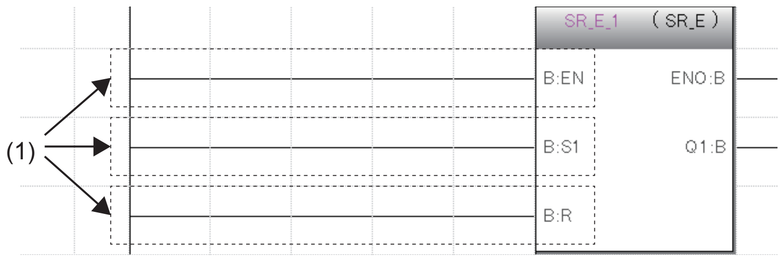
When executing the program above, Y1 and Y2 turn on corresponding to turning ON or OFF of X1 to X4 as shown below.



Precautions for using a function block in ladder diagram

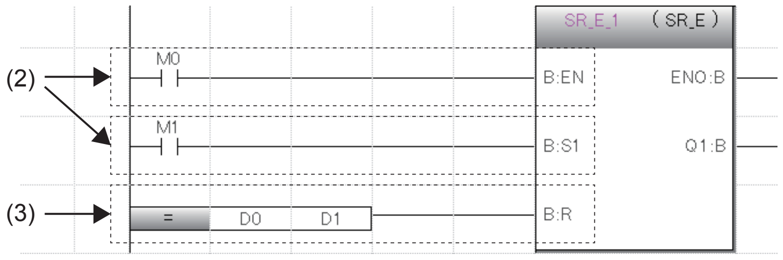
Precautions for directly connecting to an FB instance from the left rail

When EN and input variables (bit type) are directly connected to the left rail in the input circuit of the FB instance, the on/off state does not change.



(1): The on/off state does not change.

To change the on/off states of EN and the input variables (bit type), use a contact or an instruction equivalent to the contact.



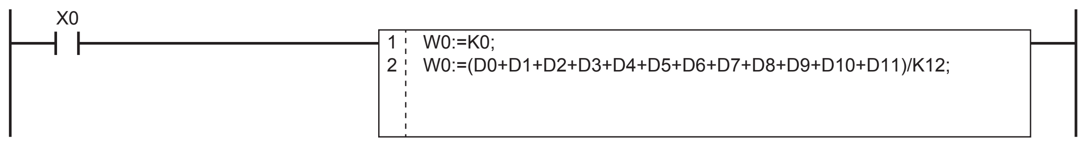
(2): Contact

(3): Instruction equivalent to a contact

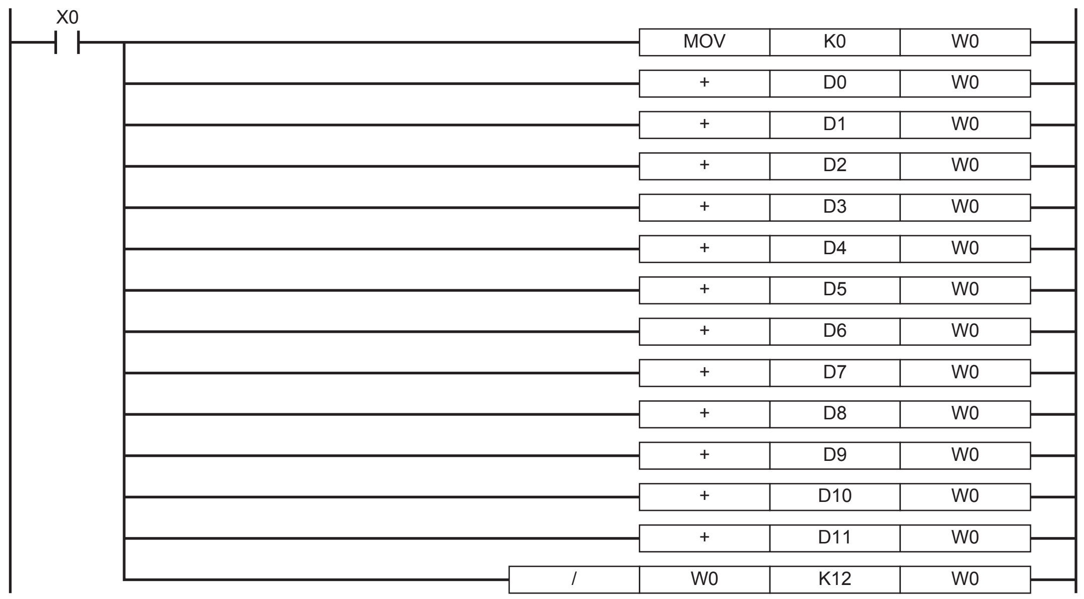
5.2 Inline ST

Inline ST is a function that creates, edits and monitors inline ST box that displays an ST program in a cell of an instruction that is equivalent to a coil in the ladder editor.
Numerical operations or character string operations can be created easily in a ladder program.

- Program with the inline ST



- Program without the inline ST



Restriction

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

Specifications

For the specifications of the inline ST, refer to the ST language specifications.
☞ Page 47 ST LANGUAGE

Precautions

- Only one inline ST can be created in one line of a ladder program.
- Creating both a function block and an inline ST box in one line of a ladder program is impossible.
- Creating an inline ST box in a position of an instruction that is equivalent to a contact creates an inline ST box in a position of an instruction that is equivalent to a coil.
- The maximum number of characters that can be input in an inline ST is 2048. (A newline is counted as two characters.)
- In inline ST, do not use rising execution instructions, falling execution instructions, special timer instructions, OUT instruction, positioning instructions, or standard function blocks (edge detection function blocks and counter function blocks) as they may not work property.
- When the RETURN syntax is used in an inline ST, the processing inside the inline ST box ends, and the processing inside the program block does not end.

5.3 Statements and Notes

In a ladder program, statements and notes can be displayed.

Statements

By using statements, users can append comments to circuit blocks. Appending statements makes the processing flow easy to understand.

Statements include line statements, P statements, and I statements.

A line statement can be displayed on a tree view of the Navigation window.

■Line statement

A comment is appended to a ladder block as a whole.

■P statement

A comment is appended to a pointer number.

■I statement

A comment is appended to an interrupt pointer number.

Notes

By using notes, users can append comments to coils and instructions in a program.

Appending notes makes the details of coils and application instructions easy to understand.

Types of statements and notes

"PLC" and "Peripheral" are the types of statements and notes.

Type	Type	Description
PLC	• Line statement • P statement • I statement • Note	Statements and notes can be stored on the CPU module. PLC statement uses the following number of steps. (Decimal fraction is rounded down.) • Without character: 3 steps • With character: 4 + (Number of characters + 2 + 14) / 15 + Number of characters (steps)
Peripheral	• Line statement • P statement • I statement • Note	Strings of the description are not embedded in the program, but they are saved as attached information of the program. One statement or note line uses one step. A * symbol is prefixed to the entered text automatically.

6 ST LANGUAGE

The ST language is one of the languages supported by IEC 61131-3, the international standard that defines the description methods for logic. ST language is a text programming language with a grammatical structure similar to C language. ST language is suitable for programming some complicated processing that cannot be easily described using ladder diagram. ST language supports control syntaxes, operational expressions, function blocks (FBs), and functions (FUNs). Therefore, the following description can be made.

Ex.

Control syntaxes using selective branches with conditional statements and repetition by iteration statements

```
(* Control conveyors of 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.

Expressions using operators (such as *, /, +, -, <, >, and =)

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

Ex.

Calling a defined function block

```
//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);
```

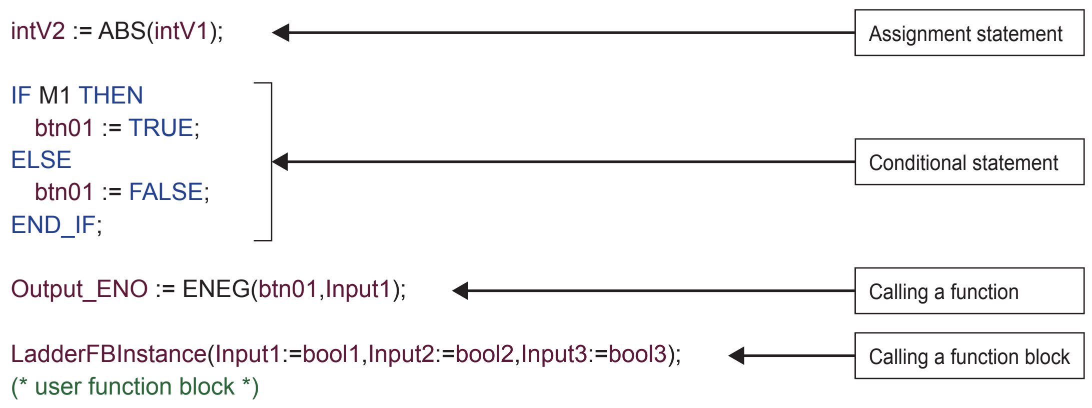
Ex.

Calling a standard function

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

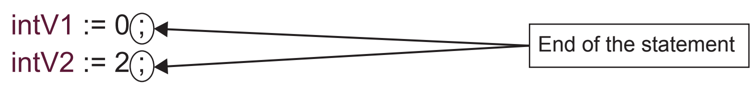
6.1 Configuration

Operators and syntaxes are used for programing in ST language.



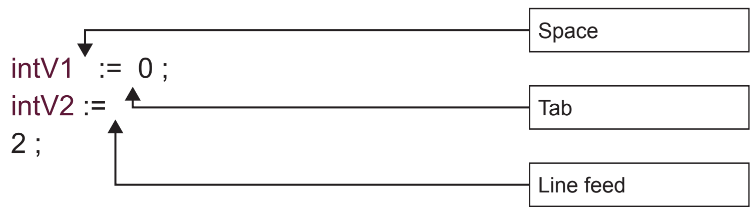
```
intV2 := ABS(intV1);

IF M1 THEN
   btn01 := TRUE;
ELSE
   btn01 := FALSE;
END_IF;

Output_ENO := ENEG(btn01,Input1);

LadderFBInstance(Input1:=bool1,Input2:=bool2,Input3:=bool3);
(* user function block *)
```

A statement must end with ";" (semicolon).



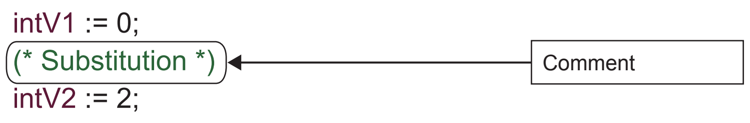
```
intV1 := 0;
intV2 := 2;
```

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



```
intV1  :=  0 ;
intV2 :=
2 ;
```

Comments can be inserted in a program.



```
intV1 := 0;
(* Substitution *)
intV2 := 2;
```

Constituent elements of a program

A ST program consists of the following elements.

Item		Example	Reference
Delimiter		;, ( )	☞ Page 49 Delimiter
Operator		+, -, <, >, =	☞ Page 49 Operator
Reserved word	Syntax	IF, CASE, WHILE, RETURN	☞ Page 50 Syntax
	Device	X0, Y10, M100	📖MELSEC iQ-F FX5 User's Manual (Application)
	Data type	BOOL, DWORD	☞ Page 32 Data Type
	Function	ADD, REAL_TO_STRING_E	📖MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)
Constant		123, 'abc'	☞ Page 58 Constant
Label		Switch_A	☞ Page 59 Label and device
Comment		(* Turn ON *), //Turn ON, /*Turn ON*/	☞ Page 60 Comment
Other symbols		One-byte space, line feed code, tab code	—

- Use one-byte delimiters, operators, and reserved words.
- For details of reserved words, refer to the following.

📖GX Works3 Operating Manual

Delimiter

The following delimiters are provided in ST language for clarifying the program structure.

Symbol	Description
()	Parenthesized
[]	Specification of an array element
. (period)	Specification of members of the structure or function block
, (comma)	Separation of arguments
: (colon)	Device type specifier
; (semicolon)	End of a sentence
' (single quotation mark)	Description of a character string (ASCII, Shift JIS)
" (double quotation mark)	Description of a Unicode character string
.. (two periods)	Specification of an integer range

Operator

The following shows the operators used in a ST program and the target data types and operation result data types for each operator.

Operator	Target 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 character string cannot be specified.

The following table shows the priorities of the operators.

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

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

Syntax

The following table shows the types of statements that can be used in a ST program.

Item	Description	Reference
Assignment statement	Assignment statement	☞ Page 50 Assignment statement
Sub-program control statement	Function block call statement, function call statement	☞ Page 51 Sub-program control statement
	RETURN statement	
Conditional statement	IF statement (IF, IF...ELSE, IF...ELSIF)	☞ Page 52 Conditional statement
	CASE statement	
Iteration statement	FOR statement	☞ Page 53 Iteration statement
	WHILE statement	
	REPEAT statement	
	EXIT statement	

Write statements using half width characters.

Assignment statement

Format	Description	Example

<Left side> := <Right side> ;	The assignment statement assigns the result of the right side expression to the label or device of the left side.
	The result of the right side expression and the data type of the left side need to be the same data type.
		intV1 := 0;
		intV2 := 2;

When an array type label or a structure label is used, check the data types of the left side and right side of the assignment statement.

When an array type label is used, the data type and the number of elements need to be the same for the left side and right side. Do not specify elements.

Ex.

intAry1 := intAry2;

When a structure label is used, the data type needs to be the same for the left side and right side.

Ex.

dutVar1 := dutVar2;

■Automatic conversion of data types

In the ST language, if a different data type is assigned or a different arithmetic operation is described, the data type may be automatically converted.

Ex.

Example of automatic conversion

```
dintLabel1 := intLabel1;
// Assignment statement : Automatically convert the INT type variable (intLabel1) to a DINT type variable,
   and assign it the DINT type variable (dintLabel1).

dintLabel1 := dintLabel2 + intLabel1;
// Arithmetic operation expression : Automatically convert the INT type variable (intLabel1) to a DINT type
   variable, and perform DINT type addition.
```

Type conversion is performed in an assignment statement, input argument pass to a function block and function (VAR_INPUT part), and an arithmetic operation.

To avoid the deletion of the data during type conversion, only conversion from smaller type to larger type is performed. Of the elementary data types, type conversion is performed only for the following data types among basic data types are the targets of a type conversion.

Data type	Description
Word [Signed]	In the case of a double word [signed] after conversion, the conversion is automatically made into a value with a sign extension. In the case of a single-precision real, an automatic conversion is made into the same value as the integer before the conversion.[*1]
Word [Unsigned]/Bit String [16-bit]	In the case of a double word [unsigned]/bit string [32 bits] or a double word [signed] after conversion, an automatic conversion is made into to a value with a zero extension.[*2] In the case of a single-precision real, an automatic conversion is made into the same value as the integer before the conversion.[*1]

*1 When the data of 16 bits (a word [signed] or a word [unsigned]/bit string [16 bits]) is transferred to an input argument of the data type ANY_REAL, an automatic conversion is made into a single- precision real.

*2 When the data of a word [unsigned]/bit string [16 bits] is transferred to an input argument of ANY32, an automatic conversion is made into a double word [unsigned]/bit string [32 bits].

For data types that are not described above, use the type conversion function.

Since type conversion is not performed in the following cases, use the type conversion function.

- Type conversion between integer-data types with different signs
- Type conversion between the data types by which the data is deleted

For the precautions for assigning the result of an arithmetic operation, refer to the following.

☞ Page 54 When an assigned arithmetic operation is used

Sub-program control statement

■Function block call statement

Format	Description
Instance name(Input variable1:= Variable1, ... Output variable1: => Variable2, ...);	Enclose the assignment statement that assigns variables to the input variable and output variable by "( )" after the instance name. When using multiple variables, delimit the assignment statement by "," (comma).

Instance name.Input variable1:= Variable1;
:
Instance name();
Variable2:= Instance name.Output variable1;
	List the assignment statement that assigns variables to the input variable and output variable before and after a function block call statement.

The following table shows the symbols used for arguments in a function block call statement and available formats.

Type	Description	Attribute	Symbol	Available formats
VAR_INPUT	Input variable	N/A, or RETAIN	:=	All formats
VAR_OUTPUT	Output variable	N/A, or RETAIN	=>	Variable only
VAR_IN_OUT	Input/Output variable	N/A	:=	Variable only

The execution result of the function block is stored by assigning the output variable that is specified by adding "." (period) after the instance name to the variable.

Function block	FB definition	Example

Calling a function block with 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;

Calling a function block with 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 an argument by "()" after the function name. When using multiple arguments, delimit them by "," (comma).

Assigning to variables stores the execution result of the function.

Function	Example
Calling a function with one input variable (Example: ABS)	Outout1 := ABS(Input1);
Calling a function with three input variables (Example: MAX)	Outout1 := MAX(Input1, Input2, Input3);
Calling a function with EN/ENO (standard functions) (Example: MAX_E)	Output1 := MAX_E(boolEN, boolENO, Input1, Input2, Input3);
Calling other than standard functions (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 that does not return a value and a function that includes a VAR_OUTPUT variable in the argument of a call statement can be executed as a statement by adding a semicolon (;) at the end.

■RETURN statement

Syntax	Format	Description	Example

■RETURN	RETURN;	The RETURN statement is used to end a program, function block, or function in the middle of processing.
		When the RETURN statement is used in a program, the processing jumps to the next step after the last line of the program.
		When the RETURN statement is used in a function block, the processing is returned from the function block.
		When the RETURN statement is used in a function, the processing is returned from the function.
		One pointer type label is used by the system for one RETURN statement.
			IF bool1 THEN
			    RETURN;
			END_IF;

A user-defined function that does not return a value and a function that includes a VAR_OUTPUT variable in the parameter of a call statement can be executed as a statement by adding a semicolon (;) at the end.

Conditional statement

Syntax	Format	Description	Example

■IF	IF <Boolean expression> THEN
	<Statement···> ;
	END_IF;
		The statement is executed when the value of Boolean expression (conditional expression) is TRUE. The statement is not executed if the value of Boolean expression is FALSE.
		Any expression that returns TRUE or FALSE as the result of the Boolean operation with a single bit type variable status, or a complicated expression that includes many variables can be used for the Boolean expression.
			IF bool1 THEN
			    intV1:=intV1+1;
			END_IF;

■IF...ELSE	IF <Boolean expression> THEN
	<Statement 1···> ;
	ELSE
	<Statement 2···> ;
	END_IF;
		Statement 1 is executed when the value of Boolean expression (conditional expression) is TRUE.
		Statement 2 is executed when the value of Boolean expression is FALSE.
			IF bool1 THEN
			    intV3:=intV3+1;
			ELSE
			    intV4:=intv4+1;
			END_IF;

■IF...ELSIF	IF <Boolean expression 1> THEN
	<Statement 1···> ;
	ELSIF <Boolean expression 2> THEN
	<Statement 2···> ;
	ELSIF <Boolean expression 3> THEN
	<Statement 3···> ;
	END_IF;
		Statement 1 is executed when the value of Boolean expression (conditional expression) 1 is TRUE. Statement 2 is executed when the value of Boolean expression 1 is FALSE and the value of Boolean expression 2 is TRUE.
		Statement 3 is executed when the value of Boolean expression 1 and 2 are FALSE and the value of Boolean expression 3 is TRUE.
			IF bool1 THEN
			    intV1:=intV1+1;
			ELSIF bool2 THEN
			    intv2:=intV2+2;
			ELSIF bool3 THEN
			    intV3:=intV3+3;
			END_IF;

Syntax	Format	Description	Example

■CASE	CASE <Integer expression> OF
	<Integer selection 1> :
	<Statement 1···> ;
	<Integer selection 2> :
	<Statement 2···> ;
	⋮
	<Integer selection n> :
	<Statement n···> ;
	ELSE
	<Statement n+1···> ;
	END_CASE;
		When the statement that has the integer selection value that matches with the value of the integer expression (conditional expression) is executed, and if no integer selection value matches with the expression value, the statement that follows the ELSE statement is executed.
		The CASE statement is used to execute a conditional statement based on a single integer value or an integer value as the result of a complicated expression.
			CASE intV1 OF
			  1:
			  bool1:=TRUE;
			  2:
			  bool2:=TRUE;
			ELSE
			  intV1:=intV1+1;
			END_CASE;

Iteration statement

Syntax	Format	Description	Example

■FOR	FOR <Repeat variable initialization>
	TO <Last value>
	BY <Incremental expression> DO
	<Statement···> ;
	END_FOR;
		The FOR...DO statement first initializes the data used as a repeat variable.
		An addition or subtraction is made to the initialized repeat variable according to the incremental expression. One or more statements from DO to END_FOR are repeatedly executed until the final value is exceeded.
		The repeat variable at the end of the FOR...DO syntax is the value at end of the execution.
			FOR intV1:=0
			  TO 30
			  BY 1 DO
			  intV3:=intV1+1;
			END_FOR;

■WHILE	WHILE <Boolean expression> DO
	<Statement···> ;
	END_WHILE;
		The WHILE...DO statement executes one or more statements while the value of Boolean expression (conditional expression) is TRUE.
		The Boolean expression is evaluated before the execution of the statement. If the value of Boolean expression is FALSE, the statement in the WHILE...DO statement is not executed. Since a return result of the Boolean expression in the WHILE statement requires only TRUE or FALSE, any Boolean expression that can be specified in the IF conditional statement can be used.
			WHILE intV1=30 DO
			  intV1:=intV1+1;
			END_WHILE;

■REPEAT	REPEAT
	<Statement···> ;
	UNTIL <Boolean expression>
	END_REPEAT;
		The REPEAT...UNTIL statement executes one or more statements while the value of Boolean expression (conditional expression) is FALSE.
		The Boolean expression is evaluated after the execution of the statement. If the value of Boolean expression is TRUE, the statement in the REPEAT...UNTIL statement are not executed. Since a return result of the Boolean expression in the REPEAT statement requires only TRUE or FALSE, any Boolean expression that can be specified in the IF conditional statement can be used.
			REPEAT
			  intV1:=intV1+1;
			  UNTIL intV1=30
			END_REPEAT;

■EXIT	EXIT;	The EXIT statement is used only in an iteration statement to end 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 processing continues from the line after the one where the iteration statement is ended.
			FOR intV1:=0
			  TO 10
			  BY 1 DO
			  IF intV1>10 THEN
			    EXIT;
			  END_IF;
			END_FOR;

Precautions

■When an assignment statement is used

- The maximum number of character strings that can be assigned is 255. If 256 or more character strings are assigned, a conversion error occurs.
- Contacts and coils of the timer type or counter type cannot be used for the left side of an assignment statement.
- The instance of a function block cannot be used for the left side of an assignment statement. Use input variables, input/output variables, and external variables of the instance for the left side of an assignment statement.

■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 right-hand 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 an assigned arithmetic operation is used

When an arithmetic operation result is assigned to a variable of the larger data type, convert the variable of the arithmetic operation to the data type of the left side in advance and execute the operation.

Ex.

When an arithmetic operation result of 16-bit data (INT type) is assigned to 32-bit data (DINT type)

```
varDint1 := varInt1 * 10;           // VarInt1 is a INT type variable, and varDint1 is a DINT type variable.
```

The arithmetic operation result is the same data type as that of the input operand. Thus, in the case of the above program, when the operation result of varInt1 * 10 exceeds the range of the INT type (-32768 to +32767), an overflow or underflow result is assigned to varDint1.

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

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

■Using the operator "-" for sign inversion in an arithmetic operation

When the operator "-" is used to invert the sign of the minimum value of a data type, the minimum value evaluates to the same value.

For example, -(-32768), where the operator "-" is used with the minimum value of INT type, evaluates to -32768. Thus, an unintended result may be produced if the operator "-" is used to invert the sign of a variable whose data type will be automatically converted.

Ex.

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

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

In the example above, the value of (-varInt1) evaluates to -32768 and -32768 is assigned to varDint2.

When using the operator "-" to invert the sign of a variable in an arithmetic operation, perform automatic conversion of the data type of the variable before the arithmetic operation. Alternatively, avoid using the operator "-" for sign inversion in the program.

Ex.

Performing automatic conversion of the data type before an arithmetic operation

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

Ex.

Avoiding the use of the operator "-" for sign inversion

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

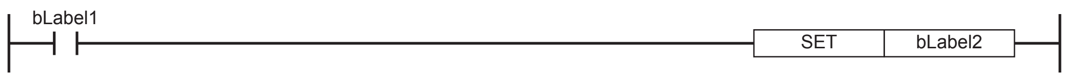
■When a bit type label is used

Once the Boolean expression (conditional expression) is satisfied in a conditional statement or an iteration statement, the bit type label that is turned ON in <Statement> is always set to ON.

Ex.

Program whose bit type label is always set to on

ST program	Ladder program equivalent to ST program

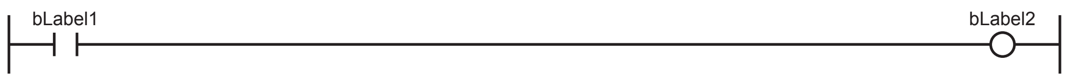
IF bLabel1 THEN
    bLabel2 := TRUE;
END_IF;
	

To avoid the bit device to be always set to ON, add a program to turn OFF the bit type label as shown below.

Ex.

Program to avoid the bit type label to be always set to ON

ST program[*1]	Ladder program equivalent to ST program

IF bLabel1 THEN
    bLabel2 := TRUE;
ELSE
    bLabel2 := FALSE;
END_IF;
	

*1 The above program can also be described as follows.
bLabel2 := bLabel1;
or
OUT(bLabel1,bLabel2);
However, when the OUT instruction is used in <Statement>, the program status becomes the same as the program whose bit type label is always set to on.

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

Boolean expression (conditional expression) in a conditional statement differs for the execution conditions of the timer function block or counter function block.

Ex.

When a timer function block is used

■Program before change

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

■Program after change

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

When a counter function block is used

■Program before change

```
IF bLabel1 THEN
    COUNTER_FB_M_1(Coil:=bLabel2,Preset:=wLabel3,ValueIn:=wLabel4,ValueOut=>wLabel5,Status=>bLabel6);
END_IF;
(* When bLabel1 = on and bLabel2 = on/off, the value is incremented by 1. *)
(* When bLabel1 = off and bLabel2 = on/off, the value is not counted.     *)
(* The counting operation does not depend on the on/off status of bLabel1. *)
```

■Program after change

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

An error occurs when the program before change is used since the statement related to the timer or counter is not executed when the selection statement is not satisfied.

When the timer or counter is operated according to the AND condition of bLabel1 and bLabel2, do not use any control statement, just use a function block only.

Using the program after change operates the timer and counter.

■When the FOR...DO statement is used

- Structure members and array elements cannot be used as repeat variables.
- Match the type used for a repeat variable with the types of <Last value expression> and <Incremental expression>.
- <Incremental expression> can be omitted. When omitted, <Incremental expression> is treated as 1 and executed.
- When 0 is assigned to <Incremental expression>, the statements after the FOR syntax may not be executed or the processing goes into an infinite loop.
- In the FOR...DO syntax, the counting process of repeat variables is executed after the execution of <Statement···> in the FOR syntax. If the count is greater than the maximum value or smaller than the minimum value of the data type of the repeat variable, the processing goes into an infinite loop.

■When a rising execution instruction or a falling execution instruction is used

Shown here is the operation when a rising execution instruction or an fall execution instruction is used in an IF statement or a CASE statement.

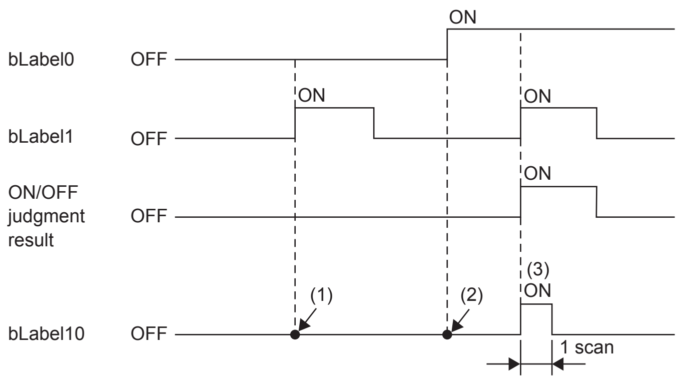
Condition			Result of operation		
Conditional expression of an IF statement or a CASE statement	**Condition to execute an instruction (EN)**	**Result of the ON/OFF judgment of the instruction at the time of the previous scan**	**Result of the ON/OFF judgment of the instruction**	**Rising execution instruction**	**Falling execution instruction**
Agreement of TRUE or CASE	TRUE	ON	ON	Not executed	Not executed
		OFF	ON	Executed	Not executed
	FALSE	ON	OFF	Not executed	Executed
		OFF	OFF	Not executed	Not executed
Disagreement of FALSE or CASE	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 This is a fall (ON to OFF), but the instruction is not executed because the condition in the IF statement or the CASE statement is not satisfied.

Ex.

When the PLS instruction (rising execution instruction) is used in an IF statement

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



(1) If bLabel0 = OFF (the condition expression in the IF statement is FALSE), the ON/OFF judgment result is OFF. The PLS instruction is not executed. (bLabel10 = OFF does not change.)
(2) If bLabel0 = ON (the condition expression in the IF statement is TRUE) and bLabel1 = OFF (the condition for executing the instruction is OFF), the ON/OFF judgment result is OFF. The PLS instruction is not executed. (bLabel10 = OFF does not change.)
(3) If bLabel0 = ON (the condition expression in the IF statement is TRUE) and bLabel1 = ON (the condition for executing the instruction is ON), the ON/OFF judgment result is OFF to ON (the condition for a rise is satisfied). The PLS instruction is executed. (bLabel10 turns ON for once scan only.)

■When a master control instruction is used

Shown here is the operation when the master control is OFF.

- The statement in a selection statement (an IF statement or a CASE statement) or in a iteration statement (a FOR statement, a WHILE statement, or a REPEAT statement) is not processed.
- Outside of a selection statement or a iteration statement, assignment statement is not processed and statement other than assignment statement is not executed.

Ex.

A statement in a selection statement (IF statement)

```
MC(M0,N1,M1);   //Master control OFF
IF M2 THEN
  M3:=M4;       //No processing is executed when the master control is OFF. So, M3 maintains the value at the time of a previous scan.
END_IF;
M20:=MCR(M0,N1);
```

Ex.

A statement out of a selection statement or a iteration statement (in the case of a bit assignment statement)

```
MC(M0,N1,M1);   //Master control OFF
M3:=M4;         //No processing is executed when the master control is OFF. So, M3 maintains the value at the time of a previous scan.
M20:=MCR(M0,N1);
```

Ex.

A statement out of a selection statement or a iteration statement (in the case of an OUT instruction)

```
MC(M0,N1,M1);   //Master control OFF
OUT(M2,M3);     //No execution is made when the master control is OFF.
M20:=MCR(M0,N1);
```

Constant

Methods for expressing constants

The following table shows the expression methods for setting a constant in a ST program.

Data type		Expressing method	Example
String(32)	STRING	Enclose character string (ASCII, Shift JIS) with single quotation (').	Stest := 'ABC';
String [Unicode](32)	WSTRING	Enclose a Unicode character string in double quotation marks (").	Stest := "ABC";

For the expression methods other than the one described the above, refer to the following.

☞ Page 39 Constant

Label and device

Specification method

Labels and devices can be directly described in the ST program. 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 available labels, refer to the following.

☞ Page 31 LABELS

For available devices, refer to the following.

📖MELSEC iQ-F FX5 User's Manual (Application)

■Device expression with type specification

A word device can be used in ST language as an arbitrary data type by adding a device type specifier to its name.

Device type specifier	Data type	Example	Description
N/A	Generic data type ANY16. When only devices are used in arithmetic operations, the data type is Word [signed]. However, when the data is specified as a device without the type specification in the argument part of FUN/FB, the data type is the one of the argument definition.	D0	When no type specifier is added to D0
:U	Word [Unsigned]/Bit String [16-bit]	D0:U	The value when D0 is Word [unsigned]/Bit string [16-bit]
:D	Double Word [Signed]	D0:D	The value when D0 and D1 are Double word [signed]
:UD	Double Word [Unsigned]/Bit String [32-bit]	D0:UD	The value when D0 and D1 are Double word [unsigned]/Bit string [32-bit]
:E	FLOAT (Single Precision)	D0:E	The value when D0 and D1 are single-precision real numbers

The following shows the devices to which device type specifiers can be added.

- Data register (D)
- Link register (W)
- Module access device (U□\G□)
- File register (R)

■Device specification method

The following methods can be used for specifying a device.

- Indexing
- Bit specification
- Nibble specification
- Indirect specification

For details, refer to the following.

📖MELSEC iQ-F FX5 User's Manual (Application)

📖MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)

Precautions

- The pointer type can be used for ST programs.
- When a value is assigned using nibble specification, use the same data type for the left side and right side of an operation.

Ex.

```
D0 := K5X0;
```

In the above case, since K5X0 is the double word type and D0 is the word type, an error occurs in the program.

- When a value is assigned using nibble specification and the data size of the right side is larger than that of the left side, data is transmitted within the range of the target points of the left side.

Ex.

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

In the above case, since the target points of K5X0 is 20, 1101_1111_0111_0011_0001 (20 bits) are assigned to K5X0.

- When the current value (such as TNn) of a counter (C), timer (T), or retentive timer (ST) is used with a type other than Word [unsigned]/Bit string [16-bit], or when the current value (such as LCNn) of a long counter (LC) is used with a type other than Double word [unsigned]/Bit string [32-bit], use the type conversion function.

Ex.

```
varInt := WORD_TO_INT(TN0); (*Use the type conversion function*)
```

Comment

The following table shows the comment formats that can be used in a ST program.

Comment format	Comment symbol	Description	Example
Single line comment	//	The character strings between the start symbol "//" and the end of the line are used as a comment.	// Comment
Multiple-line comment	(* *)	The character strings between the start symbol "(*" and the end symbol "*)" are used as a comment. Newlines can be inserted in the comment.	■Without newline (* Comment *) ■With newline (* Comment in the first line Comment in the second line *)
	/* */	The character strings between the start symbol "/*" and the end symbol "*/" are used as a comment. Newlines can be inserted in the comment.	■Without newline /* Comment */ ■With newline /* Comment in the first line Comment in the second line */

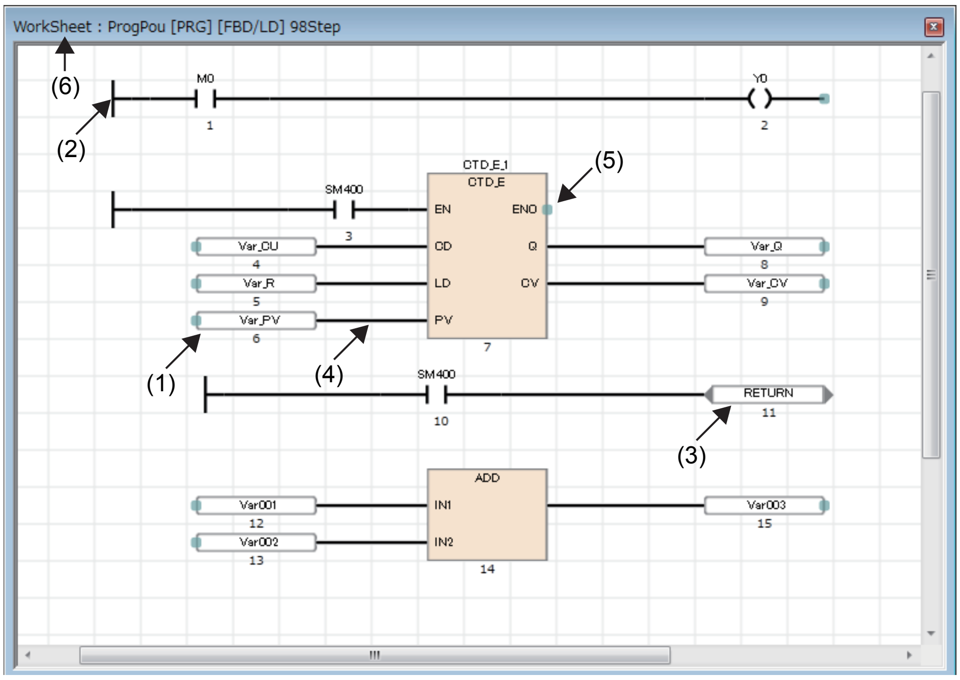
When the multiple-line comment format is used, do not use end symbols inside comments.

7 FBD/LD LANGUAGE

This is a language that creates a program by wiring blocks for specific processing, variables, and constants along with the flows of data and signals.

7.1 Configuration

With the FBD/LD language, the following program can be created.



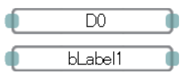
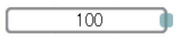
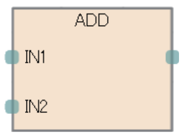
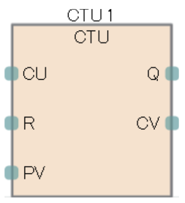
(1) FBD unit
(2) LD unit
(3) Common unit
(4) Connecting wire
(5) Connecting point
(6) Worksheet

In a program of the FBD/LD language, data flows from the output point of a function block (FB), a function (FUN), a variable unit (label or device), and constant unit to the input point of another function block, variable unit, and so forth.

Program unit

FBD unit

Units constituting FBD/LD program are shown below.

Unit	Symbol	Description
Variable		A variable is used to store each value (data). The data type of a variable should be a certain type. Only the value (data) of the data type is stored. You can specify a label or a device to a variable.
Constant		The constant specified is output.
Function (FUN)		Executes a function. • How to create functions (GX Works3 Operating Manual) • Standard function (MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks))
Function Block (FB)		Executes a function block. • How to create function blocks (GX Works3 Operating Manual) • Standard function blocks (MELSEC iQ-F FX5 Programming Manual (Instructions, Standard Functions/Function Blocks)) • Module function blocks (MELSEC iQ-F FX5 CPU Module Function Block Reference)

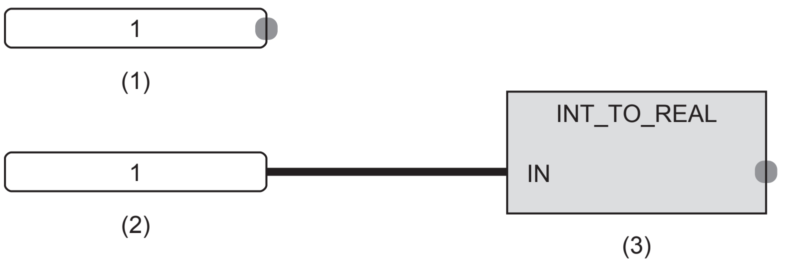
■The data type of a constant unit

In the case of a constant unit, the data type of the constant value is not determined at the time when the constant value is input. The data type is determined when the constant unit and an FBD unit are connected over a connecting wire. The data type of the constant value is the same data type as the FBD unit at the destination of the connecting wire.

Ex.

When 1 is input as a constant value

The data type can be a BOOL type, a WORD type, a DWORD type, an INT type, a DINT type, or a REAL type. So, the data type is not determined. When the constant unit and an FBD unit are connected over a connecting wire, the data type becomes the data type at the input point of the unit at the destination of the connection.



(1) The data type is not determined.
(2) INT type
(3) INT type

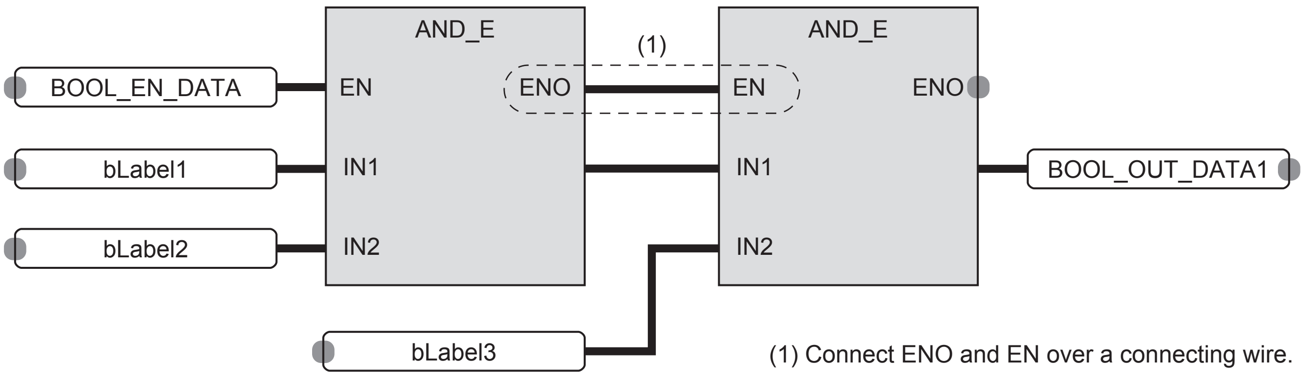
■Automatic conversion of data types

The data type of an element may be automatically converted when it is connected to another element of a different data type. To avoid the deletion of the data during the type conversion, only conversion from smaller type to larger type is performed.
Automatic conversion of data type in the FBD/LD language behaves in the same way as that in the ST language. For details, refer to the following.

☞ Page 50 Automatic conversion of data types

■The input/output point of a function

- It is necessary that all the input points of a function should be connected to other FBD units over connecting wires.
- The data types of the input variables and output variables of a function should be of certain types. It is necessary that the FBD units to be connected to the input point or output point should be of the same data types.
- Connect a variable element between an output variable (except for ENO) of a CPU module instruction or module dedicated instruction and an input variable of another function (or function block).
- In a program that connects a function with EN to another function over a connecting wire, the other function must be a function with EN and the program must connect ENO and EN over a connecting wire, in order to prevent the function from using an indefinite value.



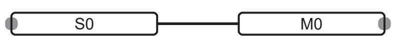
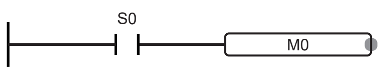
(1) Connect ENO and EN over a connecting wire.

■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.

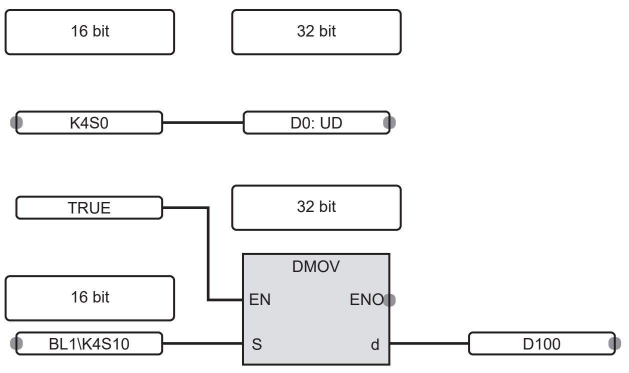
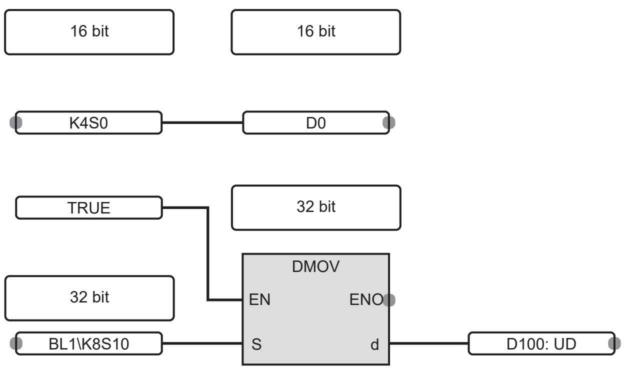
The following is the example of rewriting.

Before change	After change
	

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.

Before change	After change
	

LD unit

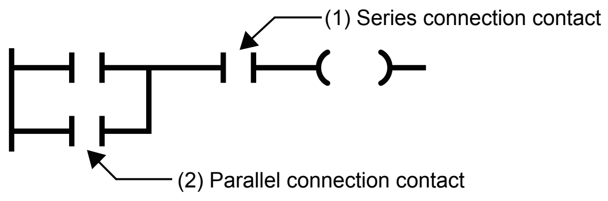
Units of ladder diagram that can be used in a program of the FBD/LD language are shown below.

Unit	Symbol	Description
Left bus	├●	This is an unit to represent a bus. This is the starting point to create a ladder circuit.
NO contact	●─┤ ├─●	Turns on when a specified device or label is ON.
NC contact	●─┤/├─●	Turns on when a specified device or label is OFF.
Rising edge	●─┤↑├─●	Turns on at the rising edge (OFF to ON) of a specified device or label.
Falling edge	●─┤↓├─●	Turns on at the falling edge (ON to OFF) of a specified device or label.
Negated rising edge	●─┤/↑├─●	Turns on when a specified device or label is OFF or ON, or at the falling edge (ON to OFF) of a specified device or label.
Negated falling edge	●─┤/↓├─●	Turns on when a specified device or label is OFF or ON, or at the rising edge (OFF to ON) of a specified device or label.
Coil	●─( )─●	Outputs an operation result to a specified device or a label.
Complementing coil	●─(/)─●	When the operation result turns OFF, the specified device or label turns ON.
Set	●─(S)─●	When the operation result turns ON, the specified device or label turns ON. The device or the label that turns ON remains ON even if the operation result turns OFF.
Reset	●─(R)─●	When the operation result turns ON, the specified device or label turns OFF. When the operation result is OFF, the status of the device or the label does not change.

■The AND operation and OR operation of a contact symbol

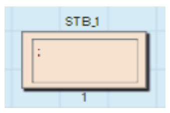
A contact symbol executes an AND operation or an OR operation depending on the status of the connection of a circuit chart. This is reflected in the operation result.

- In the case of a series connection (1), an AND operation is executed with the operation results so far. This will be the operation result.
- In the case of a parallel connection (2), an OR operation is executed with the operation results so far. This will be the operation result.


(1) Series connection contact
(2) Parallel connection contact

Common unit

This represents a common unit placed on the FBD/LD editor.

Unit	Symbol	Description
Jump[*1]		The execution processing is jumped over from a jump unit to a jump label. The portion that is jumped over is not executed. Whether a jump is made or not is controlled depending on the ON/OFF information to the jump unit. ON: The execution processing is jumped over up to a jump label. OFF: The execution processing is not jumped over but is executed.
Jump label[*1]		This is the destination of a jump from a jump instruction in the same program. The processing is executed from a program in the execution order after the jump label.
Connector		This is used as a substitute of a connecting wire. The processing moves on to the corresponding connector unit. You can use one input connector or multiple input connectors for one output connector.
Return[*1]		The processing after a return unit in the program is aborted. Use this when you want to prohibit the execution of the processing of a program, function, or a function block after the return unit. Whether the return processing is executed or not is controlled depending on the ON/OFF information to the return unit. ON: The return processing is executed. OFF: The return processing is not executed, but the ordinary execution processing is executed.
Comment		Use this to describe a comment.
Inline ST[*1]		An ST program is displayed in the FBD/LD editor. By double-clicking an inserted inline ST element, the ST editor is displayed for editing or monitoring an ST program. For details, refer to the following. ☞ Page 69 Inline ST

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

■Precautions for a jump unit

- If the timer of a coil that is ON is jumped over by using a jump unit, a normal measurement cannot be conducted.
- You can add a jump label on the top side (the execution is earlier) of a jump unit. In this case, create the program by including a method to break the loop in order not to exceed the setting value of the watchdog timer.
- You can specify only a local label of a pointer type for a jump element and jump label. The structure members cannot be used.
- The pointer branch instruction (CJ) cannot be used. For jumping, use jump elements.
- Jumps to or from outside the program block cannot be executed. The following is a list of jump operations that cannot be executed.

- Jumping to outside the program block[*1]
- Jumping from outside the program block[*1]
- Calling subroutine programs
- Called as subroutine programs

*1 Includes branches caused by the BREAK instruction.

7

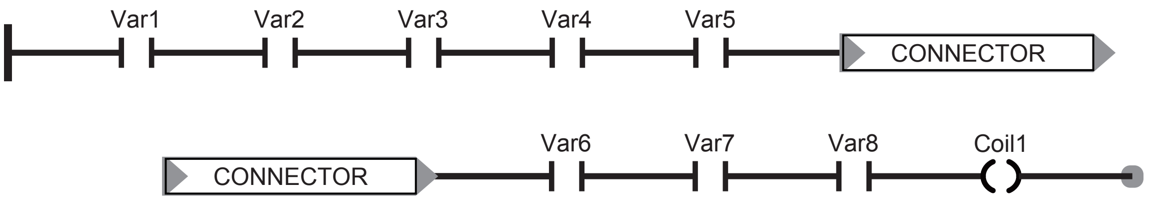
■The operation of a return unit

A return unit operates differently depending on whether a program, function, and/or function block used there.

Program unit to use	
Program	The execution of the program unit is terminated.
Function	The function is terminated, and the step goes back to the one next to the instruction that has called the function.
Function block	The function block is terminated, and the step goes back to the one next to the instruction that has called the function block.

■Connector unit

Use a connector element to place the program within the display area or print area of the FBD/LD editor.



Connecting wire

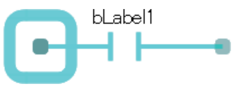
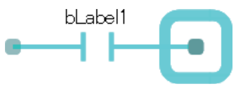
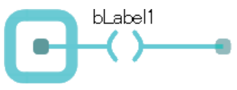
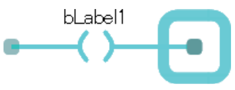
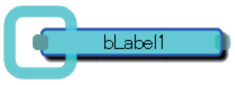
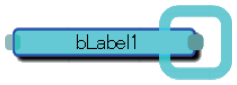
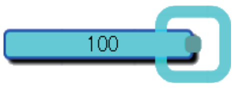
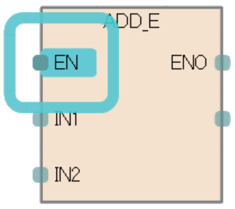
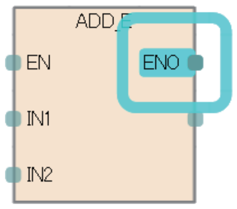
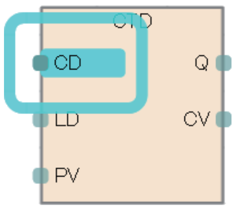
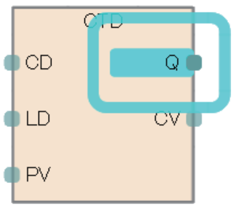
This is the wire to connect the connecting points between FBD unit, LD unit, and common unit.

After units are connected, the data is transferred from the left end to the right end. The data types of the connected units need to be the same.

Connecting point

This is a terminal point to use a connecting wire to connect FBD unit, LD unit, and common unit.

The point on the left side of each unit is the input side, while the point on the right side represents the output side.

Unit	Input connecting point	Output connecting point	Unit	Input connecting point	Output connecting point
Contact			Coil		
Variable			Constant	—	
Function		 The return value is not shown on a function.	Function Block		

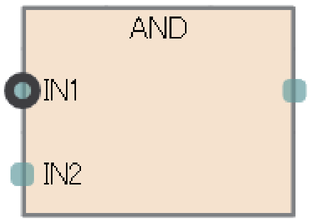
The connecting point is hidden after connecting a wire.

■Inverting input and output points

You can invert an input to an unit or an output from an unit by using a connecting point.

The connecting point having been inverted is circled with a black circle. The data to be input or output is inverted (FALSE to TRUE or TRUE to FALSE).

You can invert the following data types: BOOL, WORD, DWORD, ANY_BIT, and ANY_BOOL.



Worksheet

A worksheet is a work area for inserting program units and for connecting them with wires.

Constant

Methods for expressing constants

The following table shows the expression methods for setting a constant in FBD/LD language.

Data type		Expressing method	Example
String(32)	STRING	Enclose character string (ASCII, Shift JIS) with single quotation (').	'ABC'
String [Unicode](32)	WSTRING	Enclose a Unicode character string in double quotation marks (").	"ABC"

For the expression methods other than the one described the above, refer to the following.

☞ Page 39 Constant

Labels and devices

Specification method

You can directly describe and use labels and devices in an FBD/LD program. You can use labels and devices for inputs and output points of units, for arguments of standard functions/function blocks, return values, and so forth.

For available labels, refer to the following.

☞ Page 31 LABELS

For available devices, refer to the following.

📖MELSEC iQ-F FX5 User's Manual (Application)

■Device expression with type specification

A word device can be used as any data by adding a device type specifier to its name. If you do not specify a data type, the word device operates as a word [signed] (INT).

For the device type specifiers and the devices you can use, refer to the following.

☞ Page 59 Device expression with type specification

If you do not specify a data type for a word device, the data type is determined by the type of device.

Word device	Data type
The current value of a timer device (TN), the current value of a retentive timer device (STN), the current value of a counter device (CN)	WORD
The current value of a long counter device (LCN)	DWORD
Other than the above	INT

Caution

■When using label

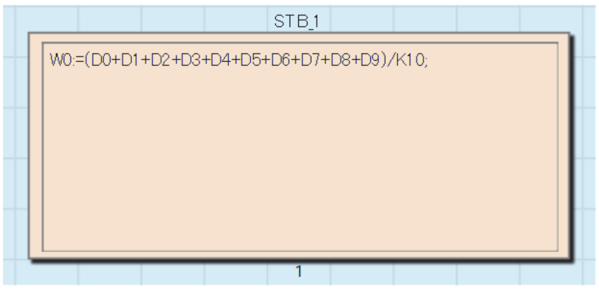
- Labels whose name ends with "_" cannot be used as an array index. To use such a device or label as an array index, assign it to another device or label and specify that device or label as an index.
- Members of labels (structures or function blocks) whose name ends with "_" cannot be specified.
- Indexes cannot be specified to labels (arrays) whose name ends with "_".

7.2 Inline ST

The inline ST is the function used to create an inline ST unit that displays an ST program in the FBD/LD editor, and edit and monitor it.

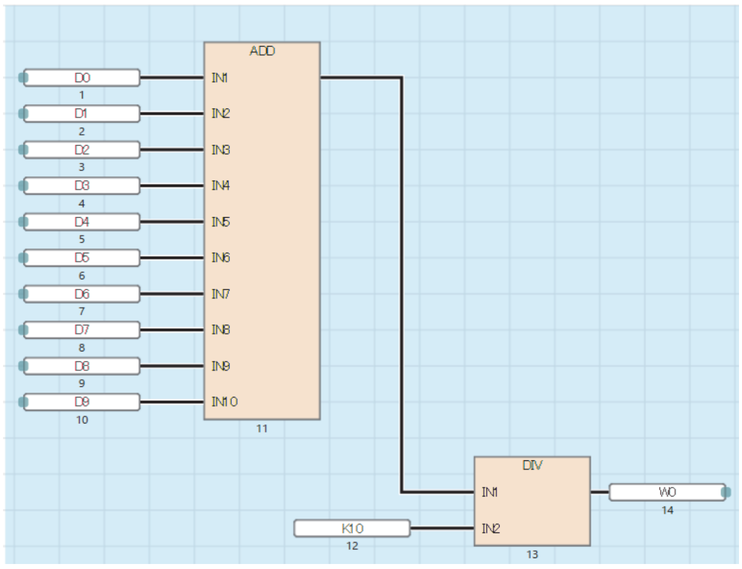
This function enables to create numerical operations and character string processing easily in FBD/LD programs.

- Program that uses the inline ST



```
W0:=(D0+D1+D2+D3+D4+D5+D6+D7+D8+D9)/K10;
```

- Program that does not use the inline ST



Restriction

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

Specifications

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

☞ Page 47 ST LANGUAGE

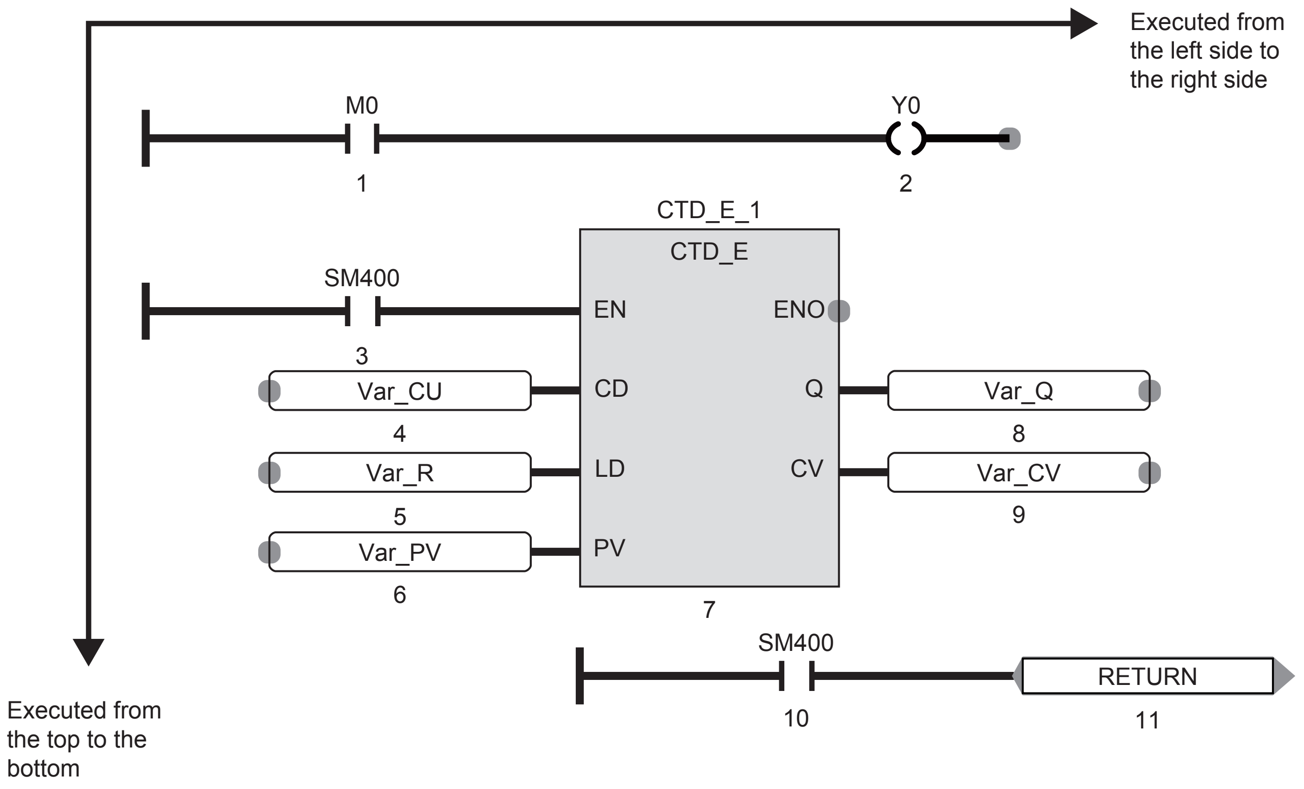
Precautions

- Up to 64 inline ST elements can be inserted into a single POU of an FBD/LD program.
- When the RETURN syntax is used in an inline ST, the processing inside the inline ST unit ends, and the processing inside the program block does not end.
- Since inline ST units have no connection points, an inline ST element inserted into an FBD/LD program is executed every scan.

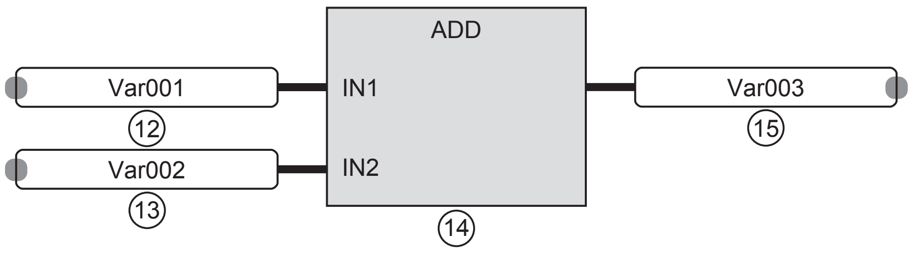
7.3 Program Execution Order

The order of executions of program units

The order of executions of the units in the FBD/LD editor is determined depending on the positional relation of the units and on the status of connecting wires.



The number of the order of the execution is shown on each unit placed on the FBD/LD editor.



8 SFC PROGRAM

SFC is a program description format in which a sequence of control operations is split into a series of steps and the execution sequence and execution conditions of each program can be clearly expressed.

Point

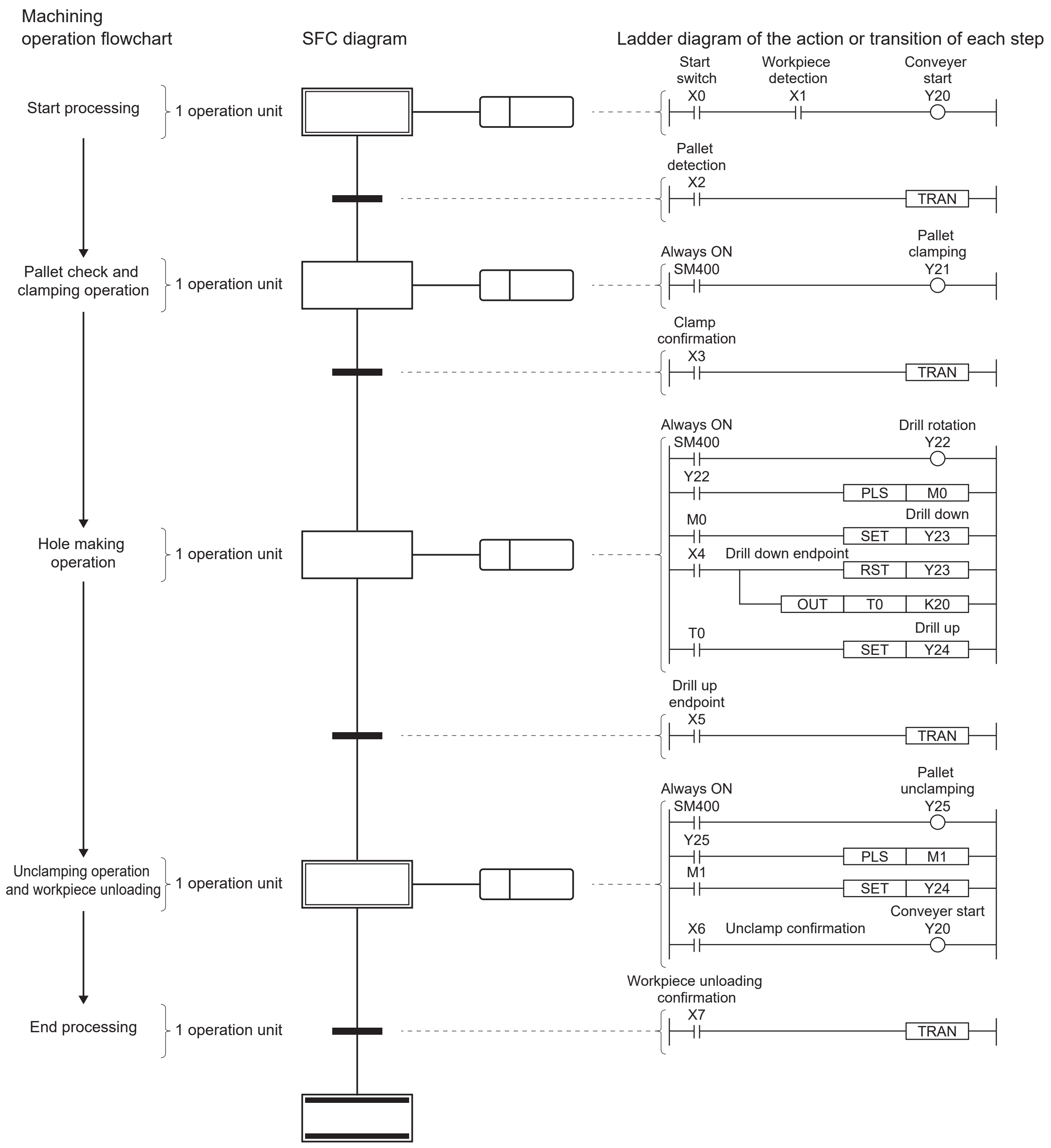
- It is compatible with the SFC program of FX3, and the configuration of FX3 can be replaced with that of FX5. (☞ Page 100 FX3 compatible transition operation mode setting)
- This chapter describes the operations and specifications of SFC programs. For details on the information not described in this chapter, refer to the following.

📖GX Works3 Operating Manual
📖MELSEC iQ-F FX5 User's Manual (Application)
📖Transition from MELSEC FX3G, FX3U, FX3UC Series to MELSEC iQ-F Series Handbook

Restriction

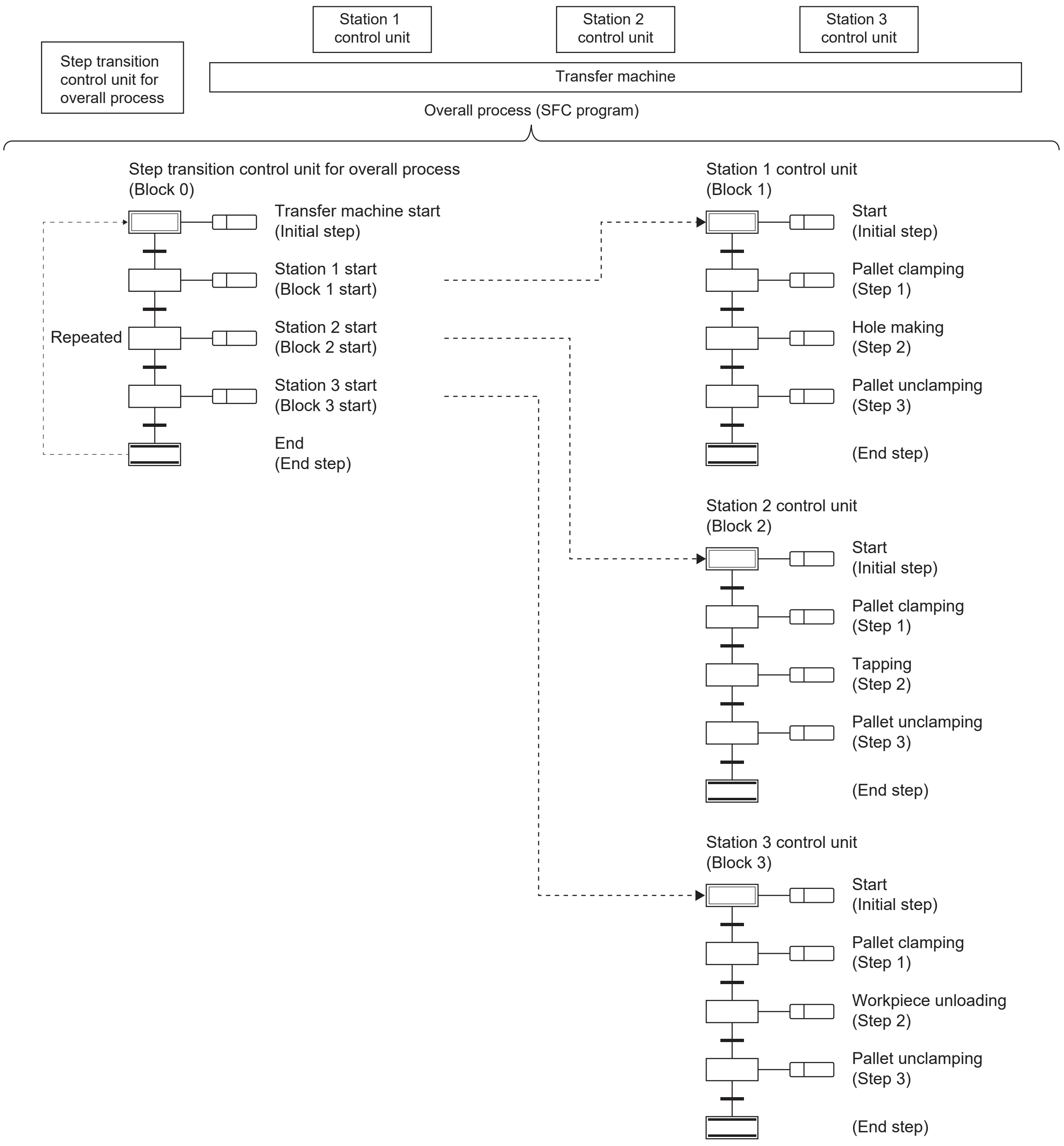
- Applicable only to FX5U/FX5UC CPU module.
- Check the versions of the CPU module and the engineering tool before using the SFC program. For the versions of the CPU module and engineering tool, refer to ☞ Page 119 Added and Changed Functions.

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 assign the actual controls of the entire facility, mechanical devices of each station, and all machines to the steps in the blocks of the SFC program.


Station 1 control unit
Station 2 control unit
Station 3 control unit
Step transition control unit for overall process
Transfer machine
Overall process (SFC program)
Step transition control unit for overall process (Block 0)
Transfer machine start (Initial step)
Station 1 start (Block 1 start)
Repeated
Station 2 start (Block 2 start)
Station 3 start (Block 3 start)
End (End step)
Station 1 control unit (Block 1)
Start (Initial step)
Pallet clamping (Step 1)
Hole making (Step 2)
Pallet unclamping (Step 3)
(End step)
Station 2 control unit (Block 2)
Start (Initial step)
Pallet clamping (Step 1)
Tapping (Step 2)
Pallet unclamping (Step 3)
(End step)
Station 3 control unit (Block 3)
Start (Initial step)
Pallet clamping (Step 1)
Workpiece unloading (Step 2)
Pallet unclamping (Step 3)
(End step)

8.1 Specifications

This section lists the performance specifications related to SFC Programs.

Item		Specifications
Number of device points (SFC)	Step relay (S)	4096 points
	SFC block device (BL)	32 points
	SFC transition device (TR)	0 points
Number of executable SFC programs		1
Number of blocks		32 blocks
Number of SFC steps		Up to 4096 steps in all blocks, up to 512 steps in 1 block
Step No.		0 to 511 per block
Number of branches		32 branches maximum
Number of simultaneously active steps		Up to 128 steps in all blocks and in 1 block
Number of initial steps		1 step maximum per block
Number of actions		4 actions maximum per step
Number of sequential steps	Action	No limit
	Transition	Only one per ladder block

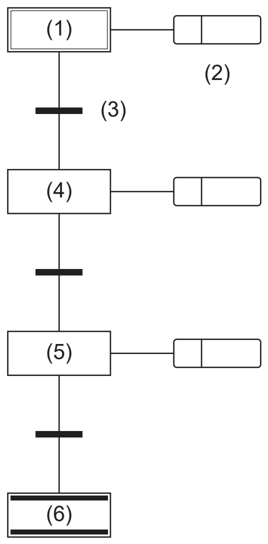
Point

For the processing time of the SFC program, refer to the following.
📖MELSEC iQ-F FX5 User's Manual (Application)

8.2 Structure

Basic operation of SFC

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) Initial step
(2) Action
(3) Transition
(4) Normal step
(5) Normal step
(6) 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).

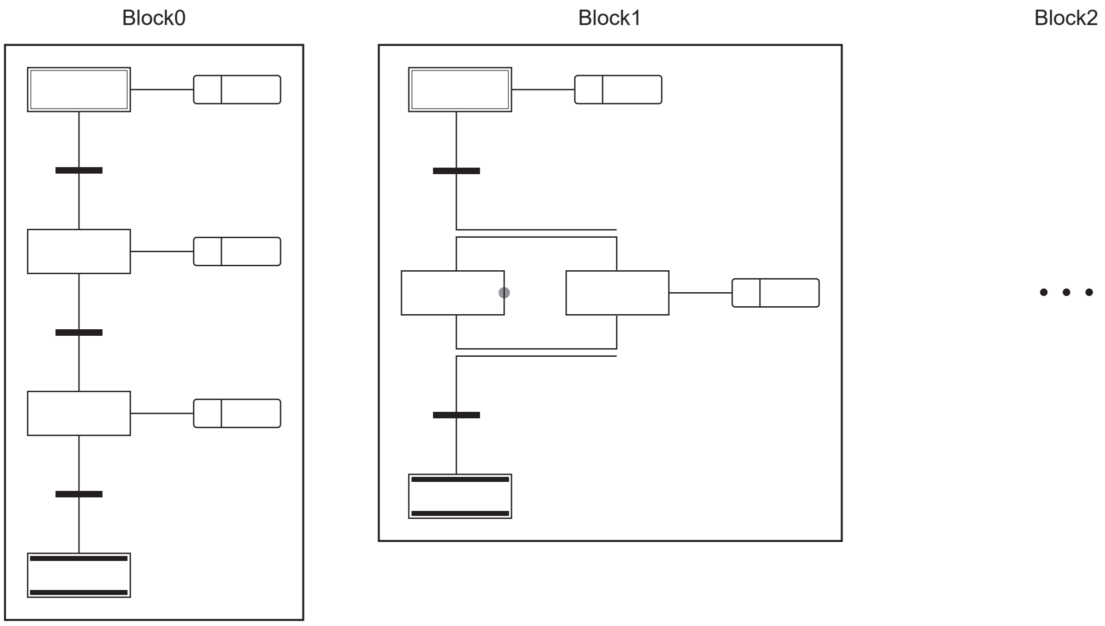
5. Every time the transition becomes TRUE, the program activates the next step and ends the block when it finally activates the end step (6).

Point

- Up to 4 actions can be created in one step. When multiple actions are created, they will be executed in order from the top. (☞ Page 85 Action)
- The operation of the initial step and normal step can be changed by adding the attribute. (☞ Page 78 Step types)

Block

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



For the maximum number of blocks that can be created in an SFC program, refer to the following.

☞ Page 74 Specifications

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 is in the active or inactive state.

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

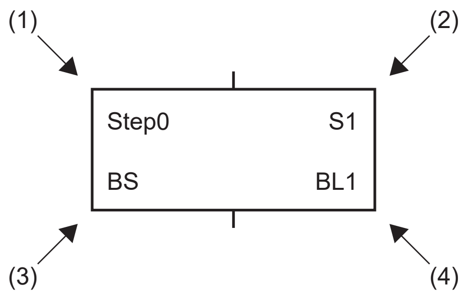
When the block state changes from inactive to active, the initial step becomes active to start sequential processing. (☞ Page 103 Block execution sequence)

Point

- Depending on the CPU parameter setting, only the block 0 can 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 will be automatically restarted and executed again from the initial step. (☞ Page 99 Start Conditions Setting)
- If a start request is issued to a step in an inactive block by using the SFC control 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.



(1) Step name
(2) Step No.
(3) Attribute
(4) Attribute target

For the maximum number of steps that can be created per block, refer to the following.

☞ Page 74 Specifications

Steps have the following characteristics.

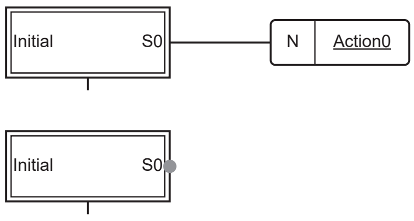
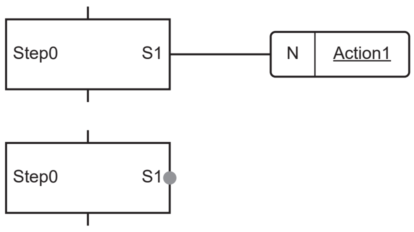
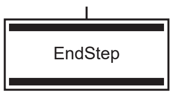
- When the step becomes active, the associated action is executed.
- A step No. is assigned to each step. The step No. is used to monitor the running step or forcibly start or stop the step by using the SFC control instruction. (☞ Page 84 Assigning the step relay (S) areas to steps)
- Each step name and No. are unique within each block. (Each cannot be a blank.)

Point

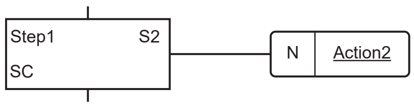
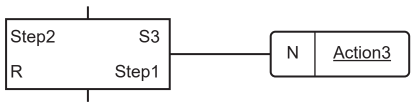
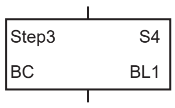
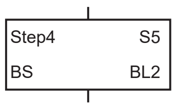
The step name, step No., attribute and attribute target can be changed in the Step Properties window. Select a step, and select [Edit] ⇨ [Properties] on the menu. Then, the Step Properties window will appear. (📖GX Works3 Operating Manual)

Step types

The following table lists the types of steps.

Item		Description
Initial step		A step that indicates the beginning of a block. While this type of step is active, the transition following the step is constantly checked, and when the transition becomes TRUE, the next step becomes active. The attributes of SC and R can be added. This step can also be used as a step without creation of an action.
Normal step		A basic step used to configure a block. While this type of step is active, the transition following the step is constantly checked, and when the transition becomes TRUE, the next step becomes active. The attributes of SC, R, BC, and BS can be added. This step can also be used as a step without creation of an action.
End step		A step that ends a block. An action cannot be created.

The following table lists the attributes of steps.

Attribute	Item		Description
SC	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.
R	Reset step [R]		Reset step [R] is 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 "Step Attribute" 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 "Step Attribute Target" in the Properties window.

For the setting procedure, refer to the following.

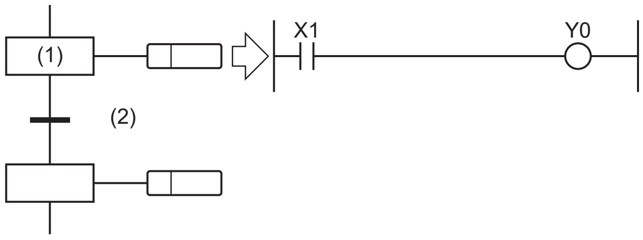
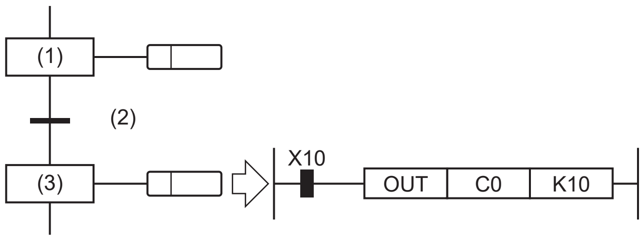
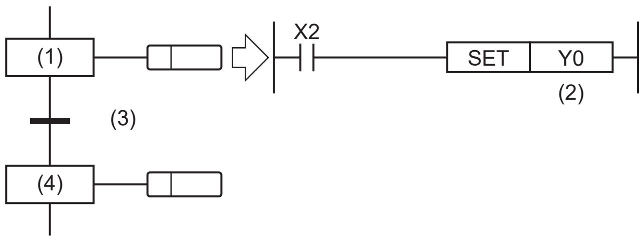
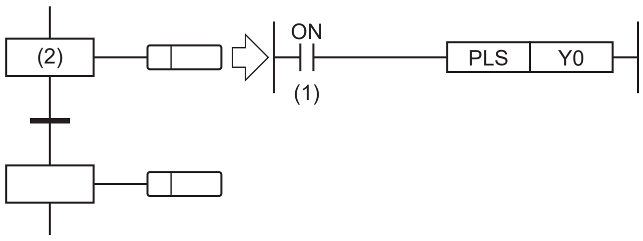
📖GX Works3 Operating Manual

Normal step (without attribute)

A basic step used to configure a block.

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

The output status of the action of a step at a transition to the next step 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.	 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.
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 using the counter, 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 using counter becomes inactive. To reset the counter, use the RST instruction in another step.	 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.
When the SET, basic, or application instruction is used	Even if the active state transitions to the next step and the step that has used the instruction becomes inactive, the ON state or the data stored in the device/label are held. To turn off the ON device/label or clear the data stored in the device/label, use the RST instruction in another step.	 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.
When the PLS instruction or instructions executed at the rising edge is used	Even when the contact of the execution condition is constantly on, the instruction is executed every time the step using the instruction changes from inactive to active.	 Even when the contact of the execution condition is on (1), the PLS instruction is executed every time the step (2) becomes active.

■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 each block. Only one initial step can be described in one block. (☞ Page 74 Specifications) 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.

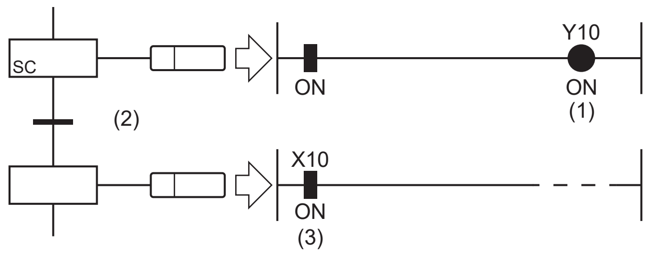
Operation of active step	Method
All initial steps become active.	When a start is made using the block start step
	When a start is made using the block START instruction of the SFC control instructions
	When block 0 is started using the auto-start setting of block 0
Only the specified step is activated.	When any of the initial steps is specified using the step control instruction of the SFC control instructions

■Operation of the initial steps with step attributes

The attribute of SC (coil HOLD step) or R (reset step) can be added to the initial step. When the attribute is added, the operations other than the operation that will be automatically activated at the start of the block are the same as those in 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 the OUT instruction is not turned off and 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 the SFC control instruction (Ending a block).
- A step is reset by using the SFC control instruction (Ending a step).
- A reset step [R] for resetting the coil HOLD step [SC] becomes active.
- SM321 (Start/stop SFC program) is turned off.
- The coil is reset by the program.
- S999 is specified at a reset step [R] within a block.

Reset step [R]

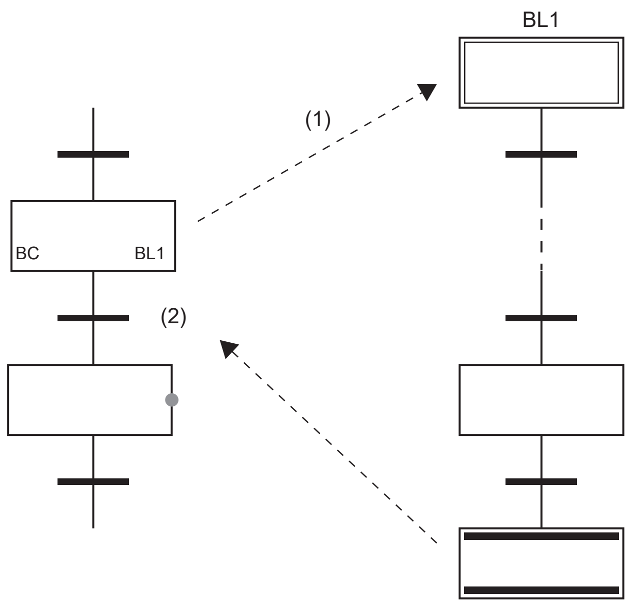
Reset step [R] is a step that deactivates the specified step.

- The reset step [R] deactivates the specified step in the current block before executing the output of every scan. Except for resetting the specified step, the reset step is the same as a normal step (without step attributes).
- The step No. of the coil HOLD step [SC] or S999 can be specified as specified step No. When the specified step No. is S999, the coil HOLD steps [SC] that hold operations in the current block are all deactivated.

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.



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 transition (2) 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 depends on "Act at block Multi-Activated." (☞ Page 101 Act at block Multi-Activated)

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

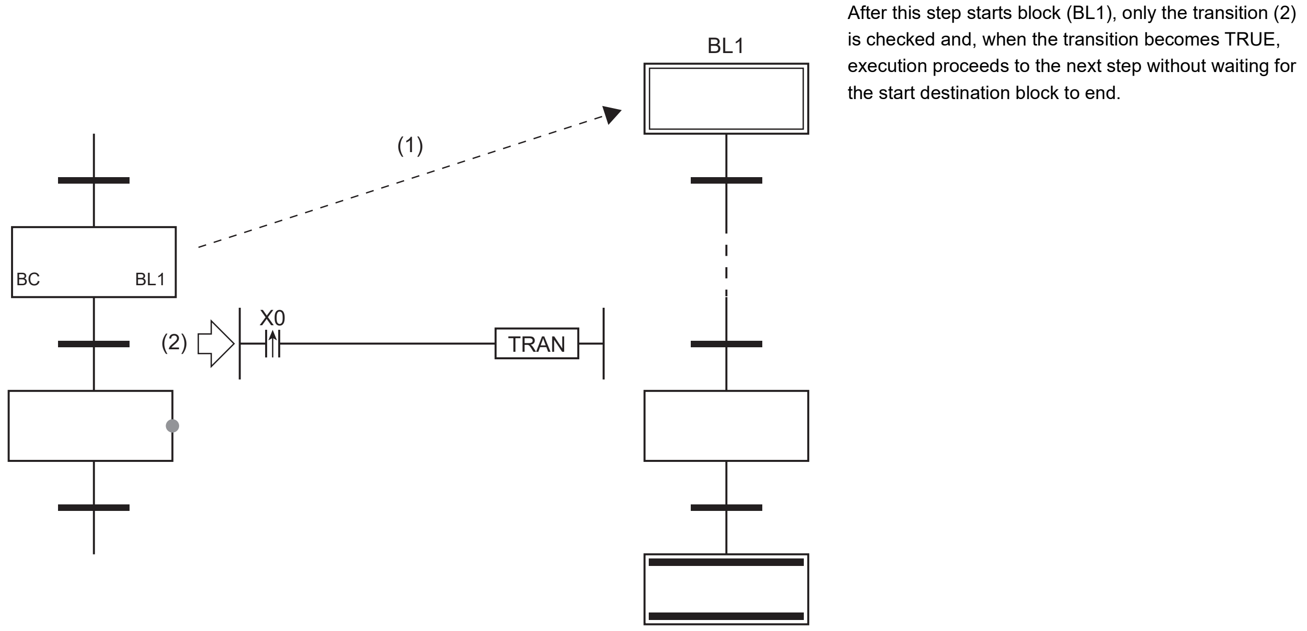
■Precautions

- 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.
BL1
(1)
BC
BL1
(2)
X0
TRAN

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 depends on "Act at block Multi-Activated." (☞ Page 101 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 if there are no active steps other than steps that hold operations in the block, all the HOLD steps 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 block continues running in the state when the end step is reached and does not end.

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

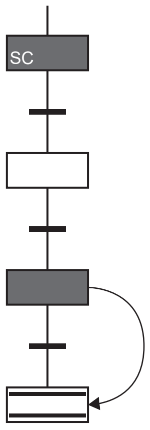
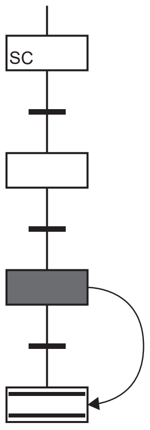
Status of SM327	Description
OFF (default)	All the outputs of the HOLD steps that hold operations are turned off.
ON	All the outputs of the HOLD steps that hold operations are held. The setting of SM327 is valid for only the coil HOLD step [SC] that holds operation. All the outputs of the coil HOLD step [SC] that does not hold operation and the transition does not become TRUE are turned off. Also, even 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	"Start Conditions Setting" is set to "Auto-start block 0" in the SFC Setting of parameters.	The initial step is automatically activated again and processing is executed repeatedly.
	"Start Conditions Setting" is set to "Do not auto-start block 0" in the SFC Setting of parameters.	The block is restarted when a start request is issued for the specified block in the following methods. • The block start step is activated by another block. • The SFC control instruction (Starting a block) is executed.
All blocks other than block 0		

Precautions

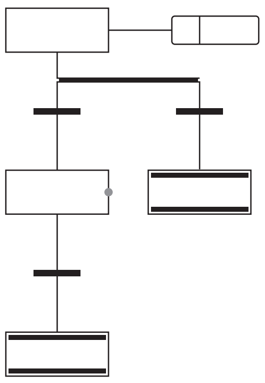
- An action cannot be created to the end step.
- The setting of SM327 (Output mode at execution of the END step) is valid only when the end step becomes active. When a forced termination is performed by the SFC control instruction (Ending a block) or the like, the coil outputs of all steps are turned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

When a normal active step remains or when a coil HOLD step [SC] whose transition has not become TRUE remains (the step does not hold an operation)	When an active step that holds an operation remains
 • When SM328 is off, the block is ended by clearing the step. • When SM328 is on, processing is continued without clearing the step.	 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.
- When "FX3 compatible transition operation mode setting" is set to "Enable", the CPU module is powered off and on, or SM328 is turned on at reset.

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 (even in the stop or hold state) and is off when the relevant step is inactive.

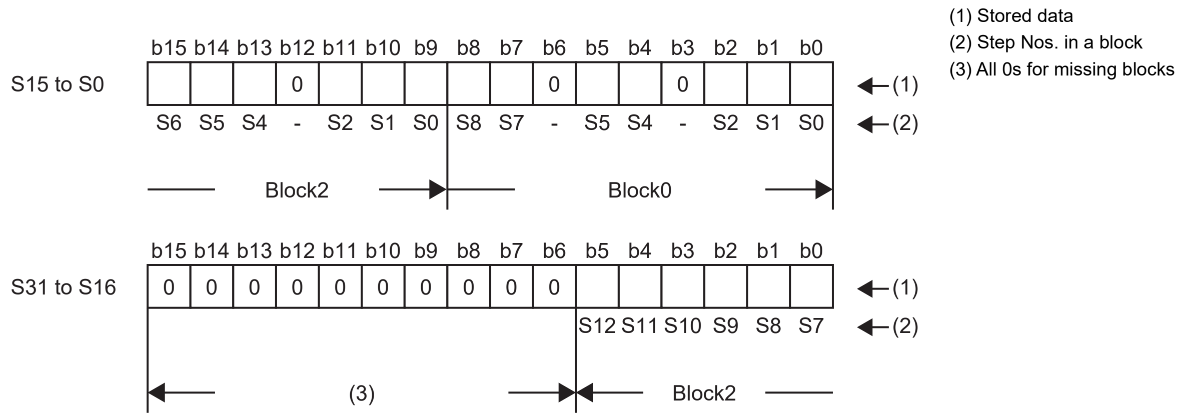
Step relays are assigned as follows.

- 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.
- Step relays are assigned to missing step Nos. in one block. The bits of the missing numbers are constantly 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 No. 3 and 6 are missing.
- Block1: Missing
- Block2: The largest step No. is 12, and step No. 3 is missing.
- Block3 and after: Missing



Point

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

- Assign step Nos. 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 initial step in a block.

For the step Nos. that can be used per block, refer to the following.

☞ Page 74 Specifications

Any step No. exceeding the upper limit cannot be assigned. Any step No. must be unique within a block. Same step Nos. can be used in different blocks.

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

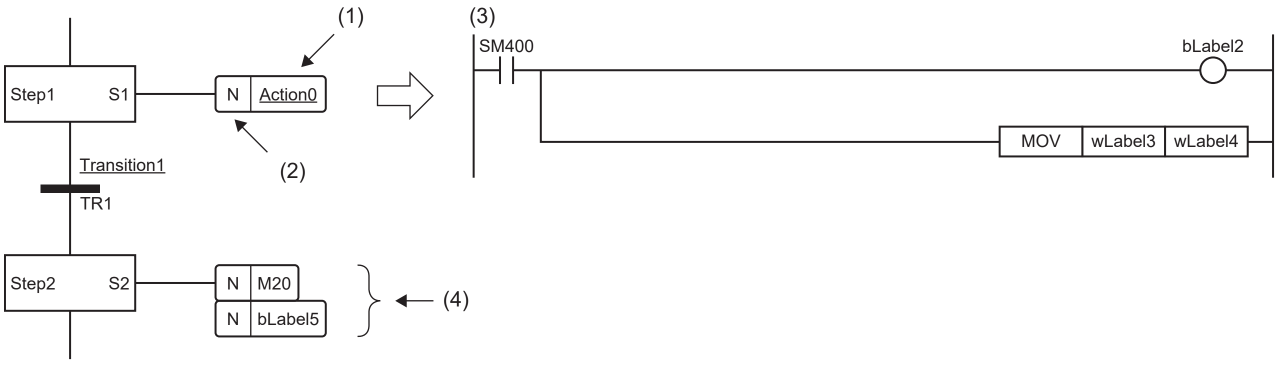
Ex.

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.

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 will be executed in order from the top.

The detailed expression of actions can be created in ladder, ST or FBD/LD language.

Point

For details on detailed expression or labels/devices, refer to the following.
📖GX Works3 Operating Manual

Instructions that cannot be used

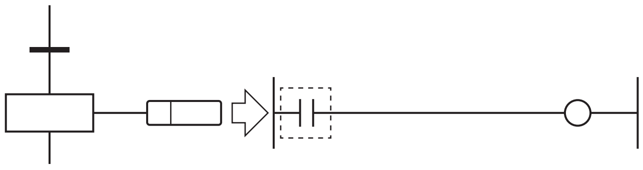
Some instructions cannot be used in actions. The following table lists the instructions that cannot be used.

Classification	Instruction symbol
Master control instruction	MC[*1]
	MCR[*1]
Termination instruction	FEND
	END
Program branch instruction	CJ[*1]
	GOEND
Program execution control instruction	IRET
Structure creation instruction	BREAK[*1]
	RET
	SRET
Creating a dummy transition condition	TRAN
Step ladder instruction	STL
	RETSTL
Initial state	IST

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

Point

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



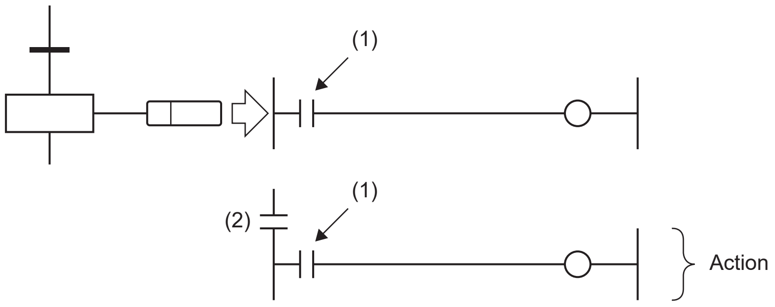
■Restrictions

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

Language		Description
Ladder diagram	Detailed expression	A pointer and an interrupt pointer cannot be input in the pointer input area. ■Functions/function blocks that cannot be used • 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 to Control EN" is set to "Yes" and "Use EN/ENO" is set to "No"
Structured text language		☞ Page 47 ST LANGUAGE
FBD/LD language		☞ Page 61 FBD/LD LANGUAGE

Precautions

• The step operation is almost the same as the following circuit. For the execution of action, refer to ☞ Page 104 Execution of action.



(1) Input condition of each instruction
(2) Contact indicating the step status (on when active, off when inactive)

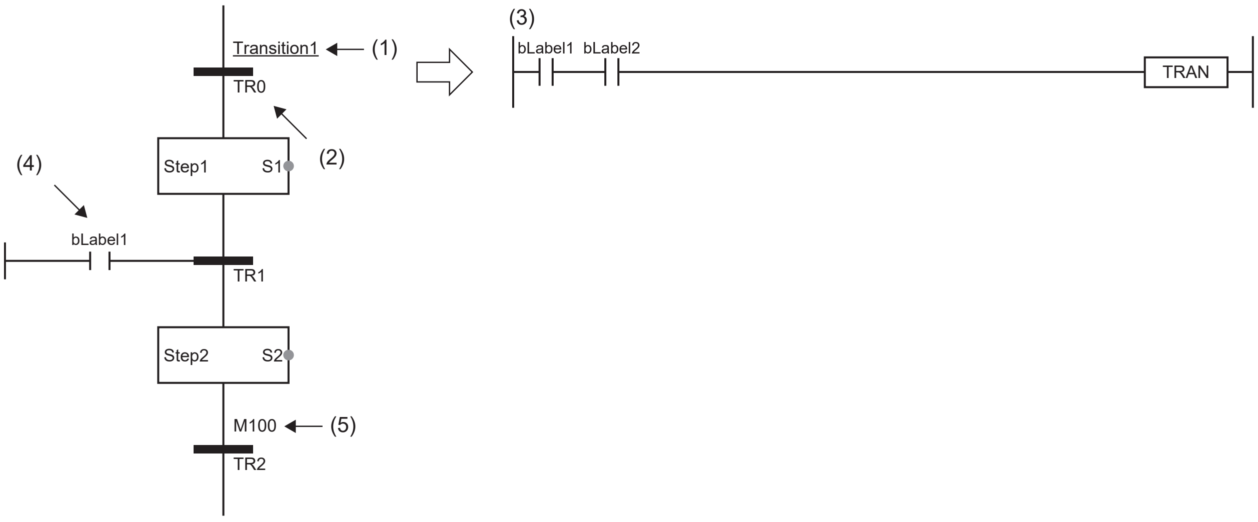
• If the CALL instruction is used to issue a subroutine call in an action of the step, the output of the call destination is not turned off even when the step becomes inactive after the transition becomes TRUE. To turn off the output of the call destination when the step becomes inactive after the transition becomes TRUE, use the XCALL instruction.
• 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 80 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.
• The operation of SFC control instruction depends on the execution condition before the no-execution status is entered.

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.

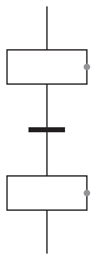
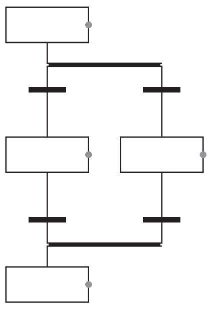
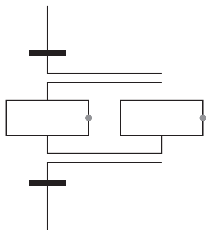
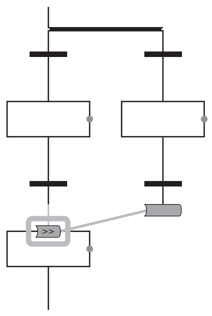


(1) Transition name
(2) Transition No.
(3) Detailed expression of transition (☞ Page 93 Detailed expression of transitions)
(4) Direct expression of transitions (☞ Page 94 Direct expression of transitions)
(5) Transition label/device (☞ Page 94 Transition label/device)

Detailed expression of a transition can be created in ladder diagrams, ST language, or FBD/LD language.

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.

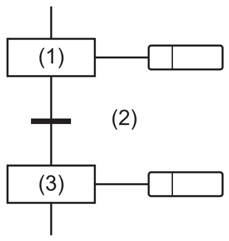
Point

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

☞ Page 111 Behavior when an active step is activated

Series sequence

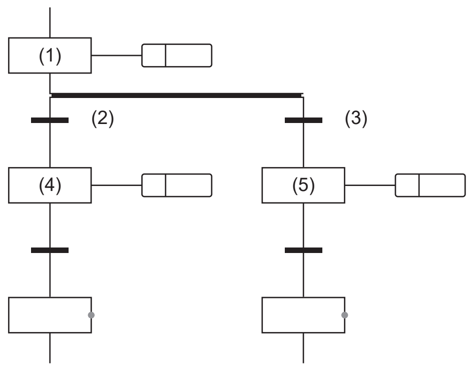
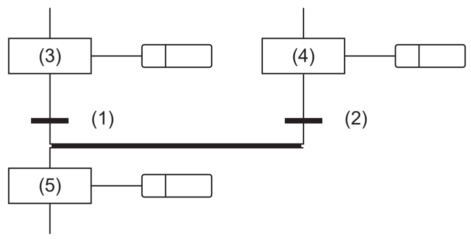
When the transition becomes TRUE, the active state transitions from the preceding step to the subsequent step.



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

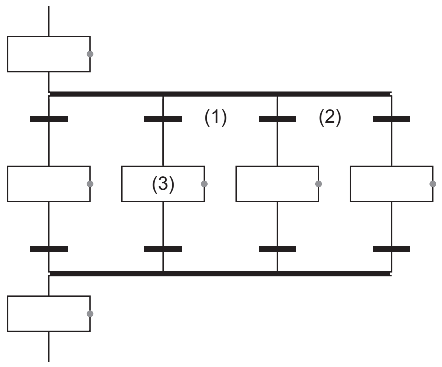
Selective sequence (divergence/convergence)

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		When the step (1) is active, the step whose transition becomes TRUE first is activated. (When the transition (3) becomes TRUE before (2) becomes TRUE, the step (5) is activated.) The step (1) becomes inactive. However, if it is a coil HOLD step [SC], the step holds the coil output or action according to its attribute. • 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		When the transition (1) or transition (2) on the activated branch becomes TRUE, the step (5) is activated. The activated step (3) or (4) becomes inactive. However, if it is a coil HOLD step [SC], the step holds the coil output or action according to its attribute.

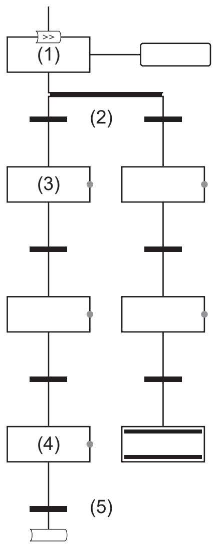
- The selective sequence allows branching to up to 32 transitions.
- If multiple transitions become TRUE simultaneously, the condition to the left will take precedence.

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



- An SFC diagram in which the number of branches is different from the number of merges in a selective sequence can also be created. However, in an SFC diagram, selective divergence and parallel convergence or parallel divergence and selective convergence cannot be combined.

- In a selective transition, a convergence can be omitted by a jump transition or end transition.



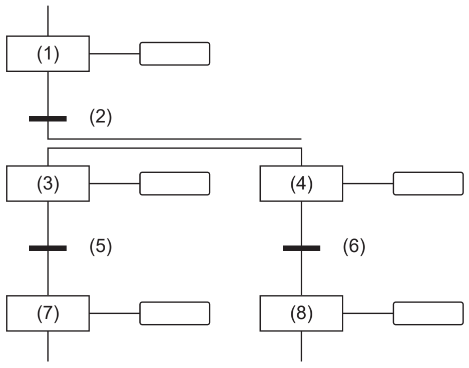
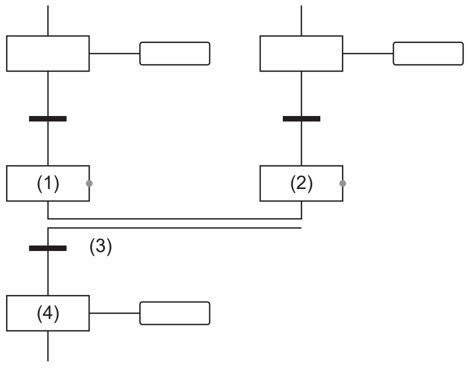
When transition (2) becomes TRUE during action of step (1), step (3) and step (4) are sequentially executed. When the transition (5) becomes TRUE, a jump sequence to step (1) occurs.

Point

The above program can be created by changing the steps other than the step of the leftmost branch to end steps and changing the end step of the leftmost 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		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 coil HOLD step [SC], the step holds the coil output or action according to its attribute. Processing will proceed to step (7) when the transition (5) becomes TRUE and to step (8) when the transition (6) becomes TRUE.
Convergence		After the step (1) and step (2) immediately before the convergence become active, the transition (3) is checked, then becomes TRUE, the step (4) is activated. The step (1) and step (2) become inactive. However, if it is a coil HOLD step [SC], the step holds the coil output or action according to its attribute.

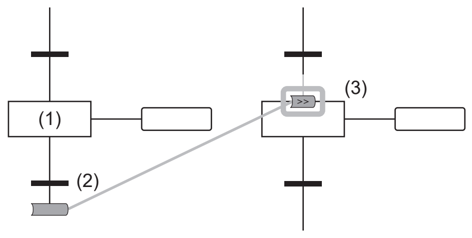
- The simultaneous sequence allows transitions to up to 32 steps.
- If another block is started by the simultaneous sequence, the START source block and START destination block will be executed simultaneously.
- A simultaneous convergence is always performed after a simultaneous branch.

■Precautions

- When the steps merged by parallel convergence include HOLD steps that hold operations, the steps are handled as inactive steps, and the transition to the next step does not occur.
- 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.

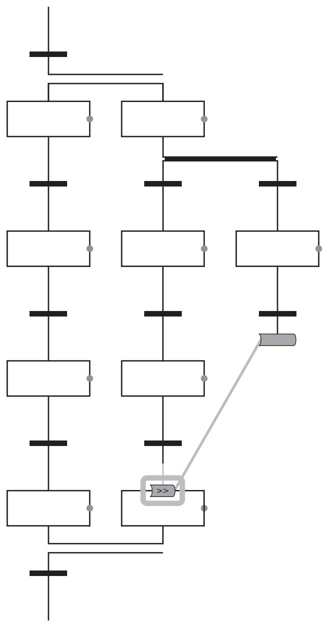


When the transition (2) becomes TRUE while the step (1) is active, the step (3) is activated.
The step (1) becomes inactive. However, if it is a coil HOLD step [SC], the step holds the coil output or action according to its attribute.

- 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.

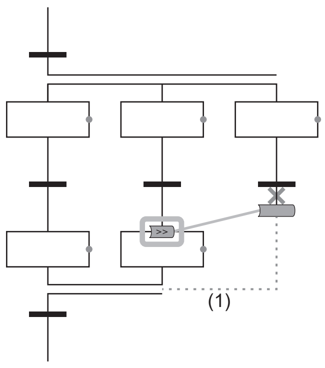
Example of jump sequence that can be specified in the simultaneous branch



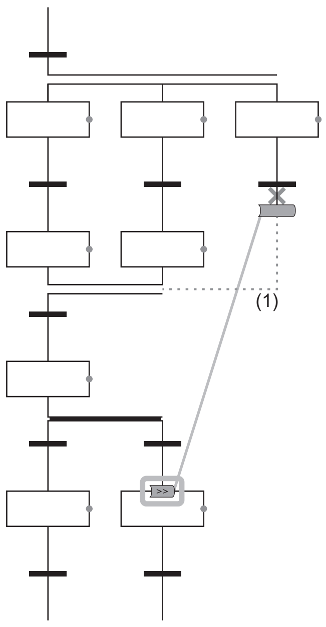
Ex.

Example of jump sequence that cannot be specified in the simultaneous branch

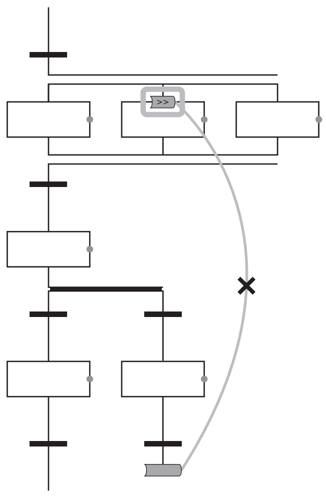
■Jump sequence to another branch within a simultaneous branch



■Jump sequence for exiting from a simultaneous branch



■Jump sequence to a simultaneous branch from outside a simultaneous branch

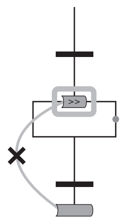


(1) A simultaneous convergence cannot be performed.

■Precautions

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

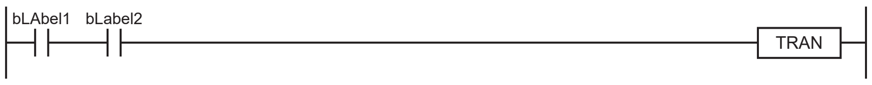
- When a step immediately before the preceding transition is specified



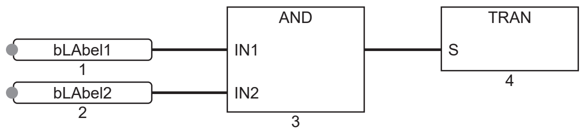
- When current step is specified (when "FX3 compatible transition operation mode setting" is set to "Enable", a step can be specified.)

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	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.  ■Restrictions • Inline ST cannot be used. • Only a TRAN instruction can be input to the coil.

Structured text language		Used to create the following transition program.
		■Method of writing a TRAN function (Creating a dummy transition condition) call statement
		TRAN(bLabel1 & bLabel2);
		//The transition becomes TRUE when the Boolean expression of the input argument is true.
		■Method of writing an assignment statement of Boolean expression for reserved word "TRAN"
		TRAN := bLabel1 & bLabel2;
		//The transition becomes TRUE when the Boolean expression of the right-hand side is true.
		■Method of writing an assignment statement of Boolean expression for the transition name
		Transition1 := bLabel1 & bLabel2;
		//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.

FBD/LD language		Used to create a transition program ending with the TRAN instruction (Creating a dummy transition condition) in a single FBD network block.  AND TRAN bLAbel1 1 IN1 S 4 bLAbel2 2 IN2 3 ■Restrictions • Inline ST cannot be used. • 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. For usable instructions other than the TRAN instruction, refer to the following. ☞ Page 93 Usable instructions

Point

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

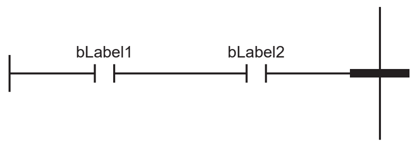
■Usable instructions

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
Association instruction	ANB, ORB
	INV
	MEP, MEF
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□
Character string processing instruction	LD$□, AND$□, OR$□
Creating a dummy transition condition	TRAN

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.

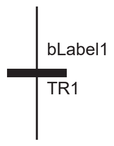
Point

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)

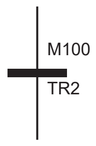
Transition label/device

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

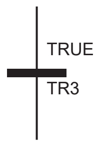
■Bit type label



■Bit device



■Boolean value



Point

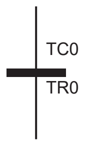
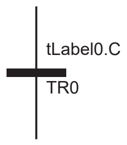
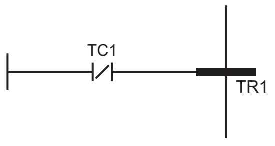
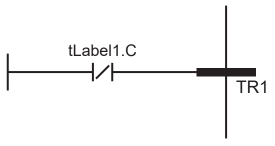
Select a transition name, select [Edit] ⇨ [Modify] ⇨ [Name] in the menu, and input the bit type label, bit device or Boolean value to be specified. (📖GX Works3 Operating Manual)

■Precautions

- When a device (T, ST, C or LC) of timer or counter is used for transition, the device operates as a contact (TS, STS, CS or LCS). Also, when a coil (TC, STC, CC or 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 a timer device is used	When a timer type label is used
 When the contact (TS0) is on, the transition becomes TRUE.	 When the coil of the timer type label (tLabel0) is on, the transition becomes TRUE.
[Direct expression]  When the contact (TS1) is off, the transition becomes TRUE.	[Direct expression]  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) [KnS□][*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. (Kn: K1 to K4)
	MOV(P) [BL□\KnS□]	
	DMOV(P) [KnS□][*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. (Kn: K1 to K8)
	DMOV(P) [BL□\KnS□]	
	BMOV(P) [KnS□][*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. (Kn: K1 to K4)
	BMOV(P) [BL□\KnS□]	
Starting a block	SET [BL□]	Activates the specified block individually and executes a step sequence starting from the initial step.
Ending a block	RST [BL□]	Deactivates the specified block.
Activating a step	SET [S□][*1]	Activates the specified step.
	SET [BL□\S□]	
Deactivating a step	RST [S□][*1]	Deactivates the specified step.
	RST [BL□\S□]	
Step start/end instruction	OUT [S□][*1]	Activates/deactivates the specified step.
	OUT [BL□\S□]	
	ZRST(P) [S□][*1]	Deactivates all specified steps collectively.
	ZRST(P)[BL□\S□]	

*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-F FX5 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 (Start/stop SFC program) is on.
- When using the SFC control instruction, set "SFC Program Setting" to "Use".
- When using the SFC program, do not specify the step relay to the instructions other than the SFC control instruction. If the step relay is specified to the instructions other than the SFC control instruction, the program may perform an unintended operation.
- When "SFC Program Setting" is set to "Not to Use", the instructions which specify the step by using the step relay without block specification (such as LD [S□] and MOV(P) [KnS□]) operate as the normal instructions.
- In the case of the CPU module not corresponding to the SFC program, when the current value is read/written to the step relay with block specification (BL□\S□) by the engineering tool, the current value is read/written to the step relay (S) without block specification. (Page 119 Added and Changed Functions)

  Example: When the current value is read to BL5\S12
  - CPU module corresponding to the SFC program: The current value of BL5\S12 is read.
  - CPU module not corresponding to the SFC program: The current value of S12 is read.

Index modification

The SFC control instructions can specify index-modified step relays (S) and SFC block devices (BL). However, instructions that control a step individually cannot specify index-modified devices.

Device	Index modification target part
S□Z□	Step relay
BL□\S□Z□	Step of step relay with block specification
BL□Z□\S□	Block of step relay with block specification
BL□Z□\S□Z□	Block and step of step relay with block specification
BL□Z□	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 4095
BL□\S□	BL□	0 to 31
	S□	0 to 511
BL□		0 to 31

Point

For details on index modification, refer to the following.
📖MELSEC iQ-F FX5 User's Manual (Application)

8.4 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.

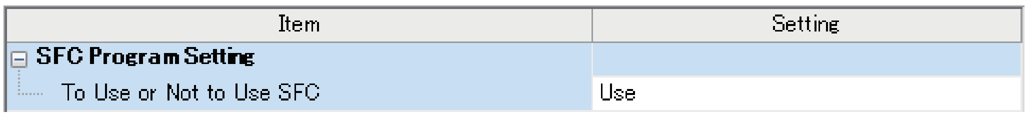
Type	Item	Description
SFC setting	SFC program setting	Set whether to use the SFC program.
	SFC program start mode setting	Set whether to start the SFC program in the initial status (Initial Start) or to start it holding the previous execution status (Resume Start).
	Start conditions setting	Set whether to automatically start and activate the block 0 or to keep it inactive until a start request is issued when starting the SFC program.
	FX3 compatible transition operation mode setting	Set whether to operate the SFC program maintaining compatibility with FX3.

SFC program setting

Set whether to use the SFC program. If the parameter is set to "Not use," other SFC settings cannot be operated.

[CPU Parameter] ⇨ [SFC Setting] ⇨ [SFC Program Setting]

Window



Item	Setting
SFC Program Setting	
To Use or Not to Use SFC	Use

Displayed items

Setting	Description
Not to Use (default)	SFC program is not used.
Use[*1]	SFC program is used.

*1 When the program language of a project is SFC, the default is "Use."

The following tables list the operation which changes according to the SFC program setting and operation depending on the setting of the SFC program setting.

■Operation which changes according to the setting

Operation	Description
SFC program execution	The SFC program can be executed only when "SFC Program Setting" is set to "Use".
Specifying the step relay (S) to the instruction	When "SFC Program Setting" is set to "Use", the step relay (S) cannot be specified to the instructions other than the SFC control instruction.

■Operation depending on the setting

SFC Program Setting	SFC program execution	Specifying the step relay (S) to the instruction
Not to use	Not supported[*1]	No restrictions
Use	Supported	Can be specified only to the SFC control instruction.[*2]

*1 When the SFC program is created, a conversion error occurs in the engineering tool. When the parameter for which "SFC Program Setting" is set to "Not to use" and the SFC program are written separately during the boot operation using the SD memory card, a self-diagnosis error of the CPU module occurs.

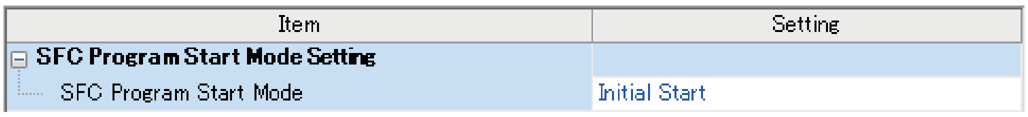
*2 When the step relay (S) is specified to the instructions other than the SFC control instruction, a self-diagnosis error of the CPU module occurs.

SFC Program Start Mode Setting

Set whether to start the SFC program in the initial status (Initial Start) or to start it holding the previous execution status (Resume Start).

[CPU Parameter] ⇨ [SFC Setting] ⇨ [SFC Program Start Mode Setting]

Window



Item	Setting
SFC Program Start Mode Setting	
SFC Program Start Mode	Initial Start

Displayed items

Setting	Description
Initial Start (default)	The program is started after the active state at the previous stop is cleared. The operation after the start is performed according to "Start Condition Setting" of the SFC Setting. (☞ Page 99 Start Conditions Setting)
Resume Start	The program starts in the active state at the previous stop. When setting the resume start, install a battery to the CPU module. (☞ Page 99 Precautions)

The combination of SFC Program Start Mode Setting and SM322 (SFC program startup status) determines whether to perform Initial Start or Resume Start.

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	Initial Start*3	Resume Start	Initial Start
(2)	The CPU module is powered off and on.		Initial Start (M322 is turned off, and the program is initialized.)	Resume Start/Initial Start*4	Resume Start/Initial Start*4*5
(3)	SM321 is turned on and off, or the CPU module is powered off and on after the operating status is changed from RUN to STOP.			Resume Start	Resume Start*5
(4)	CPU module is reset and the operating status is changed to RUN.			Resume Start/Initial Start*4	Resume Start/Initial Start*4*5
(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	Resume Start*5
(6)	Operating status is changed from STOP to RUN.	Resume Start			
(7)	Operating status is STOP, write a program (other than the SFC program), and the status is changed to RUN.	Initial Start		Resume Start	Initial Start
(8)	Operating status is STOP, write a program (SFC 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 "SFC Program Start Mode Setting" is set to "Resume Start" and no changes are made to the program before and after the program is written, the program is resumed.
*3 When the parameter is set to "Initial Start," the on state of M322 is invalid.
*4 Depending on the timing, a program cannot be resumed and starts with initial status.
*5 M322 is turned on, and the program is initialized.

■Precautions

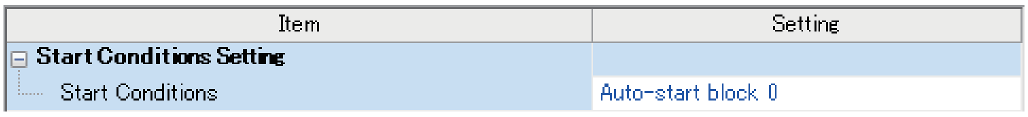
- When setting the parameter to "Resume Start," install a battery in the CPU module. If a battery trouble, such as no battery or battery voltage drop, has occurred, the SFC program may start in the Initial Start mode when it is started first time after the CPU module power is switched from off to on. Battery troubles can be detected by Option Battery Setting. For details, refer to the user's manual for the CPU module used.
- 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 for Resume Start, set them to the latch mode. (MELSEC iQ-F FX5 User's Manual (Application))
- When a program is resumed under a condition other than conditions ((1), (3), (5) in the table) under which the coil output of the coil HOLD step [SC] is turned off, the coil HOLD step [SC] that holds an operation is restarted, but the output is not turned on. To hold the output, set the labels and devices to the latch mode. (MELSEC iQ-F FX5 User's Manual (Application))
- When the CPU module is powered off or reset, the intelligent function module is initialized. To resume a program, it is recommended to create the initial program for the intelligent function module in a block which is constantly active or in a sequence program.
- When the CPU module is powered off or reset, labels and devices are also cleared.
- Depending on the timing, a program may not be resumed after the CPU module is powered off or reset. If a program is started in the initial status after the start mode is set to Resume Start, an event where the program cannot be resumed is stored in the event history. To ensure the program is started in the Resume Start mode, power off or reset the CPU module after switching SM321 from ON to OFF or switch the operating status from RUN to STOP.

Start Conditions Setting

Set whether to automatically start and activate the 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



Item	Setting
⊟ Start Conditions Setting	
Start Conditions	Auto-start block 0

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 issued by the SFC control instruction (Starting a block) 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 Conditions Setting to control the start block 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

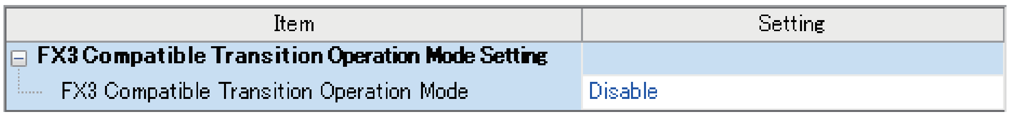
- To execute the SFC program when "Do not auto-start block 0" is set, execute the SET instruction (Starting a block) from the sequence program.
- When "Auto-start block 0," is set, create block 0 without fail.

FX3 compatible transition operation mode setting

This setting is designed to operate the SFC program in the same manner as on FX3. Use this setting to maintain the compatibility with FX3 when the configuration of FX3 used on the user's equipment is replaced with the configuration of FX5.

[CPU Parameter] ⇨ [SFC Setting] ⇨ [FX3 Compatible Transition Operation Mode Setting]

Window



Item	Setting
FX3 Compatible Transition Operation Mode Setting	
FX3 Compatible Transition Operation Mode	Disable

Displayed items

Setting	Description
Disable (default)	Operation of FX5. "Continuous transition" or "No continuous transition" can be specified.
Enable	Operation of FX3. The program operates with continuous transition (operation compatible with FX3).

For the continuous transition, refer to ☞ Page 108 Continuous transition ON/OFF operation.

Point

For the replacement of the SFC program from FX3 to FX5, refer to the following.
📖Transition from MELSEC FX3G, FX3U, FX3UC Series to MELSEC iQ-F Series Handbook

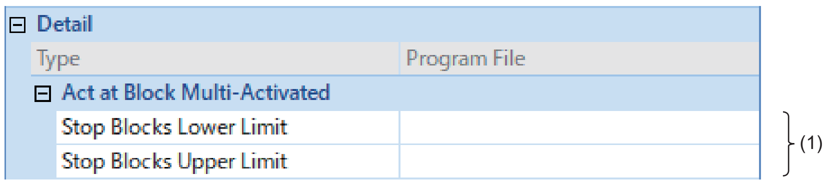
SFC block setting

Act at block Multi-Activated

Set this item to stop the operation of the CPU module when a start request for an already active block is issued in the block start step (with end check) [BC] or block start step (without end check) [BS]. 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

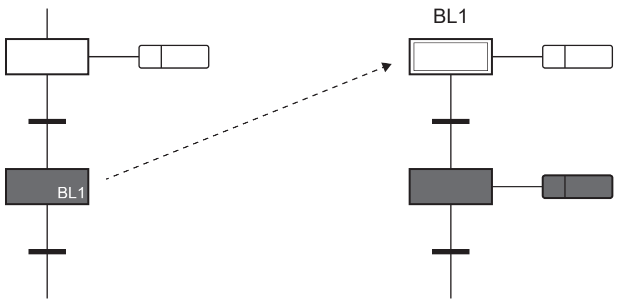


⊟ Detail	
Type	Program File
⊟ Act at Block Multi-Activated	
Stop Blocks Lower Limit	
Stop Blocks Upper Limit	

(1) Set the range of the block to be stopped.

Displayed items

Setting	Description	
No setting (default)	Standby	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.
Block stop range is set	Stop	An error results.



■Precautions

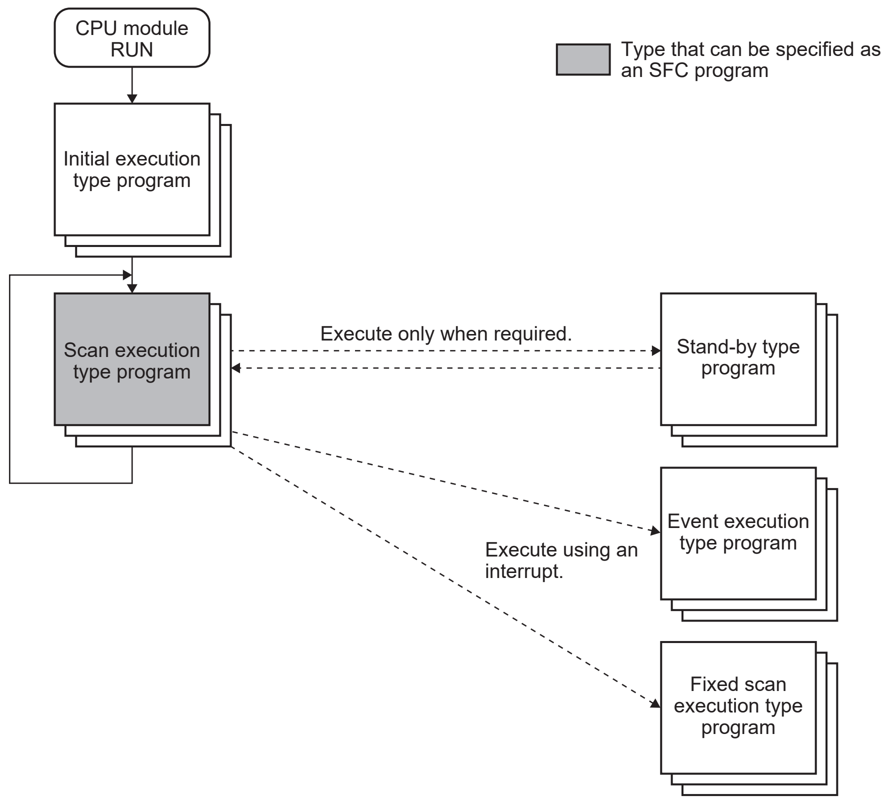
- When the SFC control instruction (Starting a block) is executed for an already active block, the start request is ignored, and the processing of the SFC program is continued.
- If an attempt to transition to an active block start step is made, the activation of the block start step is ignored. The block is not executed again from the initial step.

8.5 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.


CPU module RUN
Type that can be specified as an SFC program
Initial execution type program
Scan execution type program
Execute only when required.
Stand-by type program
Event execution type program
Execute using an interrupt.
Fixed scan execution type program

Execution Type	Specification enable/disable	Remarks
Initial execution type program	×	—
Scan execution type program	○	Only one SFC program can be executed.
Stand-by type program	×	—
Event execution type program	×	—
Fixed scan execution type program	×	—
No specification	○	—

■Precautions

In a project with an SFC program, the step ladder (STL/RETSTL instruction) cannot be used.

SFC program processing sequence

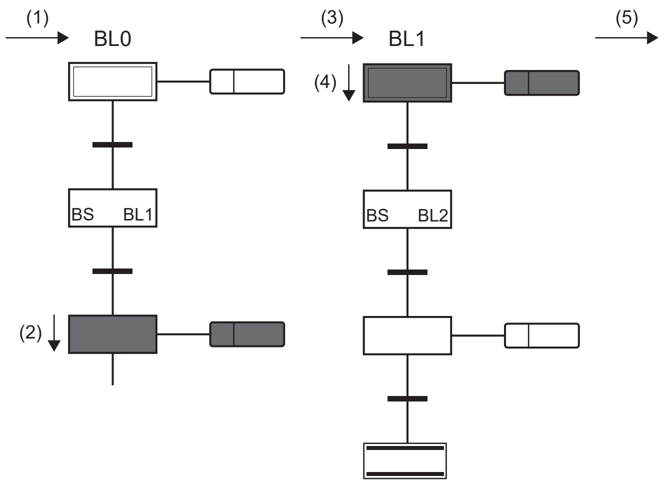
Block execution sequence

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.



Processing is performed in the following order.

(1) Processing of block 0 (BL0)

(2) Execution of the step in block 0 (BL0)

(3) Processing of block 1 (BL1)

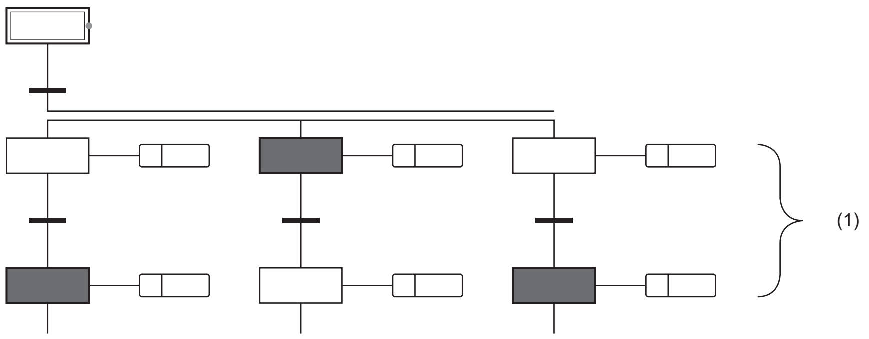
(4) Execution of the initial step of block 1 (BL1)

(5) Processing of the next block

Block 0 can be started automatically when "Auto-start block 0" is specified in "Start Conditions Setting" of the SFC setting. With this setting, even if block 0 reaches the end step and becomes inactive, it will be restarted in the next scan. (☞ Page 99 Start Conditions Setting)

Step execution sequence

In the SFC program, the actions of all active steps are processed within one scan.



(1) All the active steps in the block are executed within a single scan.

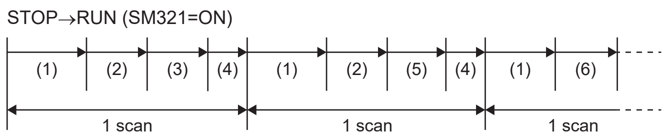
When the action of each step is finished, whether the transition to the next step becomes TRUE or not is checked.

- 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 80 Coil HOLD step [SC])

Ex.

Example of transition operation (without continuous transition)



(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.

■Execution of action

The action performs the following operation for the instruction input condition described in the action depending on the status of the step to be executed.

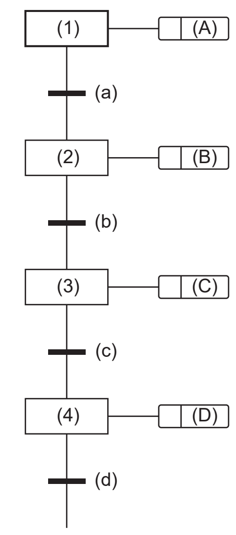
Execution of action	Description
Not executed	The input condition is not reflected in the output.
Execution in the contact ON status	The operation is performed according to the input condition.
Execution in the contact OFF status	The input condition is set to off regardless of the actual input condition, and the operation is performed.

When the step is activated, the action is executed in the contact ON status every scan. When the step is deactivated, the action is executed in the contact OFF status and will not be executed until next time the step is activated. The action is executed in the contact OFF status in the following cases. Execution in the contact OFF status is enabled only in active steps.

- When the transition just after a normal step becomes TRUE.
- When the end step is executed (other than steps that hold operations when SM327 is on and remaining active steps that do not hold operations when SM328 is on)
- When the block is forcibly terminated by using the SFC control instruction (Ending a block).
- When the step is forcibly terminated by the SFC control instruction (Ending a step).
- When SM321 (Start/stop SFC program) is turned off
- When the reset step [R] that has been set to reset the running step is activated
- When the reset step [R] that has been set to reset the steps that hold operations is activated and S999 has been specified for the reset step [R]

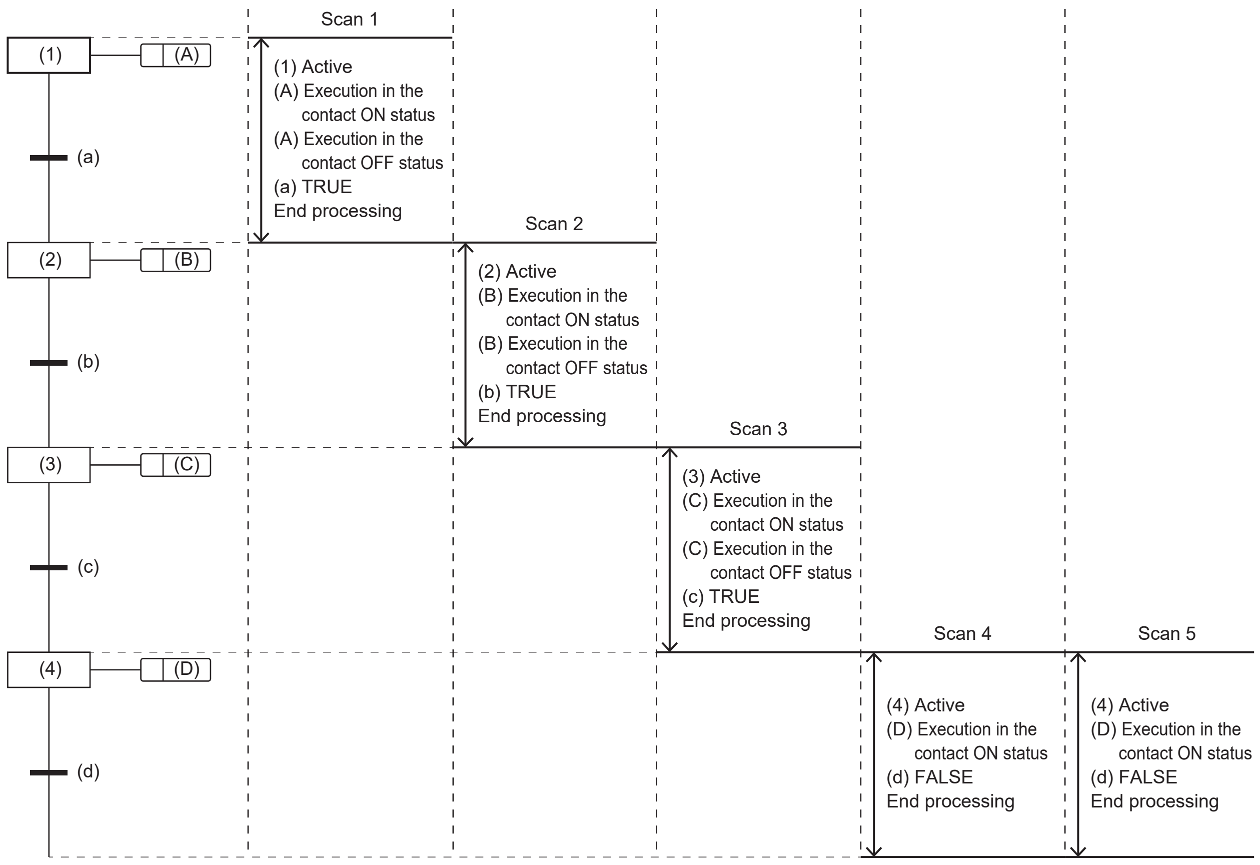
■Transition operation

The transition operation with/without continuous transition in the following SFC program is shown below. (☞ Page 108 Continuous transition ON/OFF operation)



Steps (1) to (4) are activated, and the actions (A) to (D) are executed. Although the transition (a) to (c) for Steps (1) to (3) become TRUE, the transition (d) for Step (4) becomes FALSE.

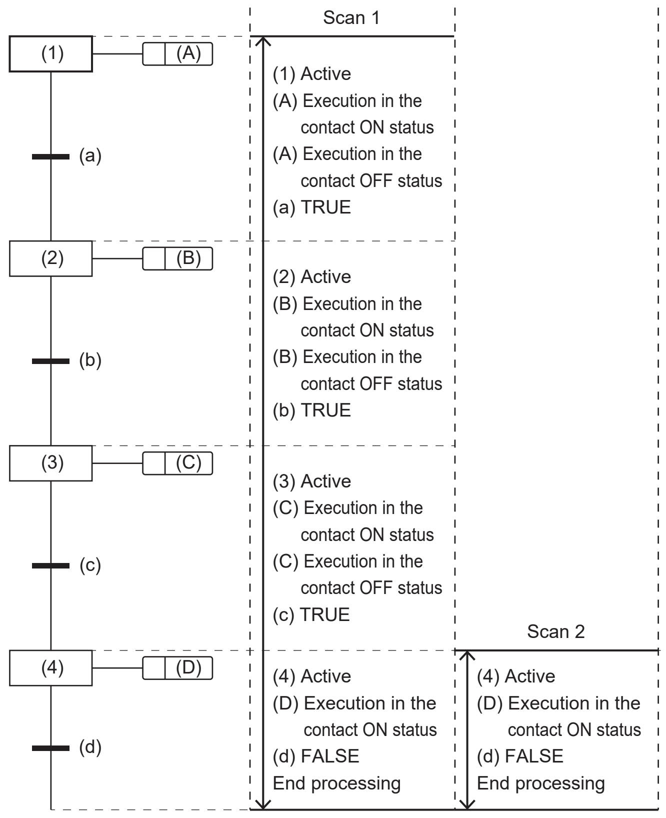
- Without continuous transition


Scan 1
(1)
(A)
(1) Active
(A) Execution in the contact ON status
(A) Execution in the contact OFF status
(a) TRUE
End processing
(a)
Scan 2
(2)
(B)
(2) Active
(B) Execution in the contact ON status
(B) Execution in the contact OFF status
(b) TRUE
End processing
(b)
Scan 3
(3)
(C)
(3) Active
(C) Execution in the contact ON status
(C) Execution in the contact OFF status
(c) TRUE
End processing
(c)
Scan 4
Scan 5
(4)
(D)
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing
(d)

Scan	Description
Scan 1	Step (1) is activated and the action (A) is executed in the contact OFF status after the execution in the contact ON status.[*1]
Scan 2	Step (2) is activated and the action (B) is executed in the contact OFF status after the execution in the contact ON status.[*1]
Scan 3	Step (3) is activated and the action (C) is executed in the contact OFF status after the execution in the contact ON status.[*1]
Scan 4	Step (4) is activated and the action (D) is executed in the contact ON status.
Scan 5 and after	Step (4) is active until the transition (d) becomes TRUE, and the action (D) is executed in the contact ON status.

*1 In the case of coil HOLD step the action is not executed in the contact OFF status.

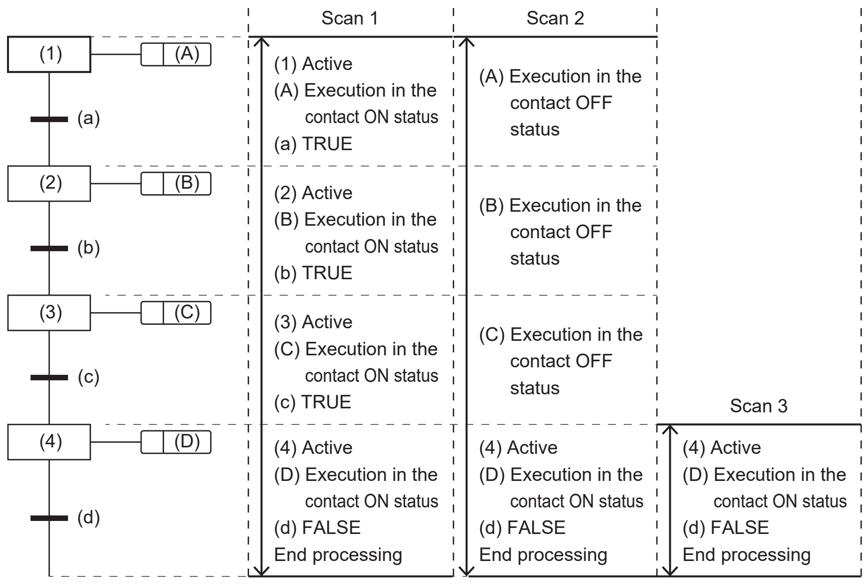
• With continuous transition



Scan 1
Scan 2
(1)
(A)
(a)
(2)
(B)
(b)
(3)
(C)
(c)
(4)
(D)
(d)
(1) Active
(A) Execution in the contact ON status
(A) Execution in the contact OFF status
(a) TRUE
(2) Active
(B) Execution in the contact ON status
(B) Execution in the contact OFF status
(b) TRUE
(3) Active
(C) Execution in the contact ON status
(C) Execution in the contact OFF status
(c) TRUE
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing

Scan	Description
Scan 1	Steps (1) to (4) are continuously activated. When each step is activated, the actions (A) to (C) are executed in the contact OFF status after the execution in the contact ON status, and the action (D) is executed in the contact ON status. However, in the case of coil HOLD step the action is not executed in the contact OFF status.
Scan 2 and after	Step (4) is active until the transition (d) becomes TRUE, and the action (D) is executed in the contact ON status.

• With continuous transition (operation compatible with FX3)



Scan 1
Scan 2
Scan 3
(1)
(A)
(a)
(2)
(B)
(b)
(3)
(C)
(c)
(4)
(D)
(d)
(1) Active
(A) Execution in the contact ON status
(a) TRUE
(2) Active
(B) Execution in the contact ON status
(b) TRUE
(3) Active
(C) Execution in the contact ON status
(c) TRUE
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing
(A) Execution in the contact OFF status
(B) Execution in the contact OFF status
(C) Execution in the contact OFF status
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing
(4) Active
(D) Execution in the contact ON status
(d) FALSE
End processing

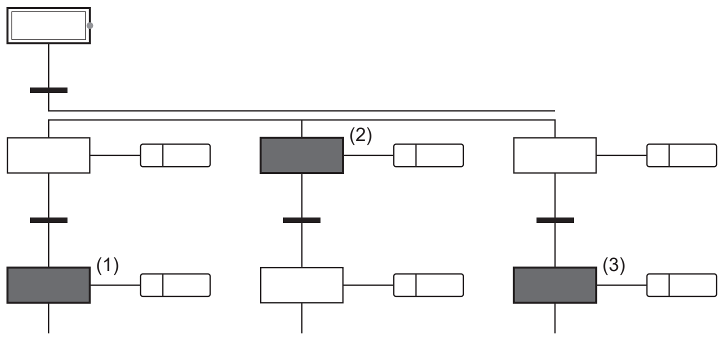
Scan	Description
Scan 1	Steps (1) to (4) are continuously activated. When each step is activated, the actions (A) to (D) are executed in the contact ON status.
Scan 2	The actions (A) to (C) are executed in the contact OFF status, and the action (D) is executed in the contact ON status. However, in the case of coil HOLD step the action is not executed in the contact OFF status.
Scan 3 and after	Step (4) is active until the transition (d) becomes TRUE, and the action (D) is executed in the contact ON status.

■Precautions

- If the transition for a step becomes TRUE at the first execution, the step starts/ends after one scan. When the step ends after one scan, the following operation is performed according to the transition condition.

No continuous transition	Continuous transition	With continuous transition (operation compatible with FX3)
I/O refresh of coil output, etc. is not reflected. To reflect the I/O refresh, design the program so that one step is scanned several times. However, turning on of the step relay can be detected before the previous scan (just after the transition from the previous step) even in another program.	I/O refresh of coil output, etc. is not reflected, and turning on of coil output cannot be detected in any other program. To reflect the I/O refresh, design the program so that one step is scanned several times.	The step is started and ended in different scans, and coil output in an action can be detected by another program. However, the step relay is turned off in a scan in which the transition becomes TRUE.

- 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.

- In the case of "Continuous transition," when a transition from one step to multiple steps occurs due to jump sequence or selective convergence, the action of one step may be performed twice in one scan.
- In the case of "Continuous transition," if a program that loops using the jump sequence is executed, an error will occur.
- In the case of "Continuous transition" (operation compatible with FX3) where the transition destination step of jump sequence is positioned above the transition source in the SFC chart, the destination step will be executed in the next scan even if the transition becomes TRUE.

Continuous transition ON/OFF operation

The transition conditions for the SFC program include "With continuous transition," "With continuous transition (operation compatible with FX3)" and "Without continuous transition."

The setting "With continuous transition" or "Without continuous transition" is determined by "FX3 Compatible Transition Operation Mode Setting" and SM323 (presence/absence of continuous transition for entire block).

FX3 compatible transition operation mode setting	SM323	Description	
Disable	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. The action of the transition source step is executed in the contact OFF status in a scan in which the transition becomes TRUE.
Enable	ON/OFF	With continuous transition (operation compatible with FX3)	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. Unlike normal continuous transitions, the action of the destination step is executed in the contact OFF status in the next scan where the transition becomes TRUE.

Point

- The tact time can be reduced by setting the condition to "Continuous transition" or "Continuous transition (operation compatible with FX3)." Accordingly, the wait time from when the transition becomes TRUE until the action of the destination step is executed can be eliminated. However, the setting to "Continuous transition" or "Continuous transition (operation compatible with FX3)" may slow down the operation of other blocks and sequence programs.
- SM324 (Continuous transition disable flag) is turned off only when "Continuous transition" is set. (Because the system is basically turned on automatically when the SFC program is executed, SM324 is always ON. Also, when "Continuous transition" (FX3 compatible operation) is set, SM324 is always ON.) Therefore, continuous transition can be prohibited by using SM324 as a transition condition.
- When "FX3 Compatible Transition Operation Mode Setting" is valid, "Continuous transition (operation compatible with FX3)" is enabled regardless of whether SM323 is on or off.

8.6 SFC Program Execution

Starting and stopping the SFC program

The SFC program can be started and stopped by either of the following methods.

- Auto-start by the CPU parameter
- Starting and stopping the program by the special relay (SM321)

Auto-start by the CPU parameter

Set "Start Conditions Setting" to "Auto-start block 0," 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 99 Start Conditions Setting)

Starting and stopping the program by the special relay (SM321)

SM321 (Start/stop SFC program) automatically turns on at execution of the SFC program.

- The program execution can be stopped by turning off SM321.
- The terminated SFC program can be re-executed by turning on SM321.

Point

Set the CPU parameter "SFC Program Start Mode Setting" to "Resume Start," and the SFC program can be resumed. (☞ Page 98 SFC Program Start Mode Setting)

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
Auto-start by the CPU parameter (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 99 Start Conditions Setting
Start by the 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 81 Block start step (with END check) [BC] ☞ Page 82 Block start step (without END check) [BS]
Start by the SFC control instruction	Start the block specified by the SFC control instruction from the action of the SFC program or another sequence program. • 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□]/OUT [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 95 SFC Control Instructions
Start by the 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 by the 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 82 End step
End by the SFC control instruction	End and deactivate the block specified by the RST [BL□] (Ending a block) instruction from the action of the SFC program or another sequence program. (The block ends when all the active steps in the specified block are deactivated by using the RST [BL□\S□]/OUT [BL□\S□]/ ZRST(P)[S□/BL□\S□] (Ending a block) instruction.)	This method is effective to end processing regardless of the operation status, such as an emergency stop.	☞ Page 95 SFC Control Instructions
End by the 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

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
Activation by the transition condition	The transition is checked at the end of the step. If it is TRUE, the next step is automatically activated.	—	☞ Page 87 Transition
Activation by the SFC control instruction	Activate the step specified by the SET [S□/ BL□\S□]/OUT[S□/BL□\S□] (Activating a step) instruction from the action of the SFC program or another sequence program.	—	☞ Page 95 SFC Control Instructions
Activation by the engineering tool	• 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

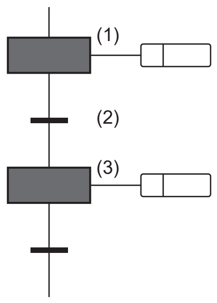
A step in the SFC program can be deactivated by either of the following methods.

Item	Method	Remarks	Reference
Deactivation by the transition condition	The transition is checked at the end of the step. If it is TRUE, the current step is automatically deactivated.	—	☞ Page 87 Transition
Deactivation by the reset step [R]	Activating this step deactivates the step specified for attribute target.	This method is effective to deactivate the coil HOLD step [SC] when the sequence for error processing is selected in the selection branch.	☞ Page 81 Reset step [R]
Deactivation by the SFC control instruction	End the step specified by the RST [S□/ BL□\S□]/OUT[S□/BL□\S□]/ZRST(P)[S□/ BL□\S□] (Deactivating a step) instruction from the action of the SFC program or another sequence program.	When all the active steps in the specified block are deactivated by the SFC control (Deactivating a step) instruction, the block also ends.	☞ Page 95 SFC Control Instructions
Deactivation by the engineering tool	• End 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



When the transition (2) becomes TRUE, the step (1) becomes inactive.
The step (3) of the transition which is supposed to be multi-activated is activated.

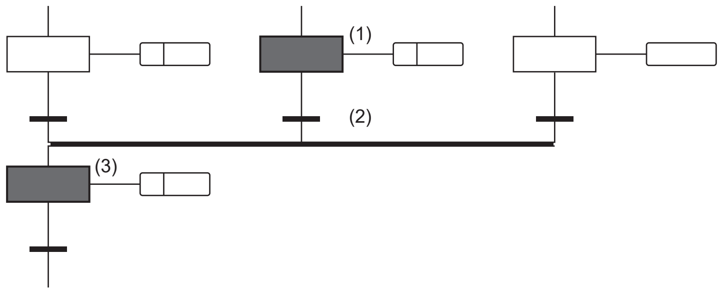
Selective sequence

■Divergence

The transition condition is checked starting from the left side, and, if the destination of the branch where the transition becomes TRUE is an active step, the same operation as the series sequence is performed. In the same manner as in the normal selective sequence, the transition condition is not checked in the lines other than the branch where the double start becomes TRUE.

■Convergence

The same operation as the series sequence is performed.

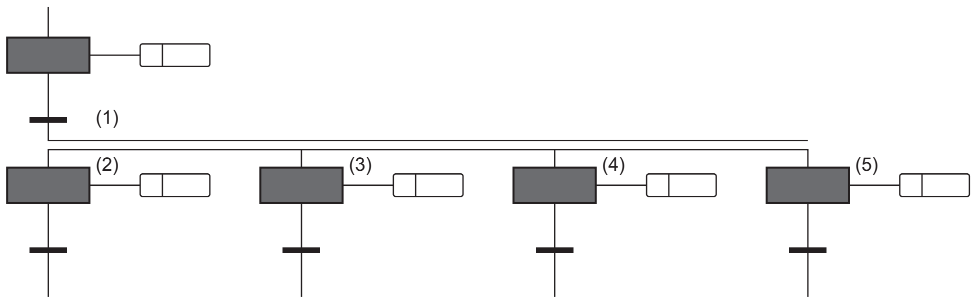


When the transition (2) becomes TRUE, the step (1) becomes inactive.
The step (3) of the transition which is supposed to be multi-activated is activated.

Simultaneous sequence

■Divergence

All steps below the transition become active in the next scan.



When the transition (1) becomes TRUE, the steps (2) to (5) which are supposed to be multi-activated at the next scan are all activated.

■Convergence

The transition source becomes inactive. The coil HOLD step [SC] holds operation.

Operation when a program is modified

To change an SFC program, use the following function.

- Write to the programmable controller

The following table shows the SFC program data that can be changed by the above method.

Change type			**Write to the programmable controller**	
			STOP/PAUSE	**RUN**
SFC program addition			○	×
SFC block addition/deletion			○	×
SFC block change	SFC diagram change	Step/transition condition addition/deletion	○	×
		Transition condition (branch/ convergence/jump) change	○	×
		Step attribute change	○	×
	Change in SFC diagram	Operation output program change	○	×
		Transition program change	○	×
	Block information change		○	×

When data is written to the programmable controller

Shown here is the operation when an SFC program is modified by writing data to the programmable controller.

SM322 (SFC program startup status)	**Program modification status**	
	Modified	**Not modified**
OFF (Initial start)	Initial start	Initial start
ON (Resumption)	Initial start	Resumption

■When the operating status is changed from STOP to RUN

If the CPU module is stopped during execution (running) of an SFC program, the device state and the SFC program active/inactive state will be restored to the state immediately before the stop when the CPU module status is changed from STOP to RUN. The program will start in the Resume Start mode regardless of the CPU parameter setting "SFC Program Start Mode Setting."

If the sequence program file (including an SFC program) or the FB file is written to the CPU module while it is in the STOP status, the SFC program will start in the Initial Start mode if the SFC program exists when the operating status is changed back to RUN. Note that it may start in the Resume Start mode if there are no changes before and after writing of the SFC program. (☞ Page 98 SFC Program Start Mode Setting)

■Precautions

- After an SFC program is modified by writing data to the programmable controller, reset the CPU module, and execute the SFC program.
- If the CPU parameter "SFC Program Start Mode Setting" has been set to "Resume Start," once turn off (Initial Start) SM322 (SFC program startup status), and modify the program by writing data to the programmable controller. Thereafter, start the SFC program in the Initial Start mode, and then turn on (Resume Start) SM322 again.

Checking SFC program operation

The functions of the engineering tool that can be used to check the SFC program operation are shown below.

- Monitor
- Watch
- Device/buffer memory batch monitor
- Control SFC steps
- SFC block list
- SFC all blocks batch monitor
- Active step monitor

Point

- For details on each functions and operation check methods, refer to the following.

📖GX Works3 Operating Manual

- When using the SFC program, the data cannot be written to the step relay (S) by the engineering tool. The data can be read from the step relay (S).

APPENDICES

Appendix 1 Operations of when the MC/MCR instructions are used to control EN

The following table lists operations of instructions, devices, and labels used in a function block when "Use MC/MCR to Control EN" is enabled in the inherent property setting of the function block.

Instruction/device/label used in a function block	**Operation of Instruction/device/label used in a function block**	
	When "Yes" is selected for "Use MC/MCR to Control EN"	**When "No" is selected for "Use MC/MCR to Control EN"**
Instructions executed at the rising edge or falling edge (PLS instruction, instructions for conversion to pulses (□P))[*1]	The next time EN is turned on, the instruction is executed if the condition contact is TRUE.	The next time EN is turned on, the instruction may not be executed even though the condition contact is TRUE.
Timer (low-speed timer/timer/high-speed timer)	The count value becomes 0 and the both coil and contact turn off.	The state of devices remains unchanged.
Retentive timer (low-speed timer/timer/high-speed timer), counter, long counter	The coil turns off, but the current count value and the current state of the contact remain unchanged.	The state of devices remains unchanged.
Devices specified as the device part of the OUT instruction	The devices are forcibly turned off.	The state of devices remains unchanged.

*1 Instructions specified in the coil side apply.

Restriction

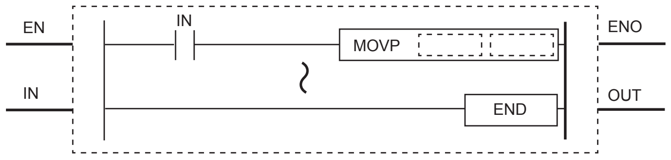
When "Yes" is selected for "Use MC/MCR to Control EN", do not use the MC/MCR instructions while the function block is being executed. If the MC/MCR instructions are used, the EN control may not operate properly.

Instructions executed at the rising/falling edge

The following describes operations of instructions executed at the rising/falling edge.

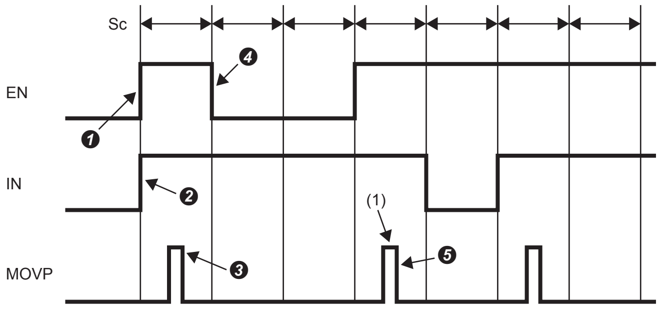
Ex.

A subroutine-type FB using an instruction executed at the rising edge



■When "Yes" is selected for "Use MC/MCR to Control EN"

When EN is turned on, the instruction is executed if the condition contact is TRUE ((1) in the following figure).



Sc: Scan

❶ EN is turned on. (User operation)

❷ IN is turned on. (User operation)

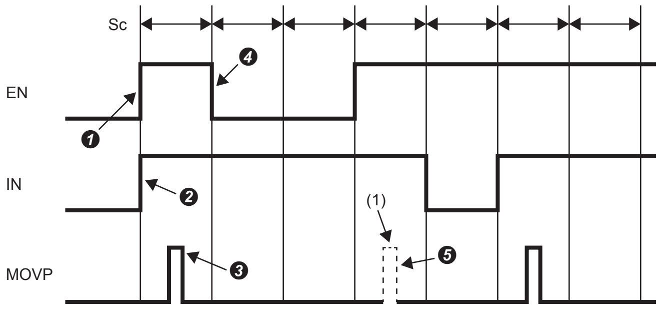
❸ The MOVP instruction is executed. (CPU module operation)

❹ EN is turned off. (User operation)

❺ The MOVP instruction is executed. (CPU module operation)

■When "No" is selected for "Use MC/MCR to Control EN"

When EN is turned off, operations of the instruction differ depending on the condition contact status ((1) in the following figure).



Sc: Scan

❶ EN is turned on. (User operation)

❷ IN is turned on. (User operation)

❸ The MOVP instruction is executed. (CPU module operation)

❹ EN is turned off. (User operation)

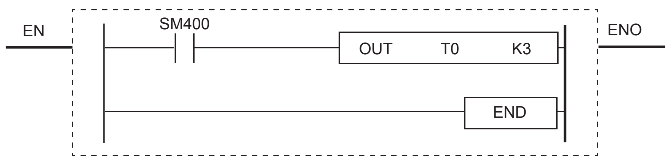
❺ The MOVP instruction is executed when the condition contact is FALSE immediately before EN is turned off at ❹. (CPU module operation) (The MOVP instruction is not executed when the condition contact is TRUE immediately before EN is turned off at ❹.)

Timer (low-speed timer/timer/high-speed timer)

The following describes operations of the timer (low-speed timer/timer/high-speed timer).

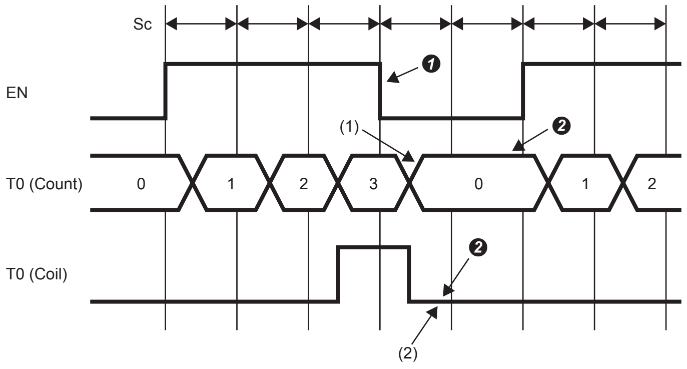
Ex.

A subroutine-type FB using a low-speed timer



■When "Yes" is selected for "Use MC/MCR to Control EN"

The count value becomes 0 ((1) in the following figure). The coil turns off ((2) in the following figure).



Sc: Scan

T0 (Count): T0 (count value)

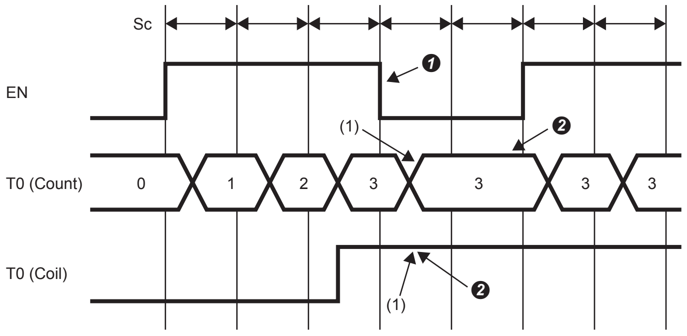
T0 (Coil): T0 (coil)

❶ EN is turned off. (User operation)

❷ The coil turns off and the timer value and the count value are cleared. (CPU module operation)

■When "No" is selected for "Use MC/MCR to Control EN"

The current count value and the current state of the coil remain unchanged. ((1) in the following figure).



Sc: Scan

T0 (Count): T0 (count value)

T0 (Coil): T0 (coil)

❶ EN is turned off. (User operation)

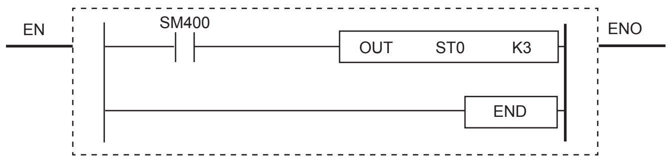
❷ The values remain unchanged. (CPU module operation)

Retentive timer (low-speed timer/timer/high-speed timer), counter, and long counter

The following describes operations of the retentive timer (low-speed timer/timer/high-speed timer), counter, and long counter.

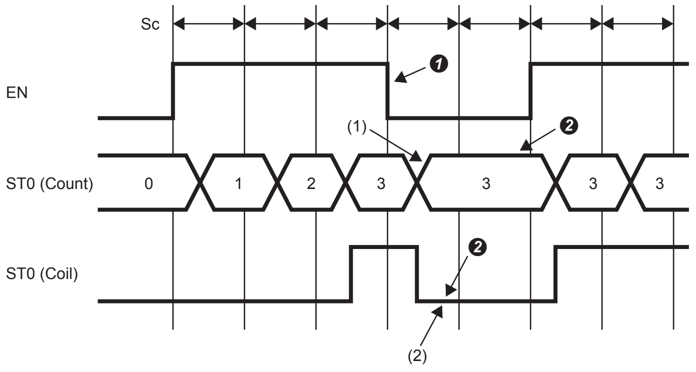
Ex.

A subroutine-type FB using a low-speed retentive timer



■When "Yes" is selected for "Use MC/MCR to Control EN"

The current count value remain unchanged. ((1) in the following figure). The coil turns off ((2) in the following figure).



Sc: Scan

ST0 (Count): T0 (count value)

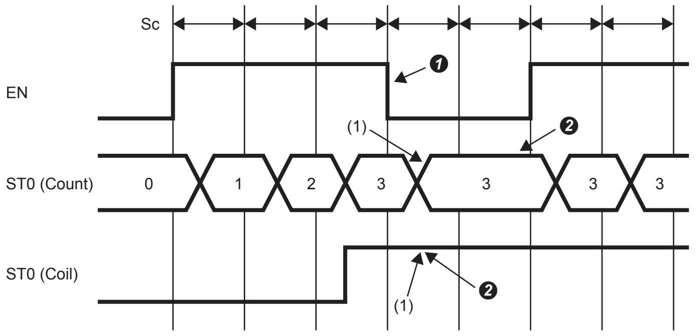
ST0 (Coil): T0 (coil)

❶ EN is turned off. (User operation)

❷ The coil turns off, but the current count value and the current state of the contact remain unchanged. (CPU module operation)

■When "No" is selected for "Use MC/MCR to Control EN"

The current count value and the current state of the coil remain unchanged. ((1) in the following figure).



Sc: Scan

ST0 (Count): T0 (count value)

ST0 (Coil): T0 (coil)

❶ EN is turned off. (User operation)

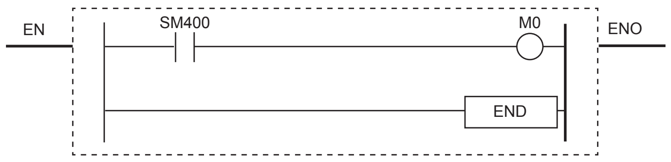
❷ The values remain unchanged. (CPU module operation)

Devices specified as the device part of the OUT instruction

The following describes operations of devices specified as the device part of the OUT instruction.

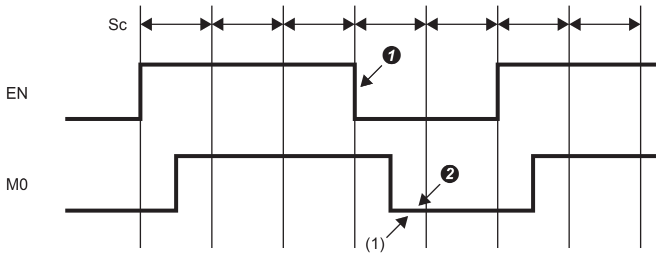
Ex.

A subroutine-type FB using M0 for the device part of the OUT instruction.



■When "Yes" is selected for "Use MC/MCR to Control EN"

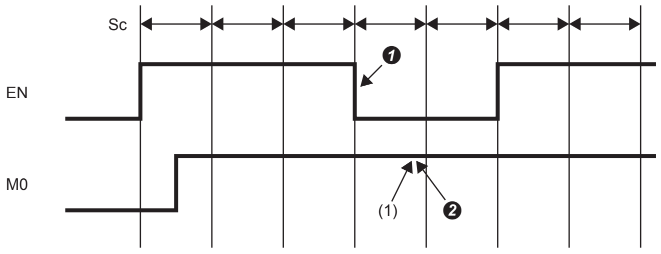
M0 is forcibly turned off ((1) in the following figure).



Sc: Scan
❶ EN is turned off. (User operation)
❷ The coil turns off. (CPU module operation)

■When "No" is selected for "Use MC/MCR to Control EN"

The current state of M0 remains unchanged ((1) in the following figure).



Sc: Scan
❶ EN is turned off. (User operation)
❷ The state of coil remains unchanged. (CPU module operation)

Appendix 2 Added and Changed Functions

The functions added or changed with the CPU module and engineering tool, and the supported CPU modules' firmware version and engineering tool software version are given below. Firmware version can be checked through the module diagnostics (CPU diagnostics). For the module diagnostics (CPU diagnostics), refer to the following manuals.
The firmware version can be confirmed with module diagnosis (CPU diagnosis). Refer to the following manuals for details on diagnosing the module (CPU diagnosis).
📖MELSEC iQ-F FX5S/FX5UJ/FX5U/FX5UC User's Manual (Hardware)
For software version, refer to the 📖GX Works3 Operating Manual.

FX5S CPU module

Add/Change Function	Supported CPU module firmware version	Supported engineering tool software version	Reference
Applicable to FX5S CPU module • Ladder diagram • Structured text language • FBD/LD language	From the first	"1.080J" or later	—
Unicode character strings are supported.	From the first	"1.080J" or later	—

FX5UJ CPU module

Add/Change Function	Supported CPU module firmware version	Supported engineering tool software version	Reference
Applicable to FX5UJ CPU module • Ladder diagram • Structured text language • FBD/LD language	From the first	1.060N and above	—
Unicode character strings are supported.	"1.030" or later	"1.085P" or later	—

FX5U/FX5UC CPU module

Add/Change Function	Supported CPU module firmware version	Supported engineering tool software version	Reference
Applicable to FX5U/FX5UC CPU module • Ladder language • Structured text language • FBD/LD language	From the first	From the first	—
Applicable to SFC program	"1.220" or later	"1.070Y" or later	Page 71 SFC PROGRAM
Unicode character strings are supported.	"1.240" or later	"1.075D" or later	—

INDEX

REVISIONS

Revision date	Revision	Description
October 2014	A	First Edition
January 2015	B	■Added functions FBD/LD language ■Added or modified parts Chapter 1, Section 3.1, 3.2, 4.1, 4.3, 4.4, 4.5, 5.2, 5.3, Chapter 6, 7
April 2015	C	A part of the cover design is changed.
August 2015	D	■Added or modified parts Section 4.4, Chapter 7
July 2018	E	■Added or modified parts RELEVANT MANUALS, TERMS, Section 3.2, 4.4, 5.2, 5.3, 6.1, 7.1
December 2018	F	■Added or modified parts RELEVANT MANUALS, TERMS, Chapter 2, 3, Section 6.1, Appendix 1, TRADEMARKS
October 2019	G	■Added models FX5UJ CPU module ■Added or modified parts RELEVANT MANUALS, TERMS, Section 3.4, 5.3, 6.1
May 2020	H	■Added or modified parts RELEVANT MANUALS, TERMS, Section 3.2, 6.1, TRADEMARKS
October 2020	J	■Added functions SFC program ■Added or modified parts RELEVANT MANUALS, Chapter 1, Section 5.2, 6.1, 7.1, Chapter 8, Appendix 2
April 2021	K	■Added or modified parts RELEVANT MANUALS, TERMS, Section 3.2, 3.3, 4.3, 4.4, 4.6, 6.1, 7.1, Appendix 2, TRADEMARKS
April 2022	L	■Added model FX5S CPU module ■Added or modified parts RELEVANT MANUALS, TERMS, GENERIC TERMS AND ABBREVIATIONS, Chapter 1, Section 3.2, 3.3, 5.2, 7.1, 7.2, 8.2, Appendix 2, WARRANTY

This manual confers no industrial property rights or any 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. Relay failure or output contact failure caused by usage beyond the specified life of contact (cycles).
6. 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.
7. Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
8. 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.

6. Product application

(1) In using the Mitsubishi MELSEC programmable controller, the usage conditions shall be that the application will not lead to a major accident even if any problem or fault should occur in the programmable controller device, and that backup and fail-safe functions are systematically provided outside of the device for any problem or fault.

(2) The Mitsubishi programmable controller has been designed and manufactured for applications in general industries, etc. Thus, applications in which the public could be affected such as in nuclear power plants and other power plants operated by respective power companies, and applications in which a special quality assurance system is required, such as for railway companies or public service purposes shall be excluded from the programmable controller applications.
In addition, applications in which human life or property that could be greatly affected, such as in aircraft, medical applications, incineration and fuel devices, manned transportation, equipment for recreation and amusement, and safety devices, shall also be excluded from the programmable controller range of applications.
However, in certain cases, some applications may be possible, providing the user consults their local Mitsubishi representative outlining the special requirements of the project, and providing that all parties concerned agree to the special circumstances, solely at the user's discretion.

(3) Mitsubishi shall have no responsibility or liability for any problems involving programmable controller trouble and system trouble caused by DoS attacks, unauthorized access, computer viruses, and other cyberattacks.

TRADEMARKS

Anywire and AnyWireASLINK are either registered trademarks or trademarks of Anywire Corporation.
Unicode is either a registered trademark or a trademark of Unicode, Inc. in the United States and other countries.
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.

Manual number: JY997D55701L
MODEL: FX5-P-PS-E
MODEL CODE: 09R538

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.