MITSUBISHI

Mitsubishi Programmable Controller

MELSEC Q series   **MELSEC L series**   **MELSEC-F**

MELSEC-Q/L/F Structured Programming Manual

Fundamentals

QSERIES
LSERIES
FSERIES

● SAFETY PRECAUTIONS ●

(Always read these instructions before using this product.)

Before using the MELSEC-Q series, MELSEC-L series, and MELSEC-F series programmable controllers, thoroughly read the manuals attached to the products and the relevant manuals introduced in the attached manuals. Also pay careful attention to safety and handle the products properly.

Please keep this manual in a place where it is accessible when required and always forward it to the end user.

●CONDITIONS OF USE FOR THE PRODUCT●

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

i) where any problem, fault or failure occurring in the PRODUCT, if any, shall not lead to any major or serious accident; and

ii) where the backup and fail-safe function are systematically or automatically provided outside of the PRODUCT for the case of any problem, fault or failure occurring in the PRODUCT.

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

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

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

• Nuclear Power Plants and any other power plants operated by Power companies, and/or any other cases in which the public could be affected if any problem or fault occurs in the PRODUCT.

• Railway companies or Public service purposes, and/or any other cases in which establishment of a special quality assurance system is required by the Purchaser or End User.

• Aircraft or Aerospace, Medical applications, Train equipment, transport equipment such as Elevator and Escalator, Incineration and Fuel devices, Vehicles, Manned transportation, Equipment for Recreation and Amusement, and Safety devices, handling of Nuclear or Hazardous Materials or Chemicals, Mining and Drilling, and/or other applications where there is a significant risk of injury to the public or property.

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

REVISIONS

Print date	Manual number	Revision
Jul., 2008	SH(NA)-080782ENG-A	First edition A-3
Jan., 2009	SH(NA)-080782ENG-B	**Model Addition** Q00UJ, Q00U, Q01U, Q10UDH, Q10UDEH, Q20UDH, Q20UDEH, FX series **Addition** MANUALS **Correction** Generic Terms and Abbreviations in This Manual, Section 1.3, Section 4.3.3, Section 4.3.4, Section 4.4.1, Section 4.4.2, Appendix 1
Jul., 2009	SH(NA)-080782ENG-C	**Model Addition** Q00J, Q00, Q01 **Addition** Section 3.5, Section 3.6, Section 4.4, Section 4.4.1, Section 4.4.2, Section 4.4.3, Section 4.4.4, Section 4.4.5, Section 4.4.6, Section 4.6, Section 4.8, Appendix 1 **Correction** PURPOSE OF THIS MANUAL is changed to Section 1.2, Generic Terms and Abbreviations in This Manual is changed to Section 1.3, Section 1.5, Section 4.2.8, Section 4.3.4, Section 4.5 is changed to Section 4.4.7, Section 4.5.2, Section 4.5.3, Section 4.6 is changed to Section 4.4.8, Appendix 2, Appendix 3, Appendix 1 to 2 are changed to Appendix 2 to 3
Oct., 2009	SH(NA)-080782ENG-D	**Correction** Section 1.3, Section 5.2.2, Appendix 2
Jan., 2010	SH(NA)-080782ENG-E	**Model Addition** L02, L26-BT **Addition** CONDITIONS OF USE FOR THE PRODUCT **Correction** MANUALS, Section 1.2, Section 1.3, Section 1.5, Section 4.4.1, Section 4.4.4, Section 4.5.1, Section 4.5.2, Section 4.6

INTRODUCTION

Thank you for purchasing the Mitsubishi MELSEC-Q series/MELSEC-L series/MELSEC-F series programmable controller.
Before using the product, thoroughly read this manual to develop full familiarity with the programming specifications to ensure correct use.
Please forward this manual to the end user.

CONTENTS

MANUALS

The manuals related to this product are shown below.

Refer to the following tables when ordering required manuals.

(1) Structured programming

Manual name	Manual number (Model code)
MELSEC-Q/L Structured Programming Manual (Common Instructions) Explains the specifications and functions of common instructions such as sequence instructions, basic instructions, and application instructions that can be used in structured programs. (Sold separately)	SH-080783ENG (13JW07)
MELSEC-Q/L Structured Programming Manual (Application Functions) Explains the specifications and functions of application functions that can be used in structured programs. (Sold separately)	SH-080784ENG (13JW08)
MELSEC-Q/L Structured Programming Manual (Special Instructions) Explains the specifications and functions of special instructions such as module dedicated instruction, PID control instruction, and built-in I/O function dedicated instruction, that can be used in structured programs. (Sold separately)	SH-080785ENG (13JW09)
FXCPU Structured Programming Manual (Device & Common) Explains the devices and parameters provided in GX Works2 for structured programming. (Sold separately)	JY997D26001 (09R925)
FXCPU Structured Programming Manual (Basic & Applied Instruction) Explains the sequence instructions provided in GX Works2 for structured programming. (Sold separately)	JY997D34701 (09R926)
FXCPU Structured Programming Manual (Application Functions) Explains the application functions provided in GX Works2 for structured programming. (Sold separately)	JY997D34801 (09R927)

(2) Operation of GX Works2

Manual name	Manual number (Model code)
GX Works2 Version1 Operating Manual (Common) Explains the system configuration of GX Works2 and the functions common to a Simple project and Structured project such as parameter setting, operation method for the online function. (Sold separately)	SH-080779ENG (13JU63)
GX Works2 Version1 Operating Manual (Structured Project) Explains operation methods such as creating and monitoring programs in Structured project of GX Works2. (Sold separately)	SH-080781ENG (13JU65)
GX Works2 Beginner's Manual (Structured Project) Explains fundamental operation methods such as creating, editing, and monitoring programs in Structured project for users inexperienced with GX Works2. (Sold separately)	SH-080788ENG (13JZ23)

⊠POINT

The Operating Manuals are included on the CD-ROM of the software package in a PDF file format. Manuals in printed form are sold separately. Order a manual by quoting the manual number (model code) listed in the table above.

1 OVERVIEW

1

OVERVIEW

2

STRUCTURED DESIGN OF SEQUENCE PROGRAMS

3

PROCEDURE FOR CREATING PROGRAMS

4

PROGRAM CONFIGURATION

5

WRITING PROGRAMS

A

APPENDIX

INDEX

1.1 Overview

This manual describes program configurations and contents for creating sequence programs using a structured programming method, and provides basic knowledge for writing programs.

1.2 Purpose of This Manual

This manual explains programming methods, programming languages, and other information necessary for creating structured programs.

Manuals for reference are listed in the following table according to their purpose.
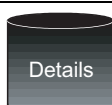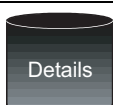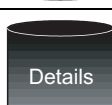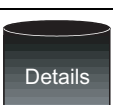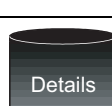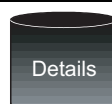
For information such as the contents and number of each manual, refer to the list of 'Related manuals'.

(1) Operation of GX Works2

Purpose		GX Works2 Installation Instructions	GX Works2 Beginner's Manual		GX Works2 Version1 Operating Manual			
		-	Simple Project	Structured Project	Common	Simple Project	Structured Project	Intelligent Function Module
Installation	Learning the operating environment and installation method	Details						
Operation of Simple project	Learning the basic operations and operating procedures		Details		Outline	Outline		
	Learning the functions and operation methods for programming				Outline	Details	Details *1	
	Learning all functions and operation methods except for programming				Details			
Operation of Structured project	Learning the basic operations and operating procedures			Details	Outline		Outline	
	Learning the functions and operation methods for programming				Outline	Details	Details	
	Learning all functions and operation methods except for programming				Details			
Operation of intelligent function module	Learning data setting methods for intelligent function module							Details

*1: ST programs only

(2) Programming (for QCPU (Q mode)/LCPU)

Purpose		MELSEC-Q/L/F Structured Programming Manual	MELSEC-Q/L Structured Programming Manual			MELSEC-Q/L Programming Manual	MELSEC-Q/L/QnA Programming Manual	User's Manual for intelligent function module/ Reference Manual for network module
		Fundamentals	Common Instructions	Special Instructions	Application Functions	Common Instructions	PID Control Instructions	–
Programming in Simple project	Learning the types and details of common instructions, descriptions of error codes, special relays, and special registers					Details		
	Learning the types and details of instructions for intelligent function modules							Details
	Learning the types and details of instructions for network modules							Details
	Learning the types and details of instructions for the PID control function						Details	
Programming in Structured project	Learning the fundamentals for creating a structured program for the first time	Details						
	Learning the types and details of the common instructions		Details					
	Learning the types and details of instructions for intelligent function modules			Details				Details
	Learning the types and details of instructions for network modules			Details				Details
	Learning the types and details of instructions for the PID control function			Details			Details	
	Learning the descriptions of error codes, special relays, and special registers					Details		
	Learning the types and details of application functions				Details			

(3) Programming (for FXCPU)

Purpose		MELSEC-Q/L/F Structured Programming Manual	FXCPU Structured Programming Manual			FXCPU Programming Manual		
		Fundamentals	Device & Common	Basic & Applied Instruction	Application Functions	FX0, FX0S, FX0N, FX1, FXU, FX2C	FX1S, FX1N, FX2N, FX1NC, FX2NC	FX3G, FX3U, FX3UC
Programming in Simple project	Learning the types and details of basic/application instructions, descriptions of devices and parameters					Details	Details	Details
Programming in Structured project	Learning the fundamentals for creating a structured program for the first time	Details						
	Learning the descriptions of devices, parameters, and error codes		Details					
	Learning the types and details of sequence instructions			Details				
	Learning the types and details of application instructions				Details			

1.2 Purpose of This Manual

1.3 Generic Terms and Abbreviations in This Manual

This manual uses the generic terms and abbreviations listed in the following table to discuss the software packages and programmable controller CPUs. Corresponding module models are also listed if needed.

Generic term and abbreviation	Description
GX Works2	Generic product name for the SWnDNC-GXW2-E (n: version)
GX Developer	Generic product name for the SWnD5C-GPPW-E, SWnD5C-GPPW-EA, SWnD5C-GPPW-EV, and SWnD5C-GPPW-EVA (n: version)
GX IEC Developer	Generic product name for the SWnD5C-MEDOC3-E (n: version)
Basic model QCPU	Generic term for the Q00J, Q00, and Q01
High Performance model QCPU	Generic term for the Q02, Q02H, Q06H, Q12H, and Q25H
Universal model QCPU	Generic term for the Q00UJ, Q00U, Q01U, Q02U, Q03UD, Q03UDE, Q04UDH, Q04UDEH, Q06UDH, Q06UDEH, Q10UDH, Q10UDEH, Q13UDH, Q13UDEH, Q20UDH, Q20UDEH, Q26UDH, and Q26UDEH
QCPU (Q mode)	Generic term for the Basic model QCPU, High Performance model QCPU, and Universal model QCPU
LCPU	Generic term for the L02 and L26-BT
FXCPU	Generic term for the MELSEC-FX series (FX0, FX0S, FX0N, FXU, FX2C, FX1S, FX1N, FX1NC, FX2N, FX2NC, FX3G, FX3U, FX3UC)
CPU module	Generic term for the QCPU (Q mode), LCPU, and FXCPU
Personal Computer	Generic term for personal computer on which Windows® operates
IEC61131-3	Abbreviation for the IEC 61131-3 international standard
Common instruction	Generic term for the sequence instructions, basic instructions, application instructions, data link instructions, multiple CPU dedicated instructions, and multiple CPU high speed transmission dedicated instructions
Special instruction	Generic term for the module dedicated instructions, PID control instructions, socket communication function instructions, built-in I/O function instructions, and data logging function instructions

1.4 Features of Structured Programs

This section explains the features of structured programs.

(1) Structured design

A structured design is a method to program control contents performed by a programmable controller CPU, which are divided into small processing units (components) to create hierarchical structures. A user can design programs knowing the component structures of sequence programs by using the structured programming.

The following are the advantages of creating hierarchical programs.

- A user can start programming by planning the outline of a program, then gradually work into detailed designs.
- Programs stated at the lowest level of a hierarchical design are extremely simple and each program has a high degree of independence.

The following are the advantages of creating structured programs.

- The process of each component is clarified, allowing a good perspective of the program.
- Programs can be divided and created by multiple programmers.
- Program reusability is increased, and it improves the efficiency in development.

(2) Multiple programming languages

Multiple programming languages are available for structured programs. A user can select the most appropriate programming language for each purpose, and combine them for creating programs.
Different programming language can be used for each program component.

Table 1.4-1 Programming languages that can be used for structured programs

Name	Description
ST (structured text)	A text language similar to C language, aimed for computer engineers.
Structured ladder	A graphic language that is expressed in form of ladder by using elements such as contacts and coils.

For outlines of the programming languages, refer to the following section.

☞ Section 4.2.6. Programming languages for POUs

For details on each programming language, refer to the following chapter.

☞ Chapter 5. WRITING PROGRAMS

The ladder/SFC languages used in the existing GX Developer and Simple projects of GX Works2 can be used.
For details on writing programs, refer to the following manuals.

☞ Programming manuals for each CPU

(3) Improved program reusability

Program components can be stored as libraries. This means program assets can be utilized to improve the reusability of programs.

1.5 Applicable CPU Modules

The following table shows the applicable CPU modules for programs in the Structured project.

Table 1.5-1 Applicable CPU modules

Programmable controller CPU type	
Basic model QCPU	Q00J, Q00, Q01
High Performance model QCPU	Q02, Q02H, Q06H, Q12H, Q25H
Universal model QCPU	Q00UJ, Q00U, Q01U, Q02U, Q03UD, Q03UDE, Q04UDH, Q04UDEH, Q06UDH, Q06UDEH, Q10UDH, Q10UDEH, Q13UDH, Q13UDEH, Q20UDH, Q20UDEH, Q26UDH, Q26UDEH
LCPU	L02, L26-BT
FXCPU	FX$_0$, FX$_{0S}$, FX$_{0N}$, FX$_U$, FX$_{2C}$, FX$_{1S}$, FX$_{1N}$, FX$_{1NC}$, FX$_{2N}$, FX$_{2NC}$, FX$_{3G}$, FX$_{3U}$, FX$_{3UC}$

1.6 Compatible Software Package

The following programming tool is used for creating, editing, and monitoring the programs in the Structured project.

Table 1.6-1 Compatible software package

Software package name	Model name
GX Works2	SW1DNC-GXW2-E

(1) What is GX Works2?

GX Works2 is a software package used for editing and debugging sequence programs, monitoring programmable controller CPUs, and other operations. It runs on a personal computer in the Microsoft$^®$ Windows$^®$ Operating System environment.

Created sequence programs are managed in units of 'projects' for each programmable controller CPU. Projects are broadly divided into 'Simple project' and 'Structured project'.

⊠POINT

This manual explains the basic programming by referring the Structured project in GX Works2.

MEMO

2

STRUCTURED DESIGN OF SEQUENCE PROGRAMS

1

OVERVIEW

2

STRUCTURED DESIGN OF
SEQUENCE PROGRAMS

3

PROCEDURE FOR
CREATING PROGRAMS

4

PROGRAM
CONFIGURATION

5

WRITING
PROGRAMS

A

APPENDIX

I

INDEX

2.1 What is a Hierarchical Sequence Program?

The hierarchy is to create a sequence program by dividing control functions performed in a programmable controller CPU into a number of levels.

In higher levels, the processing order and timing in a fixed range is controlled.

With each move from a higher level to a lower level, control contents and processes are progressively subdivided within a fixed range, and specific processes are described in lower levels.

In the Structured project, hierarchical sequence programs are created with the configuration that states the highest level as the project, followed by program files, tasks, and POUs (abbreviation for Program Organization Units).
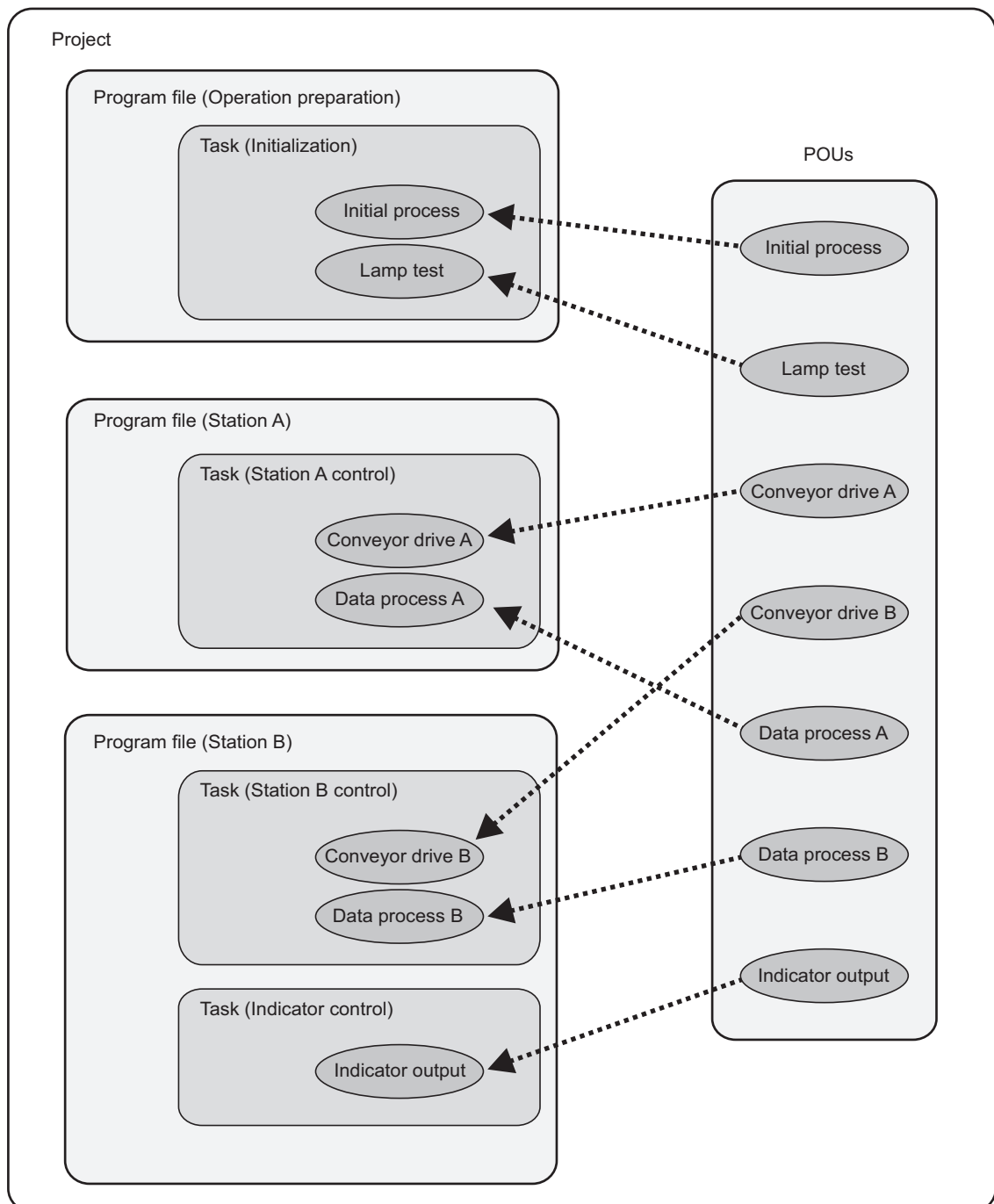
2.2 What is a Structured Sequence Program?

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

In a structured program design, segmenting processes in lower levels as much as possible is recommended.

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

The following shows examples of the process that would be ideal to be structured.

- A process that is used repeatedly in a sequence program.
- A process that can be divided into components.

MEMO

3 PROCEDURE FOR CREATING PROGRAMS

1

OVERVIEW

2

STRUCTURED DESIGN OF
SEQUENCE PROGRAMS

3

PROCEDURE FOR
CREATING PROGRAMS

4

PROGRAM
CONFIGURATION

5

WRITING
PROGRAMS

A

APPENDIX

INDEX
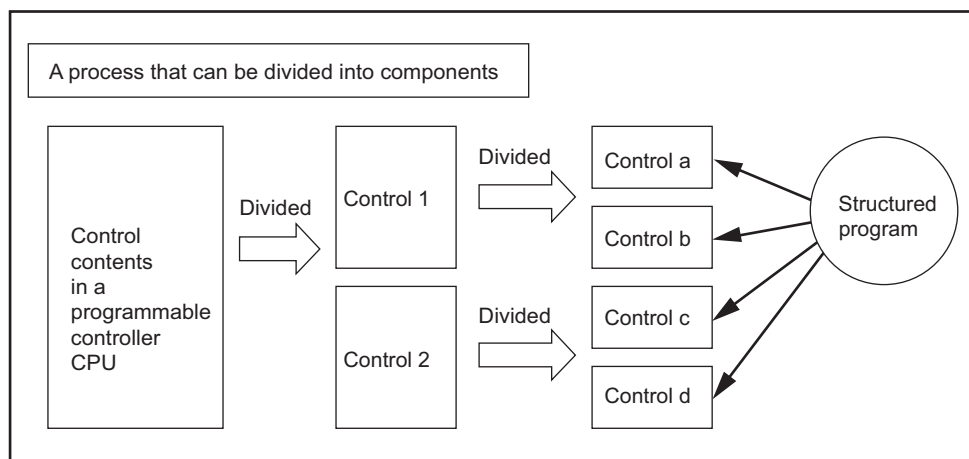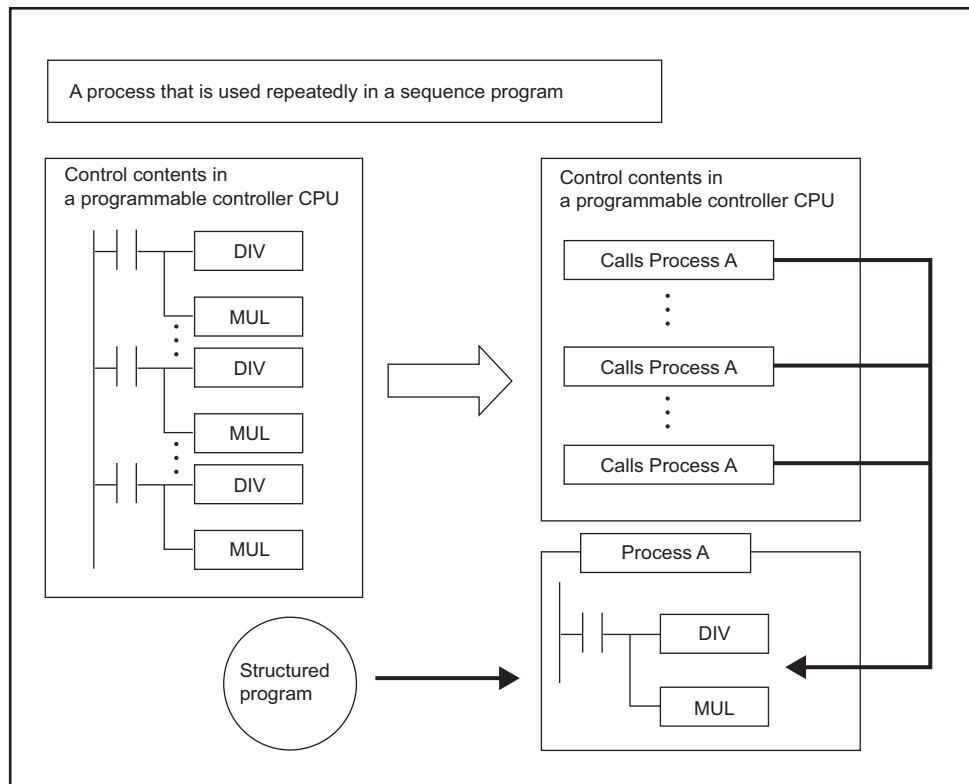
3.1 Procedure for Creating Sequence Programs in Structured Project

This section explains the basic procedure for creating a sequence program in the Structured project.

(1) Creating a program structure

Procedure
Create program files.
Create tasks.

⬇

(2) Creating POUs

Procedure
Create POUs.
Define global labels.
Define local labels.

⬇

(3) Editing the programs

Procedure
Edit the programs of each POU.

⬇

(4) Compiling the programs

Procedure
Register the POUs in the tasks.
Compile the programs.

4 PROGRAM CONFIGURATION

1
OVERVIEW

2
STRUCTURED DESIGN OF SEQUENCE PROGRAMS

3
PROCEDURE FOR CREATING PROGRAMS

4
PROGRAM CONFIGURATION

5
WRITING PROGRAMS

A
APPENDIX

I
INDEX
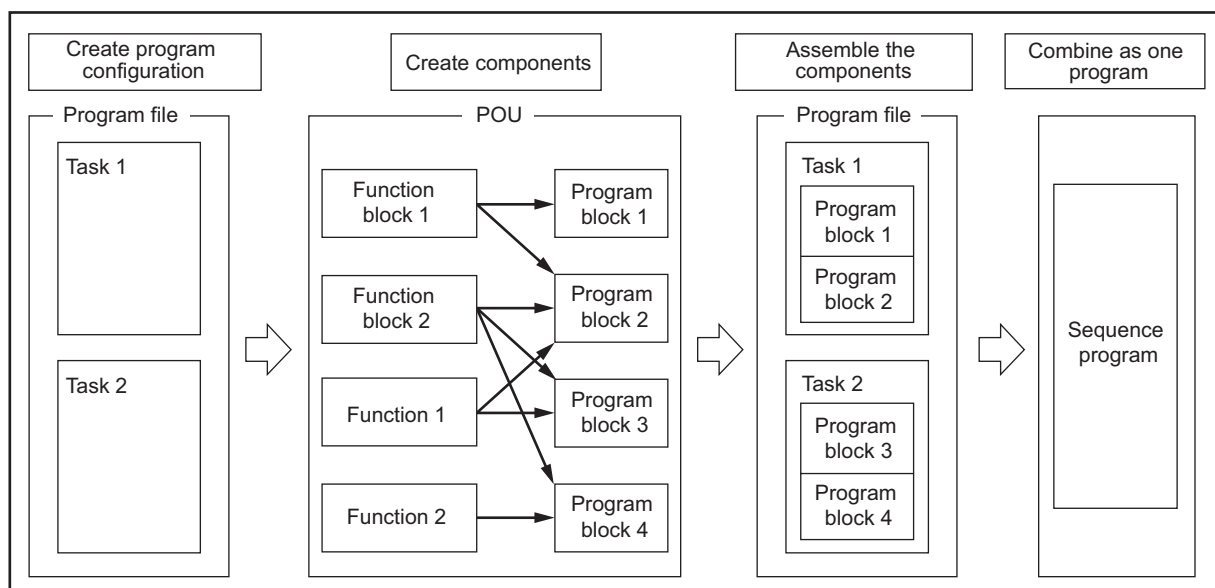
4.1 Overview of Program Configuration

A sequence program created in the Structured project is composed of program files, tasks, and POUs.

For details of program components, refer to the following sections.

For projects: ☞ Section 4.1.1 Project

For program files: ☞ Section 4.1.2 Program files

For tasks: ☞ Section 4.1.3 Tasks

For POUs: ☞ Section 4.2 POUs

The following figure shows the configuration of program files, tasks, and POUs in the project.
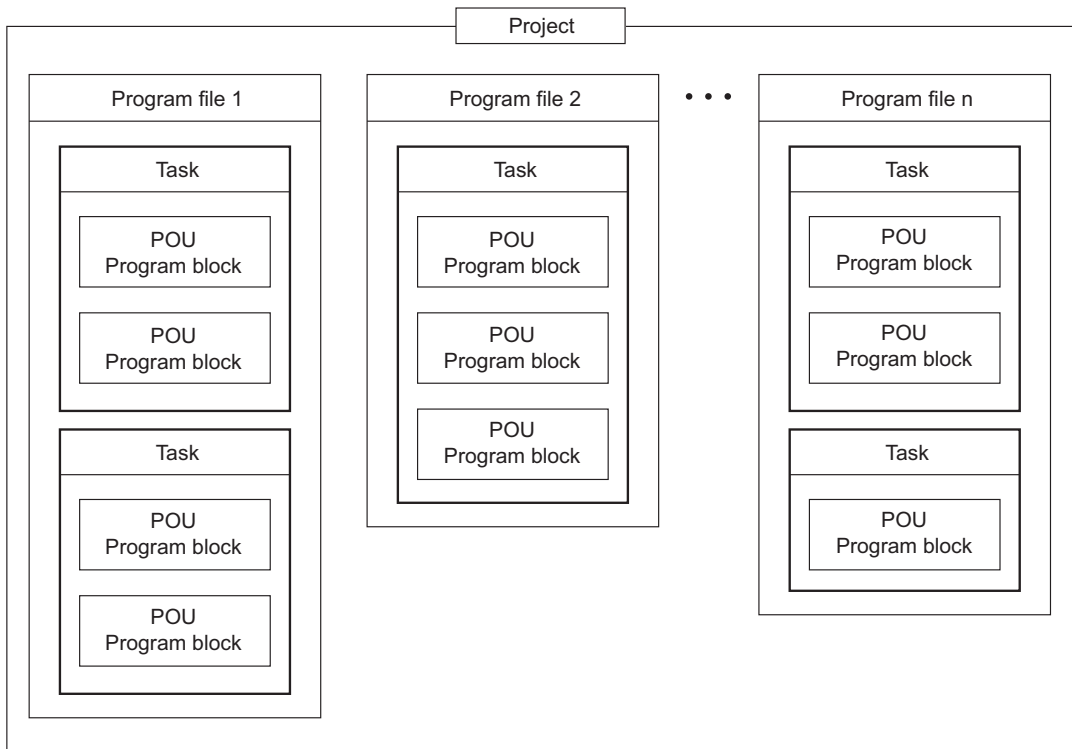
4.1.1 Project

A project is a generic term for data (such as programs and parameters) to be executed in a programmable controller CPU.

One or more program files need to be created in a project.

4.1.2 Program files

One or more tasks need to be created in a program file. (Created tasks are executed under the control of the program file.)

The execution types (such as scan execution and fixed scan execution) for executing program files in a programmable controller CPU are set in the program setting of the parameter.

For details on the execution types set in the parameter, refer to the user's manual of each CPU module.
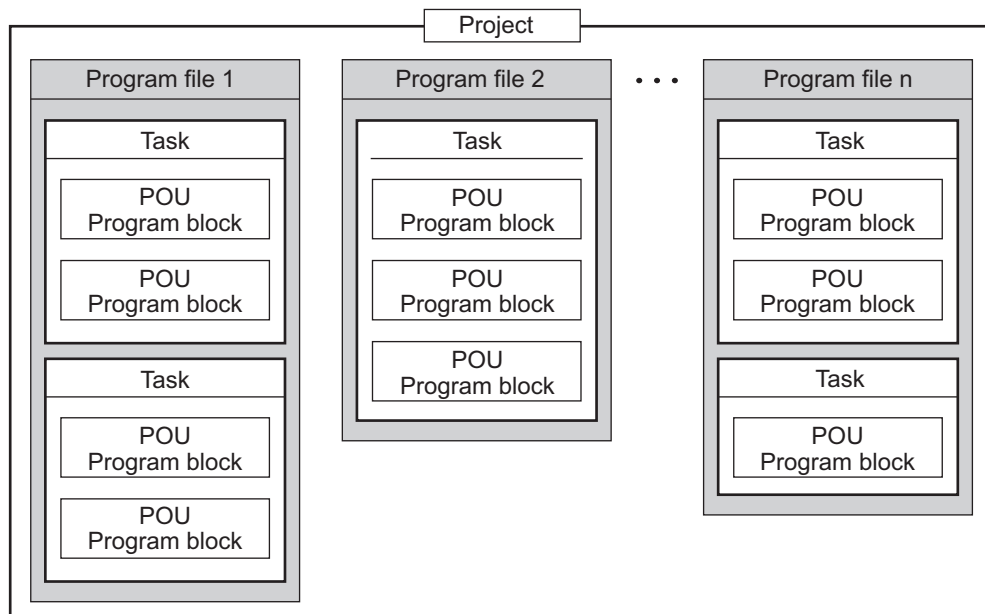
4.1.3 Tasks

A task is an element that contains multiple POUs, and it is registered to a program file.

One or more program blocks of POU need to be registered in a task. (Functions and function blocks cannot be registered in a task.)
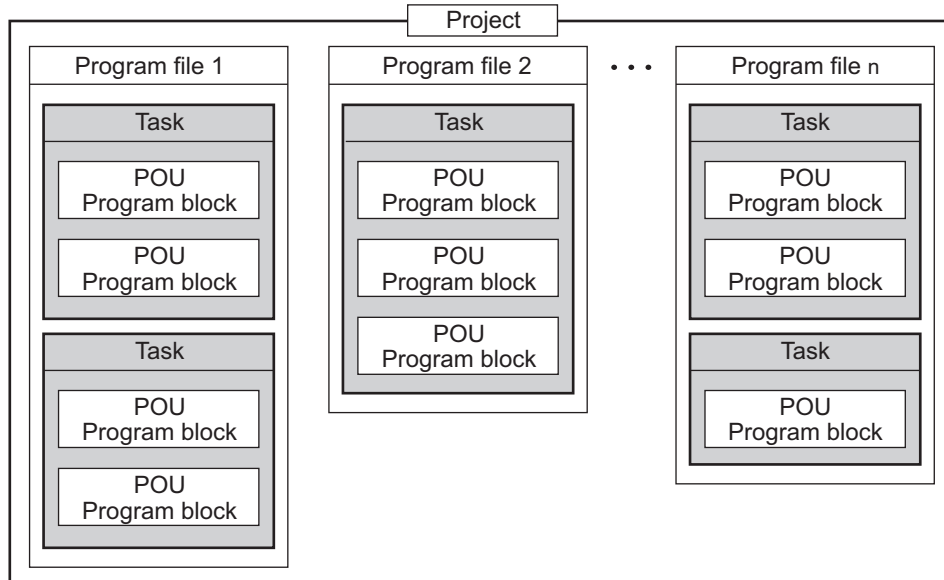
(1) Task executing condition

The executing conditions in a programmable controller CPU are set for each task that is registered to program files. Executing processes are determined for each task by setting the executing condition.

The following are the types of task executing condition.

(a) Scan execution (Default executing condition)
Executes registered program blocks for each scan.

(b) Event execution
Executes tasks when values are set to the corresponding devices or labels.

(c) Fixed scan execution
Executes tasks in a specified cycle.

A priority can be set for each task execution.

● Priority
When executing conditions of multiple tasks are met simultaneously, the tasks are executed according to the set priority.
Tasks are executed in the order from the smallest priority level number.
Tasks set with a same priority level number are executed in the order of task data name.

4.2 POUs

A POU (abbreviation for Program Organization Unit) is a program component defined by each function.

4.2.1 Types of POU

The following three types can be selected for each POU according to the contents to be defined.
- Program block
- Function
- Function block

Each POU consists of local labels[1] and a program.

A process can be described in a programming language that suits the control function for each POU.
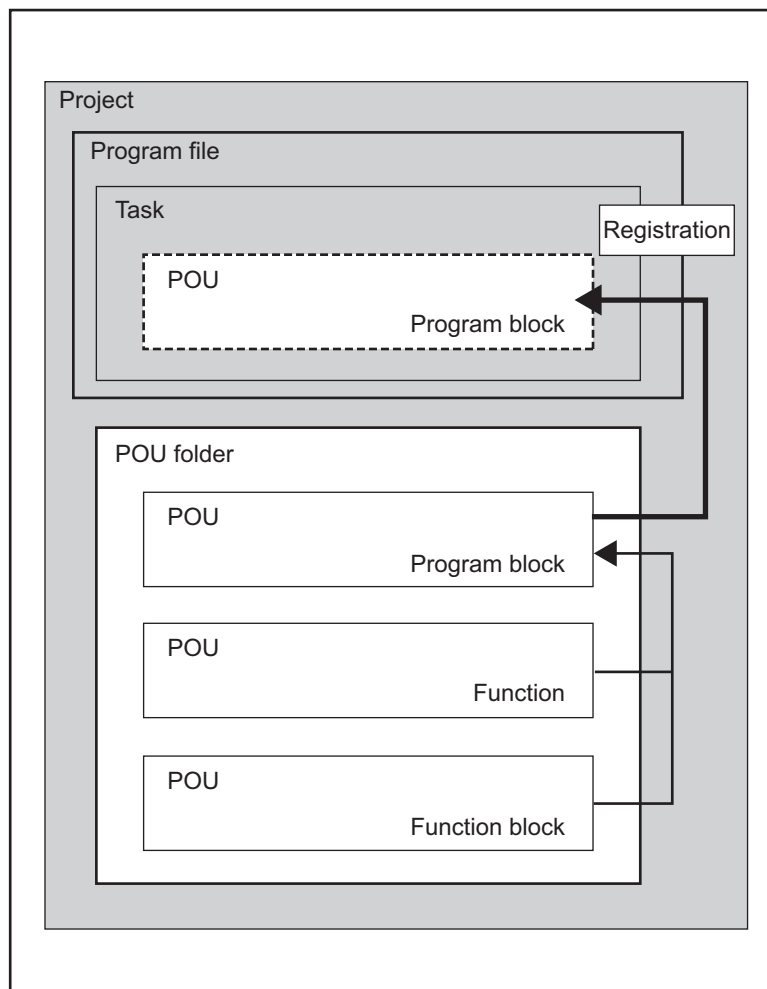
*1: Local labels are labels that can be used only in programs of declared POUs. For details of local labels, refer to the following section.
☞ Section 4.3.2 Local labels

4.2.2 Program blocks

A program block is an element that is stated at the highest level of POU. Libraries, functions, and function blocks are used to edit program blocks.

```
┌─────────────────┐         ┌─────────────────┐
│  Program block  │─────┬───│     Library     │
└─────────────────┘     │   └─────────────────┘
                        │
                        │   ┌─────────────────┐
                        ├───│    Function     │
                        │   └─────────────────┘
                        │
                        │   ┌─────────────────┐
                        └───│  Function block │
                            └─────────────────┘
```
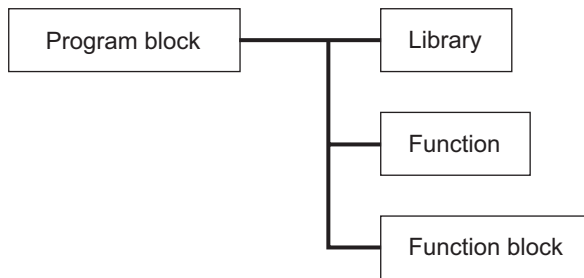
Sequence programs executed in a programmable controller CPU are created by program blocks of POU.

For a simplest sequence program, only one program block needs to be created and registered to a task in order to be executed in a programmable controller CPU.

Program blocks can be described in the ST or structured ladder language.

4.2.3 Functions

Libraries and functions are used to edit functions.

Functions can be used by calling them from program blocks, function blocks or functions.

```
┌─────────────────┐         ┌─────────────────┐
│    Function     │─────┬───│     Library     │
└─────────────────┘     │   └─────────────────┘
                        │
                        │   ┌─────────────────┐
                        └───│    Function     │
                            └─────────────────┘
```
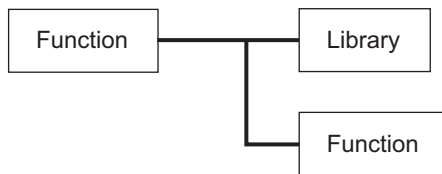
Functions always output same processing results for same input values.

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

Functions can be described in the ST or structured ladder language.

4.2.4 Function blocks

Libraries, functions, and other function blocks are used to edit function blocks.

Function blocks can be used by calling them from program blocks or function blocks. Note that they cannot be called from functions.
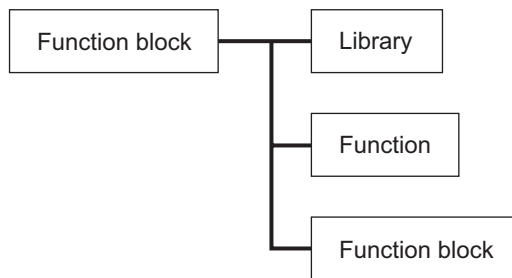
Function blocks can retain the input status since they can store values in internal and output variables. Since they use retained values for the next processing, they do not always output the same results even with the same input values.

Function blocks can be described in the ST or structured ladder language.

● Instantiation
Function blocks need to be instantiated to be used in program blocks.
For details of instantiation, refer to the following section.

☞ Section 4.2.7 Functions and function blocks

⊠**POINT**

Instances are variables representing devices assigned to labels of function blocks.
Devices are automatically assigned when instances are created with local labels.

4.2.5 Ladder blocks

In the structured ladder language, a program is divided into units of ladder blocks.

In the ST language, ladder blocks are not used.

● Ladder block labels
   A ladder block label can be set to a ladder block. A ladder block label is used to indicate a jump target for the Jump instruction.

| Ladder block label | Jump instruction | Ladder block |
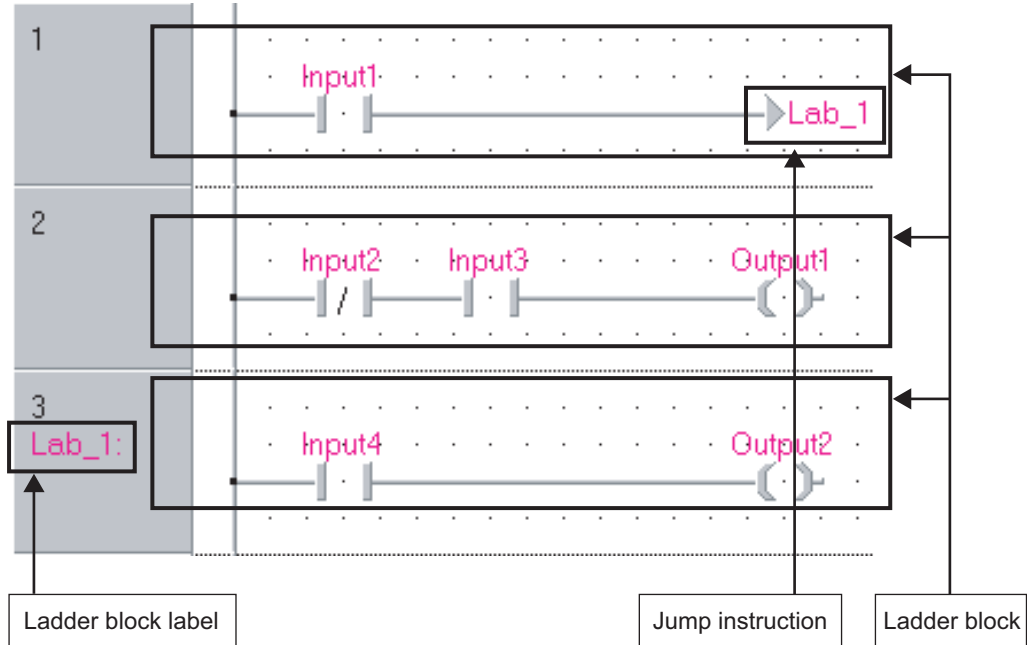
4.2.6 Programming languages for POUs

Two types of programming language are available for programs of POU.

The following explains the features of each programming language.

(1) ST: Structured text

Control syntaxes such as selection branch by conditional syntaxes or repetitions by iterative syntaxes can be described in the ST language, as in the high-level language such as C language. Clear and simple programs can be written by using these syntaxes.

```
intV2 := ABS( intV1);

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

Output_ENO := ENEG(btn01, Input1);
```

(2) Structured ladder: (ladder diagram)

The structured ladder language is a graphic language developed based on the relay ladder programming technique. Since it can be understood intuitively, it is commonly used for the sequence programming.
Ladders always start from the base line on the left.
A program written in the structured ladder language is composed of contacts, coils, function blocks, and functions. These elements are connected by vertical and horizontal lines.
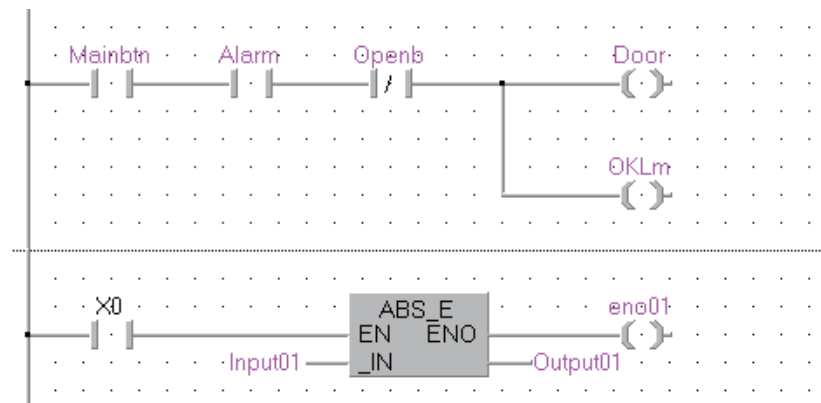
4.2.7 Functions and function blocks

The following table shows differences between functions and function blocks.

Table 4.2.7-1 Differences between functions and function blocks

Item	Function	Function block
Output variable assignment	Can not be assigned	Can be assigned
Internal variable	Not used	Used
Creating instances	Not necessary	Necessary

(1) Output variable assignment

A function always outputs a single operation result. A function that does not output any operation result or outputs multiple operation results cannot be created.
A function block can output multiple operation results. It also can be created without any output.
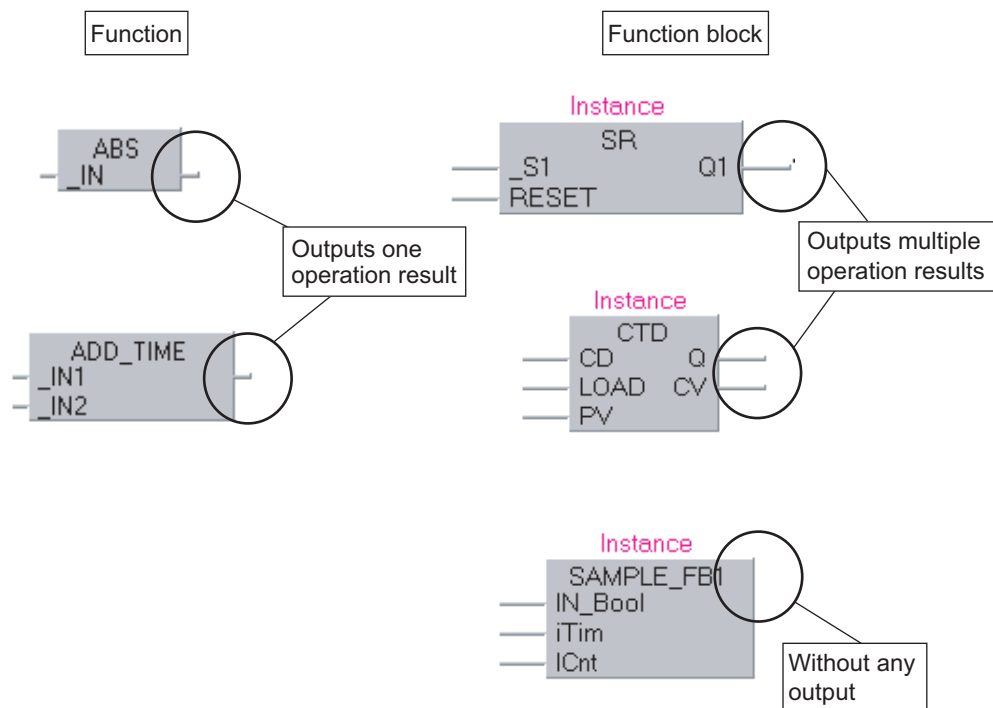
(2) Internal variables

A function does not use internal variables. It uses devices assigned directly to each input variable and repeats operations.

(a) A program that outputs the total of three input variables (When using a function (FUN1))

```
                     ┌─────────────────────────────────┐
                     │           Function              │
                     │                                 │
   X0  ──────────────┤                                 │
                     │   ┌───────┐                      │
   D109 ─────────────┤   │ D109  │                      │
                     │   ├───────┤       ┌──────┐        │
                     │   │   +   │  ───▶  │ FUN1 ├─── D120
   D110 ─────────────┤   │ D110  │       └──────┘        │
                     │   ├───────┤                      │
                     │   │   +   │                      │
   D111 ─────────────┤   │ D111  │                      │
                     │   └───────┘                      │
                     └─────────────────────────────────┘
```
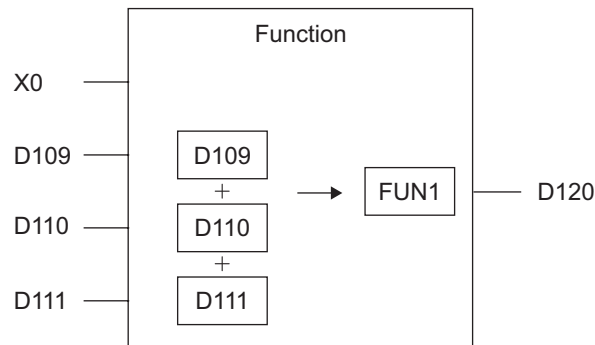
A function block uses internal variables. Different devices are assigned to the internal variables for each instance of function blocks.

(b) Programs that output the total of three input variables (When using function blocks)

```
            Instance A                              Instance B
   ┌──────────────────────────┐        ┌──────────────────────────┐
   │      Function block       │        │      Function block       │
   │   ┌────────┐              │        │   ┌────────┐              │
 D10┤  │ D6200  │              │      D10┤  │ D6210  │              │
   │   ├────────┤  ┌────────┐   │        │   ├────────┤  ┌────────┐   │
   │   │   +    │─▶│ D6203  ├─D13│        │   │   +    │─▶│ D6213  ├─D13│
 D11┤  │ D6201  │  └────────┘   │      D11┤  │ D6211  │  └────────┘   │
   │   ├────────┤              │        │   ├────────┤              │
   │   │   +    │              │        │   │   +    │              │
 D12┤  │ D6202  │              │      D12┤  │ D6212  │              │
   │   └────────┘              │        │   └────────┘              │
   └──────────────────────────┘        └──────────────────────────┘
```
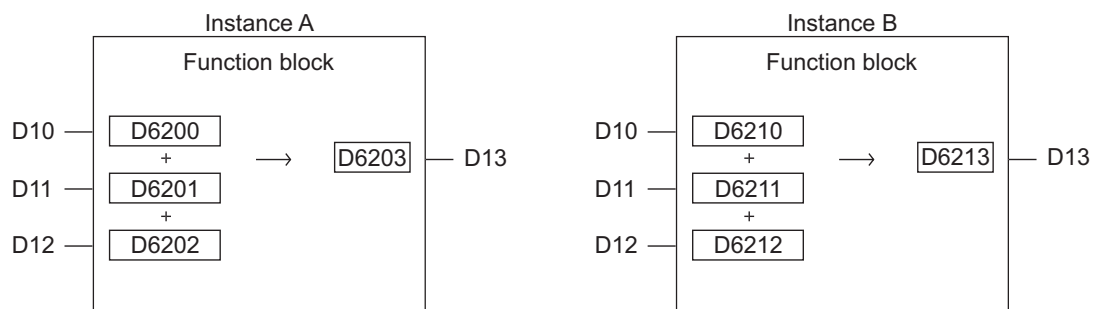
(3) Creating instances

When using function blocks, create instances to reserve internal variables.
Variables can be called from program blocks and other function blocks by creating instances for function blocks.

To create an instance, declare as a label in a global label or local label of POU that uses function blocks. Same function blocks can be instantiated with different names in a single POU.
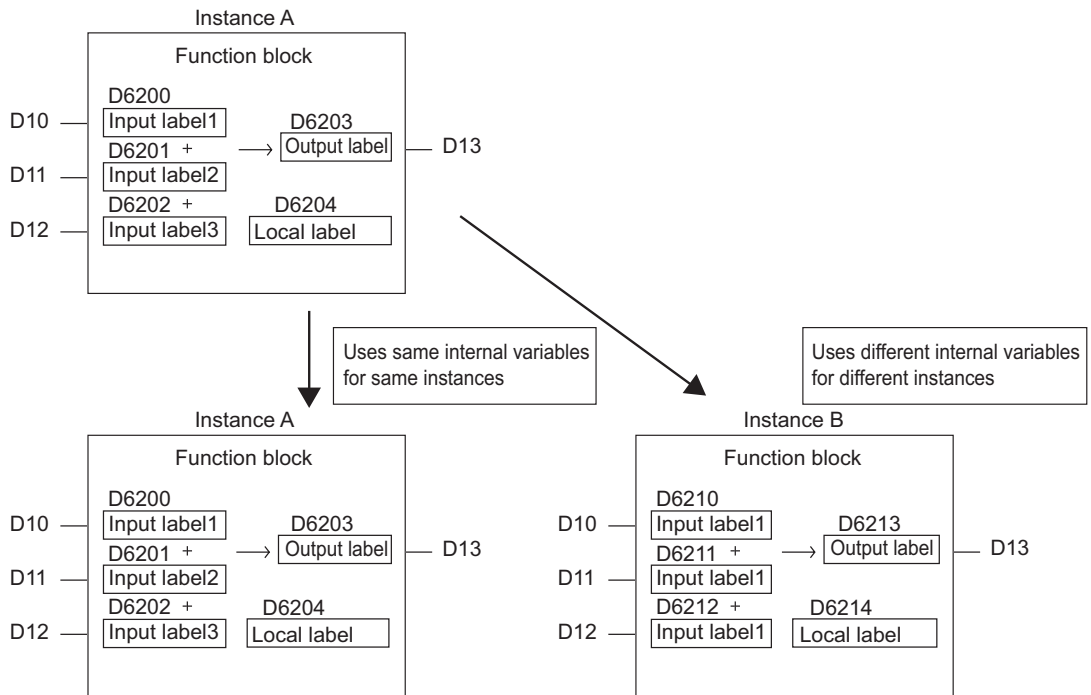
Function blocks perform operations using internal variables assigned to each instance.

4.2.8 EN and ENO

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

A Boolean variable used as an executing condition of a function is set to an EN.

A function with an EN is executed only when the executing condition of the EN is TRUE.

A Boolean variable used as an output of function executing status is set to an ENO.

The following table shows the status of ENO and the operation result according to the status of EN.

Table 4.2.8-1 Status of ENO and the operation result according to the status of EN

EN	ENO	Operation result
TRUE (Operation execution)	TRUE (No operation error)	Operation output value
	FALSE (Operation error)	Undefined value
FALSE (Operation stop)	FALSE	Undefined value

⊠**POINT**

1. A setting of an output label to an ENO is not essential.
2. As for application functions, functions with an EN are shown as 'Function name_E'.

● Usage example of EN and ENO
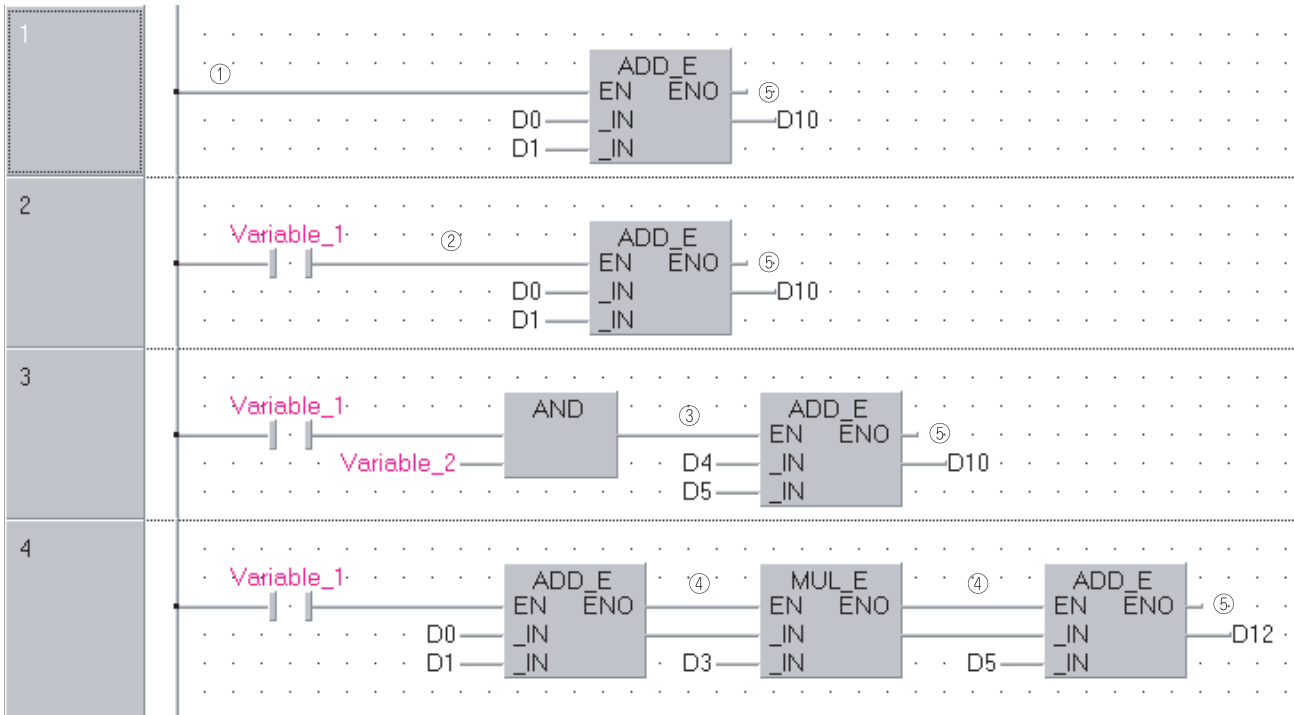
No	Control description
①	When the EN input is directly connected from the left base line, the EN input is always TRUE and the instruction is always executed. If the ADD_E instruction is used in this manner, the operation result is the same as the ADD instruction without the EN input.
②	When Variable_1 is connected to the EN input, the instruction is executed when Variable_1 is TRUE.
③	When the result of Boolean operation is connected to the EN input, the instruction is executed when the result of Boolean operation is TRUE.
④	When the ENO outputs are connected to the EN inputs, three instructions are executed when Variable_1 is TRUE.
⑤	When the ENO outputs are not connected, the execution status of the instruction is not output.

4.2 POUs
4.2.8 EN and ENO

4.3 Labels

Labels include global labels and local labels.

4.3.1 Global labels

The global labels are labels that can be used in program blocks and function blocks.

In the setting of a global label, a label name, a class, a data type, and a device are associated with each other.

4.3.2 Local labels

The local labels are labels that can be used only in declared POUs. They are individually defined per POU.

In the setting of a local label, a label name, a class, and a data type are set.

For the local labels, the user does not need to specify devices. Devices are assigned automatically at compilation.

4.3.3 Label classes

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

The following table shows label classes.

Table 4.3.3-1 Label classes

Class	Description	Applicable POU		
		Program block	Function	Function block
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	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 constant cannot be used in other POUs.	○	○	○
VAR_RETAIN[1]	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 receives a value This label cannot be changed in a POU.	×	○	○
VAR_OUTPUT	Label that outputs a value from a function block	×	×	○
VAR_IN_OUT	Local label that receives a value and outputs the value from a POU This label can be changed in a POU.	×	×	○

*1: Not supported with the FXCPU.

⊠POINT

- Input variables, output variables, and input/output variables
  VAR_INPUT is an input variable for functions and function blocks, and VAR_OUTPUT is an output variable for function blocks.
  VAR_IN_OUT can be used for both input and output variables.
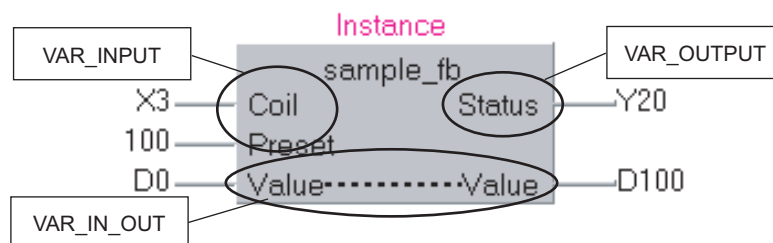
4.3.4 Setting labels

Labels used in a program require setting of either global label or local label.

The following describes setting examples of the arguments Var_D10 and Var_D20 of the DMOV instruction.

```
    X0                ┌────────────┐
 ───┤ ├──────────────┤   DMOV     │
                      │ EN     ENO ├───
         Var_D10 ─────┤ s       d  ├─── Var_D20
                      └────────────┘
```
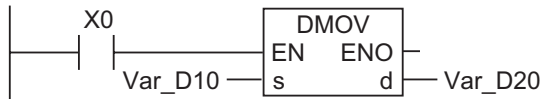
● Using the arguments of the DMOV instruction as global labels

Set the Class, Label Name, Data Type, Device, and Address.

	Class		Label Name	Data Type		Constant	Device	Address	Comment	Remark
1	VAR_GLOBAL	▼	g_bool1	Bit	...		M0	%MX0.0		
2	VAR_GLOBAL	▼	g_int1	Word[Signed]	...		D0	%MW0.0		
3		▼			...					

● Using the arguments of the DMOV instruction as local labels

Set the Class, Label Name, and Data Type.

	Class		Label Name	Data Type		Constant	Device	Address	Comment
1	VAR	▼	g_bool1	Bit	...				
2	VAR	▼	g_int1	Word[Signed]	...				
3		▼			...				

4.3.5 Data types

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

(1) Elementary data types

The following data types are available as the elementary data type.[1]

- Boolean type (bit): Represents the alternative status, such as ON or OFF.
- Bit string type (word (unsigned)/16-bit string, double word (unsigned)/32-bit string): Represents bit arrays.
- Integer type (word (signed), double word (signed)): Handles positive and negative integer values.
- Real type (single-precision real, double-precision real): Handles floating-point values.
- String type (character string): Handles character strings.
- Time type (time): Handles numeric values as day, hour, minute, and second (in millisecond).

Table 4.3.5-1 Elementary data types

Elementary data type	Description	Value range	Bit length
Bit	Bool	0 (FALSE), 1 (TRUE)	1 bit
Word (signed)	Integer	-32768 to 32767	16 bits
Double word (signed)	Double-precision integer	-2147483648 to 2147483647	32 bits
Word (unsigned)/16-bit string	16-bit string	0 to 65535	16 bits
Double word (unsigned)/32-bit string	32-bit string	0 to 4294967295	32 bits
Single-precision real	Real	-2^{128} to -2^{-126}, 0, 2^{-126} to 2^{128}	32 bits
Double-precision real[2]	Double-precision real	-2^{1024} to -2^{-1022}, 0, 2^{-1022} to 2^{1024}	64 bits
String	Character string	Maximum 255 characters	Variable
Time[3]	Time value	T#-24d20h31m23s648ms to T#24d20h31m23s647ms	32 bits

*1: The following data types cannot be used for the structured ladder and ST languages.
  They can be only used for the ladder language.
  - Timer data type: Handles programmable controller CPU timer devices (T).
  - Retentive timer data type: Handles programmable controller CPU retentive timer devices (ST).
  - Counter data type: Handles programmable controller CPU counter devices (C).
  - Pointer data type: Handles programmable controller CPU pointer devices (P).
*2: Can be used for the Universal model QCPU/LCPU only.
*3: The time type is used in time type operation instructions of application function.
  For details of the application functions, refer to the following manuals.

  ⏩ MELSEC-Q/L Structured Programming Manual (Application Functions)

  ⏩ FXCPU Structured Programming Manual (Application Functions)

(2) Generic data types

Generic data type is the data type of labels summarizing some elementary data types. Data type name starts with 'ANY'.

ANY data types are used when multiple data types are allowed for function arguments and return values.
Labels defined in generic data types can be used in any sub-level data type.
For example, if the argument of a function is ANY_NUM data type, desired data type for an argument can be specified from word (signed) type, double word (signed) type, single-precision real type, and double-precision real type.

Arguments of functions and instructions are described using generic data types, in order to be used for various different data types.

The following figure shows the types of generic data type and their corresponding elementary data types.

ANY

ANY_SIMPLE		Array[*1]	Structure[*2]

ANY_NUM	ANY_BIT	Time	String

ANY_BIT:
- Bit
- Word (unsigned)/ 16-bit string
- Double word (unsigned)/ 32-bit string

ANY_REAL	ANY_INT
Single-precision real	Word (signed)
Double-precision real	Double word (signed)

The higher 'ANY' data types include sub-level data types.
The highest 'ANY' data type includes all data types.

ANY16

- Word (unsigned)/ 16-bit string
- Word (signed)

ANY32

- Double word (unsigned)/ 32-bit string
- Double word (signed)

*1: For arrays, refer to the following section. ☞ Section 4.4.7 Arrays
*2: For structures, refer to the following section. ☞ Section 4.4.8 Structures

4.3.6 Expressing methods of constants

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

Table 4.3.6-1 Constant expressing method

Constant type	Expressing method	Example
Bool	Input FALSE or TRUE, or input 0 or 1.	TRUE, FALSE
Binary	Append '2#' in front of a binary number.	2#0010, 2#01101010
Octal	Append '8#' in front of an octal number.	8#0, 8#337
Decimal	Directly input a decimal number, or append 'K' in front of a decimal number.	123, K123
Hexadecimal	Append '16#' or 'H' in front of a hexadecimal number. When a lowercase letter 'h' is appended, it is converted to uppercase automatically.	16#FF, HFF
Real number	Directly input a real number or append 'E' in front of a real number.	2.34, E2.34
Character string	Enclose a character string with single quotations (') or double quotations (").	'ABC', "ABC"
Time	Append 'T#' in front.	T#1h, T#1d2h3m4s5ms

4.3 Labels
4.3.6 Expressing methods of constants

4.4 Method for Specifying Data

The following shows the six types of data that can be used for instructions in CPU modules.

Data that can be handled by CPU module

— Bit data .... Section 4.4.1

— Numeric data — Integer data — Word (Signed) data .... Section 4.4.2

Double word (Signed) data .. Section 4.4.3

Real number data — Single-precision real data .... Section 4.4.4 (1)

Double-precision real data .... Section 4.4.4 (2)

— Character string data .... Section 4.4.5

— Time data ....Section 4.4.6

4

PROGRAM
CONFIGURATION
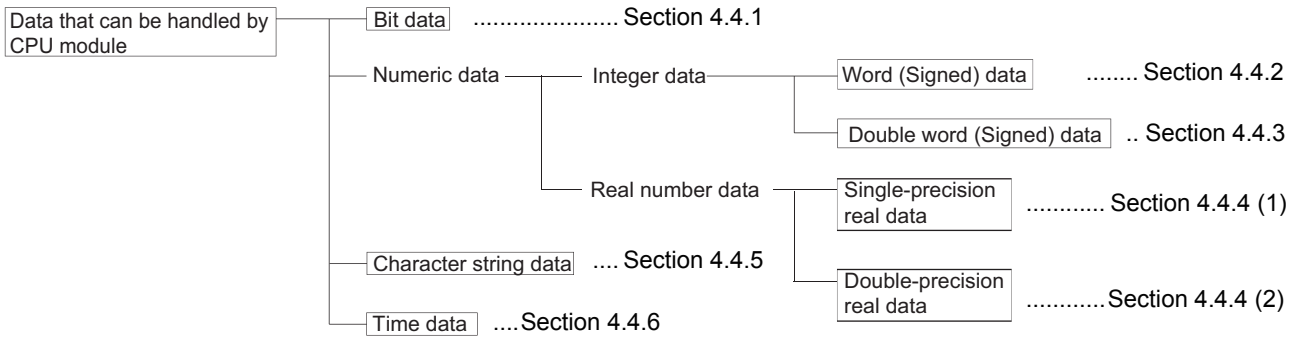
4.4.1 Bit data

Bit data are data handled in units of 1 bit, such as contacts and coils.

'Bit devices' and 'bit specification for word devices' can be used as bit data.

(1) Using bit devices
A bit device is specified in unit of one point.

One point of M0 is the target bit device

One point of Y10 is the target bit device
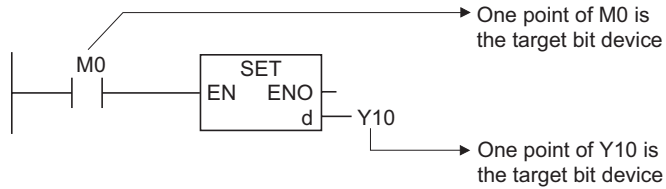
(2) Using word devices

    (a) By specifying a bit number for a word device, 1/0 of the specified bit number can be used as bit data.

Each bit of a word device can be used (1=ON, 0=OFF)
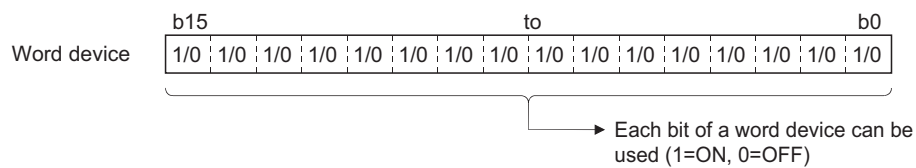
    (b) Specify a bit for word device as ' Word device . Bit number '. (Bit number is specified in hexadecimal.)
For example, bit 5 (b5) of D0 is specified as D0.5 and bit 10 (b10) of D0 is specified as D0.A. Note that bit specifications are not applicable for timers (T), retentive timers (ST), counters (C), and index registers (Z). (Example: Z0.0 is not available).

Bit specification for word devices (Turns ON bit 5 (b5) of D0)

Bit specification for word devices (Turns ON/OFF according to the status (1/0) of bit 5 (b5) of D0)
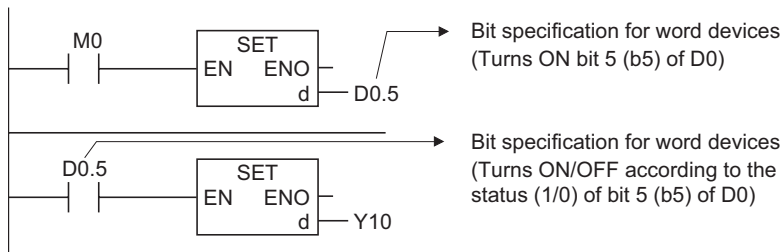
☒ POINT

For FXCPU, bit specification for word devices can be used for FX3U and FX3UC.

4.4.2 Word (16 bits) data

Word data are 16-bit numeric value data used in basic instructions and application instructions.

The following shows the two types of word data that can be handled in CPU modules.

- Decimal constants .... -32768 to 32767
- Hexadecimal constants .... 0000H to FFFFH

For word data, word devices and digit specification for bit devices can be used.

Note that word data cannot be specified using digit specification for direct access inputs (DX) and direct access outputs (DY). (For direct access inputs and direct access outputs, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used.)

(1) Using bit devices

(a) By specifying a digit for bit devices, word data can be used.

Specify a digit for bit data as ' Number of digits   Start number of bit device '. A digit can be specified in the range from K1 to K4 in unit of 4 points (4 bits). (For a link direct device,

specify as 'J Network No. \ Number of digits   Start number of bit device '. To specify X100 to X10F of Network No.2, specify as J2\K4X100.)
The following are the examples of the target points when a digit is specified for X0.

- K1X0 .... 4 points of X0 to X3
- K2X0 .... 8 points of X0 to X7
- K3X0 .... 12 points of X0 to XB
- K4X0 .... 16 points of X0 to XF
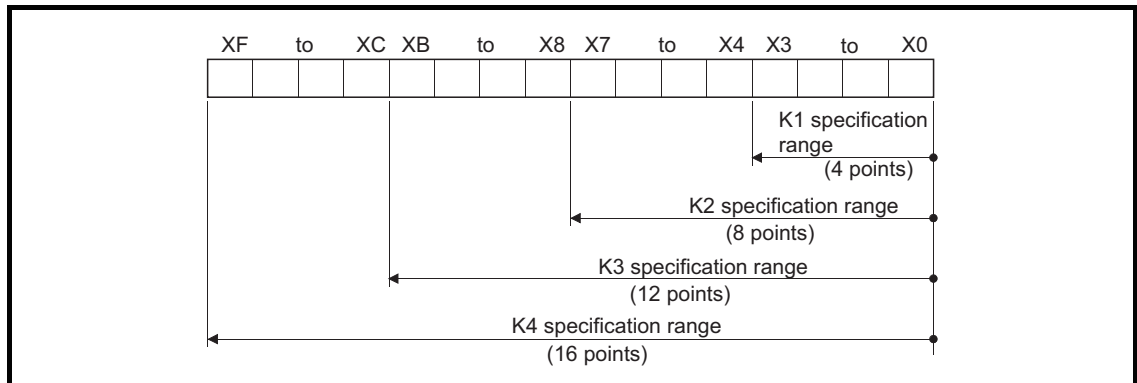
Figure 4.4.2-1 Digit specification setting range of word data (16 bits)

(b) The following table shows the numeric values that can be used as source data when a digit is specified at the source s .

Table 4.4.2-2 List of digit specification and numeric values that can be used

Number of specified digits	Value range
K1 (4 points)	0 to 15
K2 (8 points)	0 to 255
K3 (12 points)	0 to 4095
K4 (16 points)	−32768 to 32767

(c) When the destination is a word device

For the word device at the destination side, 0s are stored to bits starting from the digit specified at the source.

Ladder example	Processing
Instruction that processes 16-bit data X010 —[ ]— MOV EN ENO K1X0 — s d — D0 Source ⓢ	K1X0 |X3|X2|X1|X0| 0s are stored ⇩ b15 ------------------- b4 b3 b2 b1 b0 D0 |0|0|0|0|0|0|0|0|0|0|0|0|X3|X2|X1|X0|
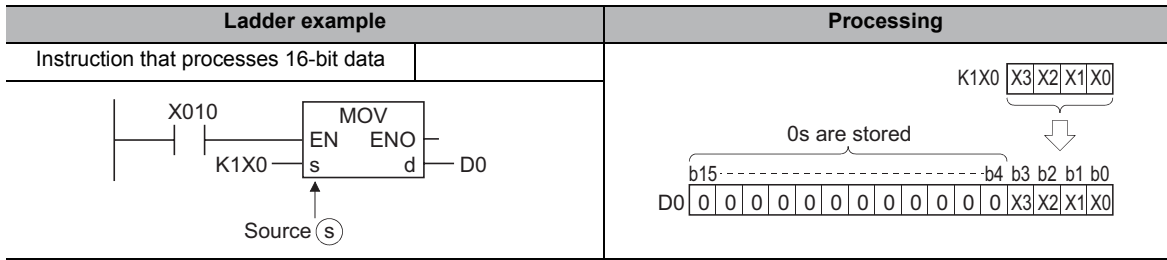
Figure 4.4.2-2 Ladder example and processing details

(d) When a digit is specified at the destination ⓓ, the points by the specified digit are the target of destination.

Bit devices following the points by the specified digit are not changed.

Ladder example	Processing
When the source ⓢ is a numeric value X010 —[ ]— MOV EN ENO H1234 — s d — K2M0 Destination ⓓ	1 2 3 4 H1234 |0|0|0|1|0|0|1|0|0|0|1|1|0|1|0|0| ⇩ M15 ------------- M8 M7 ------------- M0 K2M0 | | | | | | | |0|0|1|1|0|1|0|0| Not changed 3 4
When the source ⓢ is a word device X010 —[ ]— MOV EN ENO D0 — s d — K2M100 Destination ⓓ	b15 ------------- b8 b7 ------------- b0 D0 |1|1|1|0|1|0|1|0|1|0|0|1|1|1|0|1| ⇩ M115 ---------- M108 M107 ---------- M100 K2M100 | | | | | | | |1|0|0|1|1|1|0|1| Not changed
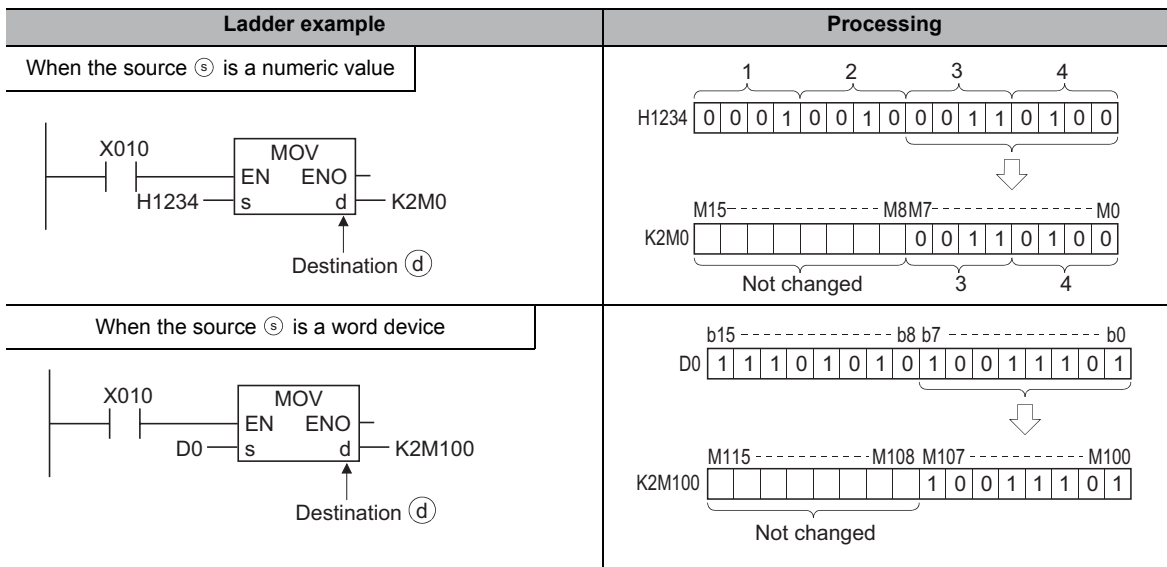
Figure 4.4.2-3 Ladder example and processing details

(2) Using word devices

A word device is specified in unit of one point (16 bits).
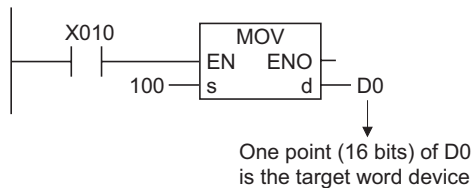
X010 —[ ]— MOV
EN ENO
100 — s d — D0
↓
One point (16 bits) of D0
is the target word device

⊠ **POINT**

1. When performing the process with a digit specification, a desired value can be used for the start device number of bit devices.
2. Digits cannot be specified for direct access inputs/outputs (DX, DY).

4.4.3 Double word (32 bits) data

Double word data are 32-bit numeric value data used in basic instructions and application instructions.

The following shows the two types of double word data that can be handled in CPU modules.
- Decimal constants .... -2147483648 to 2147483647
- Hexadecimal constants .... 00000000H to FFFFFFFFH

For double word data, word devices and digit specification for bit devices can be used.

Note that double word data cannot be specified using digit specification for direct access inputs (DX) and direct access outputs (DY).

(1) Using bit devices

    (a) By specifying a digit for bit devices, double word data can be used.

        Specify a digit for bit data as ' [Number of digits] [Start number of bit device] '. (For a link direct device, specify as 'J [Network No.] \ [Number of digits] [Start number of bit device] '. To specify X100 to X11F of Network No.2, specify as J2\K8X100.)
A digit cannot be specified in the range from K1 to K8 in unit of 4 points (4 bits).
The following are the examples of the target points when a digit is specified for X0.

- K1X0 ...... 4 points of X0 to X3
- K2X0 ...... 8 points of X0 to X7
- K3X0 ...... 12 points of X0 to XB
- K4X0 ...... 16 points of X0 to XF
- K5X0 .... 20 points of X0 to X13
- K6X0 .... 24 points of X0 to X17
- K7X0 .... 28 points of X0 to X1B
- K8X0 .... 32 points of X0 to X1F
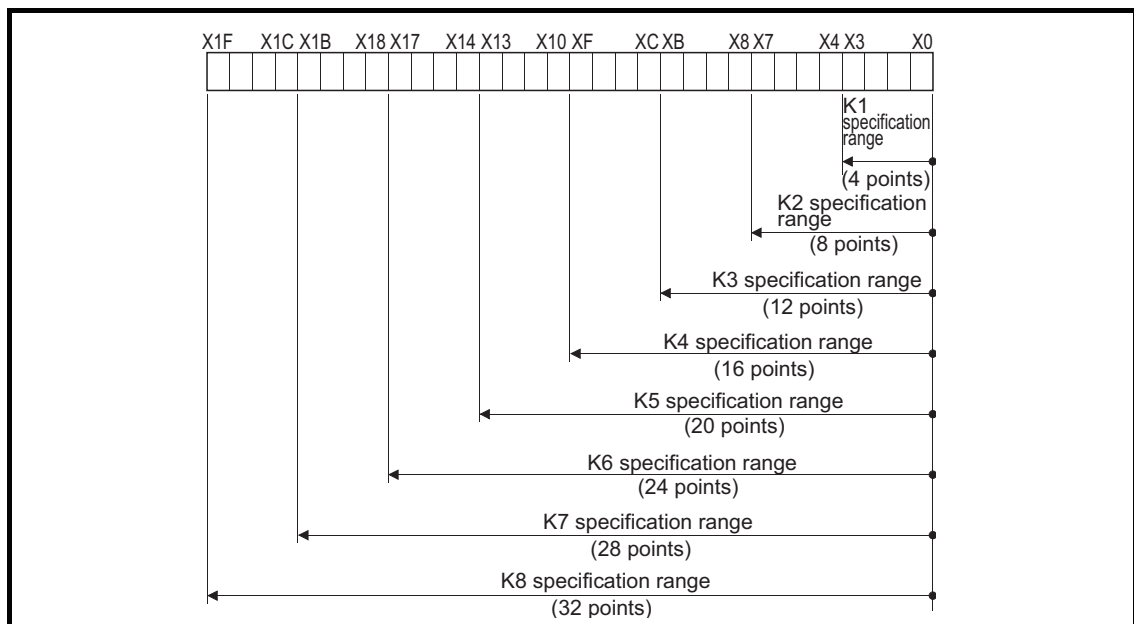
Figure 4.4.3-1 Digit specification setting range of double word data (32 bits)

    (b) The following table shows the numeric values that can be used as source data when a digit is specified at the source ⓢ.

Table 4.4.3-1 List of digit specification and numeric values that can be used

Number of specified digits	Value range	Number of specified digits	Value range
K1 (4 points)	0 to 15	K5 (20 points)	0 to 1048575
K2 (8 points)	0 to 255	K6 (24 points)	0 to 16777215
K3 (12 points)	0 to 4095	K7 (28 points)	0 to 268435455
K4 (16 points)	0 to 65535	K8 (32 points)	−2147483648 to 2147483647

(c) When the destination is a word device

For the word device at the destination side, 0s are stored to bits starting from the digit specified at the source. (Data_s:K1X0, Data_d:D0)

Ladder example	Processing
Instruction that processes 32-bit data	
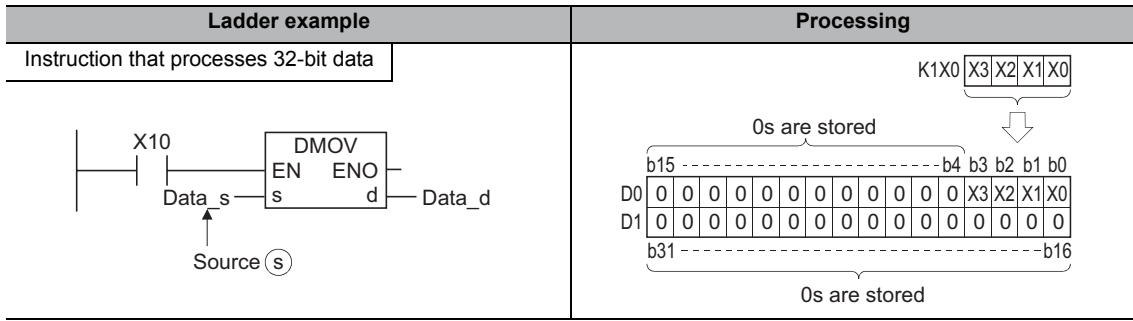
Figure 4.4.3-2 Ladder example and processing details

(d) When a digit is specified at the destination ⓓ, the points by the specified digit are the target of destination. (Data_d1:K5M0, Data_d2:K5M10, Data_s:D0)

Bit devices following the points by the specified digit are not changed.

Ladder example	Processing
When the source ⓢ is a numeric value	
When the source ⓢ is a word device	
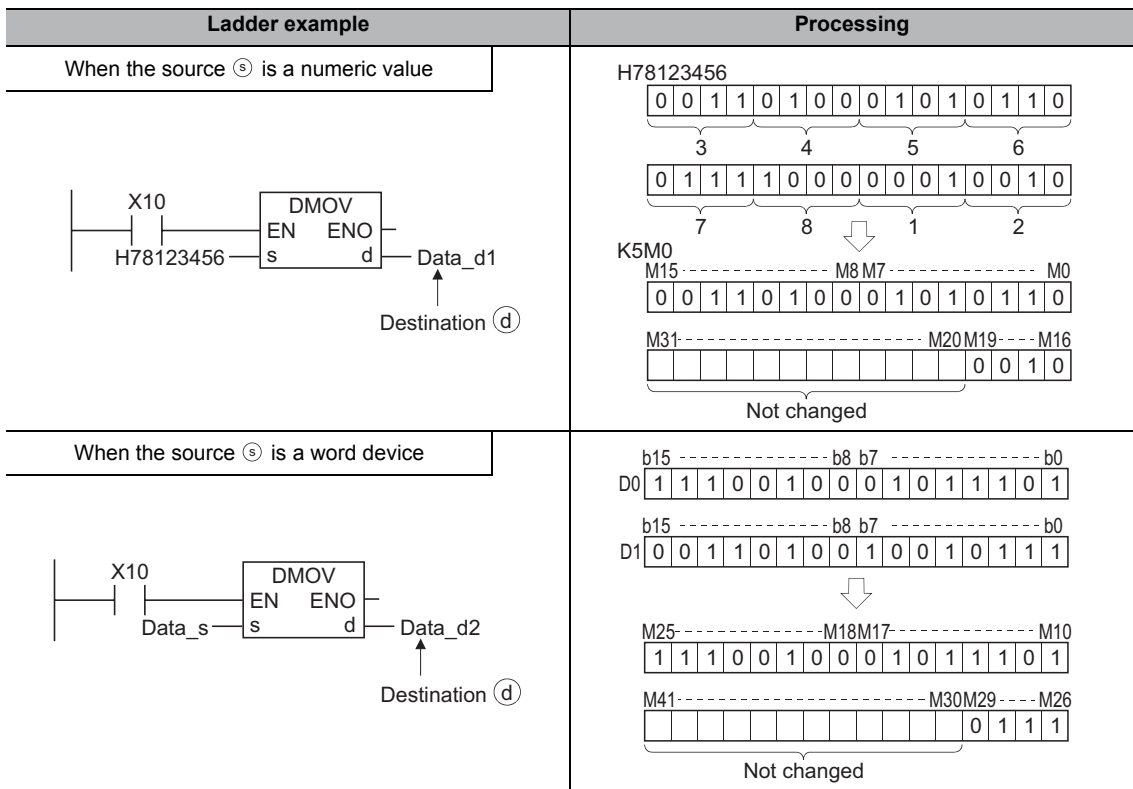
Figure 4.4.3-3 Ladder example and processing details

⊠ **POINT**

1. When performing the process with a digit specification, a desired value can be used for the start device number of bit devices.
2. Digits cannot be specified for direct access inputs/outputs (DX, DY).

4.4 Method for Specifying Data
4.4.3 Double word (32 bits) data

(2) Using word devices

Devices used in lower 16 bits are specified for a word device.
'Specified device number' and 'specified device number +1' are used for instructions that process 32-bit data.

```
    M0            DMOV      ┌→ Transfers 32-bit data
 ──┤ ├──         EN   ENO ─┤
          100 ── s      d ─┴─ D0
                                │
                                ↑
```
Two points (32 bits) of D0 and D1
are the target word devices
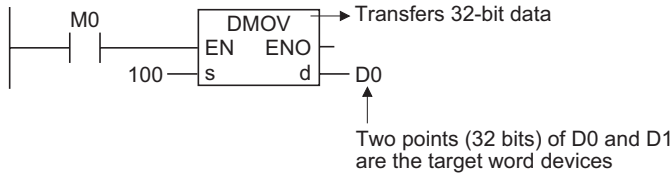
4.4.4 Single-precision real/double-precision real data

Single-precision real/double-precision real data are 32-bit floating-point data used in basic instructions and application instructions.

Real number data can be stored only in word devices.

(1) Single-precision real (single-precision floating-point data)

Devices used in lower 16 bits are specified for instructions that use real number data.

Real number data are stored in 32 bits of 'specified device number' and 'specified device number +1'.

```
    M0            EMOV      ┌→ Transfers real number data
 ──┤ ├──         EN   ENO ─┘
      Var_R100 ── s      d ── Var_D0 ──→ Two points (32 bits) of D0 and D1
             │                           are the target word devices
             └→ Two points (32 bits) of R100 and R101 are the target word devices
```
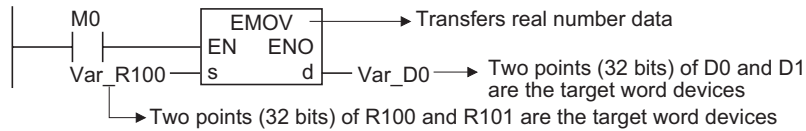
Remark •

Floating-point data are represented by two word devices.

[Sign] 1. [Fraction] $\times 2^{[Exponent]}$

The following explains the bit configuration and its meaning when floating-point data are internally represented.

| b31 | b30 to b23 | b22 to b16 | b15 to b0 |

b31
Sign

b23 to b30
Exponent

b0 to b22
Fraction
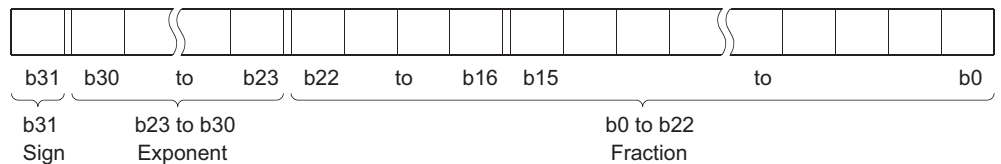
• Sign
b31 represents a sign.
0: Positive
1: Negative

• Exponent
b23 to b30 represent n of 2^n.
The values of n are as follows according to BIN values of b23 to b30.

b23 to b30	FF_H	FE_H	FD_H		81_H	80_H	7F_H	7E_H		02_H	01_H	00_H
n	Not used	127	126		2	1	0	-1		-125	-126	Not used

• Fraction
23 bits of b0 to b22 represent a value of XXXXXX... when the fraction is expressed as 1.XXXXXX... in binary.

• •

(2) Double-precision real (double-precision floating-point data)

Devices used in lower 16 bits are specified for instructions that use real number data.
Real number data are stored in 64 bits of 'specified device number' and 'specified device number + 3'.


```
     M0            EDMOV              ──► Transfers real number data
   ──┤ ├──       ┌─EN    ENO─┐
              Var_R100 ──────d── Var_D0 ──► Four points (64 bits) of D0, D1, D2, and D3
                     │                        are the target word devices
                     └──► Four points (64 bits) of R100, R101, R102, and R103 are the target word devices
```
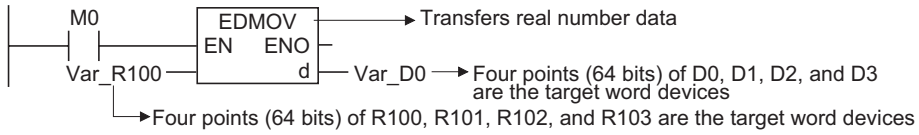
Remark •

1) Floating-point data are represented by four word devices.

$$[\text{Sign}]\ 1.\ [\text{Fraction}] \times 2^{[\text{Exponent}]}$$

The following explains the bit configuration and its meaning when floating-point data are internally represented.

| b63 | b62 | to | b52 | b51 | to | b16 | b15 | to | b0 |

b63 — Sign
b52 to b62 — Exponent
b0 to b51 — Fraction
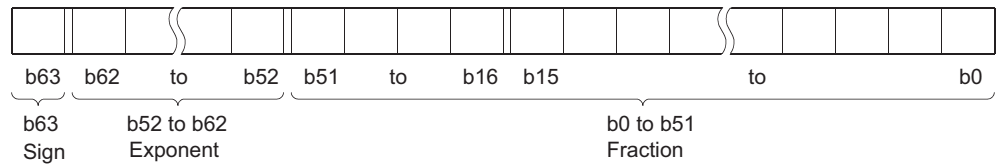
- Sign

  b63 represents a sign.

  0: Positive

  1: Negative

- Exponent

  b52 to b62 represent n of 2^n.

  The values of n are as follows according to BIN values of b52 to b62.

b52 to b62	7FF$_H$	7FE$_H$	7FD$_H$		400$_H$	3FF$_H$	3FE$_H$	3FD$_H$	3FC$_H$		02$_H$	01$_H$	00$_H$
n	Not used	1023	1022		2	1	0	-1	-2		-1021	-1022	Not used

- Fraction

  52 bits of b0 to b51 represent a value of XXXXXX... when the fraction is expressed as 1.XXXXXX... in binary.

• •

4.4 Method for Specifying Data
4.4.4 Single-precision real/double-precision real data

⊠POINT

1. Floating-point data in a CPU module can be monitored by the monitoring function of the programming tool.

2. To express 0 in floating-point data, set all of the following bits to 0.
   - (a) Single-precision floating-point data: b0 to b31
   - (b) Double-precision floating-point data: b0 to b63

3. The setting range of real number is shown below:[1]
   - (a) Single-precision floating-point data

     $-2^{128} < \text{Device} \leq -2^{-126},\ 0,\ 2^{-126} \leq \text{Device} < 2^{128}$
   - (b) Double-precision floating-point data

     $-2^{1023} \leq \text{Device} \leq -2^{-1022},\ 0,\ -2^{-1022} \leq \text{Device} \leq 2^{1023}$

4. Do not specify -0 (when only the highest bit of the floating-point real number is 1) for floating-point data. (A floating-point operation with -0 results an operation error.)

   For a CPU module that performs an internal operation of floating-point operation with double precision, a floating-point operation does not result an error since -0 is converted to 0 in a CPU module when -0 is specified.

   For a CPU module that performs an internal operation of floating-point operation with single precision, a floating-point operation results an error since the process speed is given a priority and -0 is used for an operation as it is when -0 is specified.
   - (a) The following is the CPU module in which the operation does not result an error when -0 is specified.
     - • High Performance model QCPU in which the internal operation is set to double precision[2] (The default setting of internal floating-point operation is double precision.)
   - (b) The following are the CPU modules in which the operation results an error when -0 is specified.
     - • Basic model QCPU[3]
     - • High Performance model QCPU in which the internal operation is set to single precision[2]
     - • Universal model QCPU
     - • LCPU

*1: For operations when an overflow or underflow is occurred or when a special value is input, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used.

*2: Switching between single precision and double precision of the internal floating-point operation is set in the PLC system setting of the PLC parameter. For single precision and double precision of floating point operation, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used.

*3: The floating point operation is supported with the Basic model QCPU with a serial number whose first five digits are '04112' or higher.

4.4.5 String data

String data are character data used in basic instructions and application instructions.

From the specified character to the NULL code (00H) that indicates the end of the character string are the target string data.

(1) When the specified character is NULL code

The NULL code is stored by using one word.

Specification of NULL code (00H)
Transfers character string data
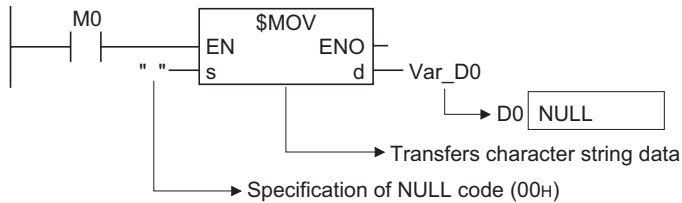
(2) When the number of characters is an even number

Character string data and NULL code are stored by using the 'number of characters /2+1' words.
For example, when 'ABCD' is transferred to word devices starting from D0, the character string 'ABCD' is stored to D0 and D1, and the NULL code to D2. (The NULL code is stored to the last one word).

Transfers character string data
Specification of a character string composed of even numbers
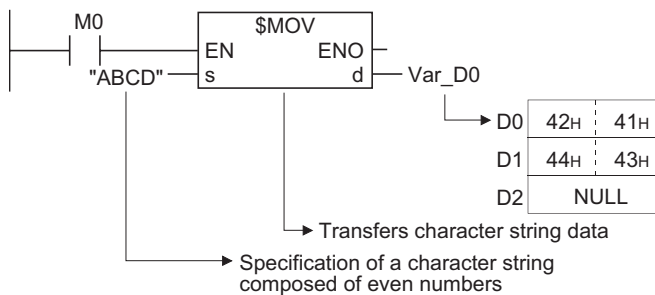
(3) When the number of characters is an odd number

Character string data and NULL code are stored by using the 'number of characters /2' words (Rounding the fractional part).
For example, when 'ABCDE' is transferred to word devices starting from D0, the character string 'ABCDE' and the NULL code are stored to D0 to D2. (The NULL code is stored to the higher 8 bits of the last one word).

Transfers character string data
Specification of a character string composed of odd numbers
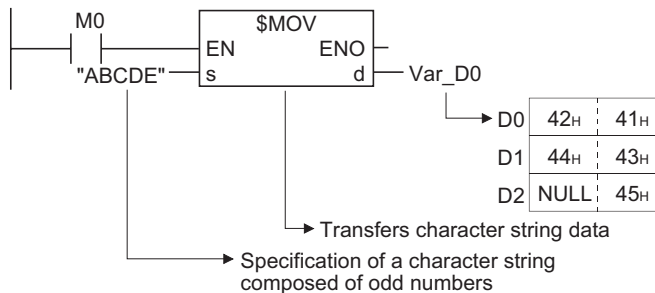
4.4.6 Time data

Time data are used in time type operation instructions of application functions.

Specify time data in the T#10d20h30m40s567ms form.

For example. the following adds '1 Day, 2 Hours, 3 Minutes, and 4 Seconds' to '10 Days, 20 Hours, 30 Minutes, 40 Seconds, and 567 Milliseconds'.

```
T#10d20h30m40s567ms ── g_time1


                       ┌──────────┐
                       │ ADD_TIME │
            g_time1 ───│_IN1      │─── g_time2
          T#1d2h3m4s ──│_IN2      │
                       └──────────┘
```
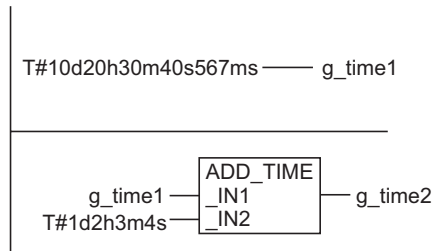
Each value of time data can be specified within the following range.

Table 4.4.6-1 Allowable specification range of time data

Value	Range
d (Day)	0 to 24
h (Hour)	0 to 23
m (Minute)	0 to 59
s (Second)	0 to 59
ms (Millisecond)	0 to 999

For application functions, refer to the following manuals.

☞ MELSEC-Q/L Structured Programming Manual (Application Functions)

☞ FXCPU Structured Programming Manual (Application Functions)

4.4.7 Arrays

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

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

(☞ GX Works2 Version1 Operating Manual (Structured Project))

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

One-dimensional array

Two-dimensional array

Label name · · · Index   Label name · · · Index

boolary1

[1]
[2]
⋮
[n]

boolary2

[0,0]	[0,1]	· · ·	[0,n]
[1,0]	[1,1]	· · ·	⋮
⋮	⋮	⋮	⋮
[m,0]	· · ·	· · ·	[m,n]
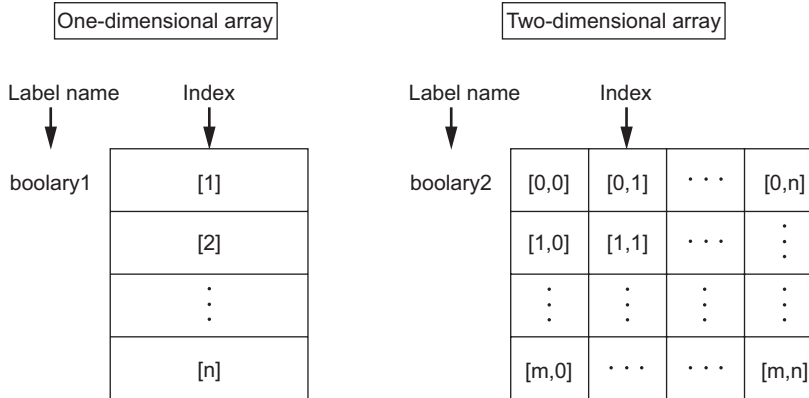
(1) Definition of arrays

The following table shows the format of definition.

Table 4.4.7-1 Form used to define array

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 ☞ Section 4.3.5 For structured data types ☞ Section 4.4.8
	(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)	

(2) Expression of arrays

To identify individual labels of an array, append an index enclosed by '[ ]' after the label name.
Values that can be specified for indexes are within the range from -32768 to 32767.

Example)

boolary1 [0]   boolary2 [0.3]

Label name   index
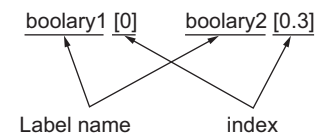
For an array with two or more dimensions, delimit indexes in '[ ]' by ','.
For the ST and structured ladder languages, labels (word (signed) or double word (signed) data type) can be used for indexes as shown on the next page.
Note that Z0 or Z1 cannot be used in the programs if labels are used for indexes.

[Structured ladder]

```
          MOV
     EN        ENO
  0  s          d ── Index1

          FOR
     EN        ENO
  4  n

          INC
     EN        ENO
             d ── Var_D0[Index1]

          INC
     EN        ENO
             d ── Index1

          NEXT
     EN        ENO
```
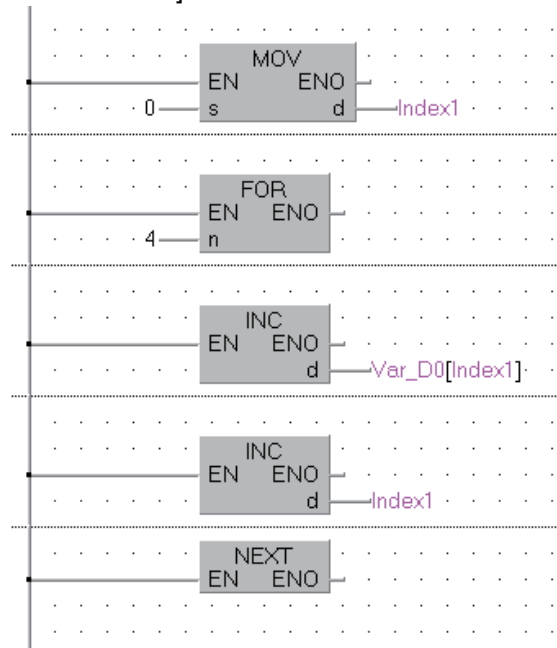
[ST]
FOR Index1:=0
    TO 4
    BY 1 DO
    INC(TRUE,Var_D0[Index1]);
END_FOR;

(3) Maximum number of array elements
   The maximum number of array elements differs depending on data types as shown below.

Table 4.4.7-2 Maximum number of array

Data type	Maximum number
Bit, word (signed), word (unsigned)/16-bit string, timer, counter, and retentive timer	32768
Double word (signed), double word (unsigned)/32-bit string, single-precision real, and time	16384
Double-precision real	8192
String	32768 divided by string length

4.4.8 Structures

A structure is an aggregation of different data type labels.

Structures can be used in all POUs.

To use structures, first create the configuration of structure, and define a structure label name for the created structure as a new data type

(☞ GX Works2 Version1 Operating Manual (Structured Project))

To use each element of structure, append an element name after the structure label name with '.' as a delimiter in between.

Example) When using the element
of the structured data

dut_a1 . in00

Structure   Element name
label name
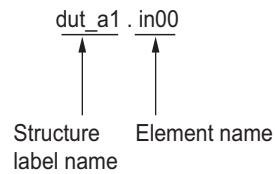
Structures can also be used as arrays. When a structure is declared as an array, append an index enclosed by '[ ]' after the structure label name.

The arranged structured data can be specified as arguments of functions and function blocks.

Example) When using the element
of the arranged structured data

dut_b1 [0]   . in00

Structure   Index   Element
label name   name

Creating structures	

Structure name	Element
samp_fb1	Bit bo00
	Bit bo01
	Word (signed) in00
	.
	.
	.

Define labels

Structure label name	Structure name
dut_a1	samp_fb1

Expression in a program

```
    dut_a1.bo00      dut_a1.bo01
    ──┤├────┬──────────┤├──────
            │
            │          ┌──────────┐
            │          │   MOV    │
            │        ──┤EN    ENO ├──
  dut_a1.in00 ─────────┤s       d ├── idata1
            │          └──────────┘
```
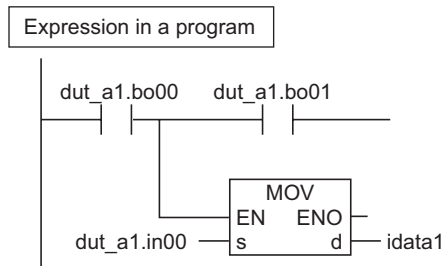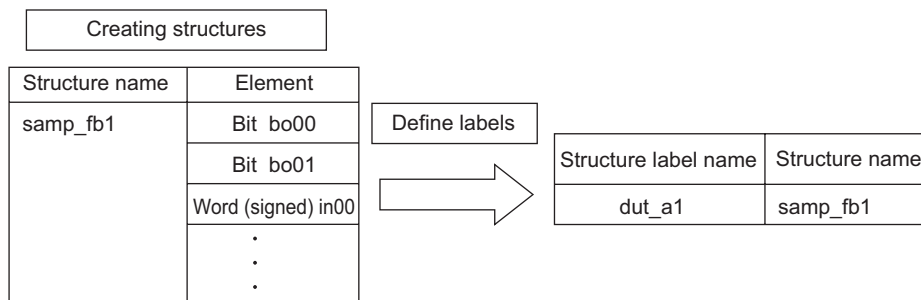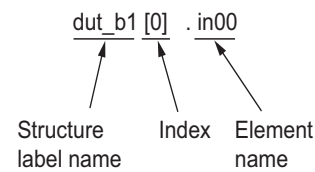
4.5 Device and Address

This section explains the method for expressing programmable controller CPU devices. The following two types of format are available.

- Device: This format consists of a device name and a device number.
- Address: A format defined in IEC61131-3. In this format, a device name starts with %.

4.5.1 Device

Device is a format that uses a device name and a device number.
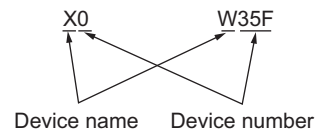
For details of devices, refer to the following manuals.

☞ User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used

☞ FXCPU Structured Programming Manual (Device & Common)

Example)

X0   W35F

Device name   Device number

4.5.2 Address

Address is a format defined in IEC61131-3.

The following table shows details of format that conforms to IEC61131-3.

Table 4.5.2-1 Address definition specifications

Start	1st character: position		2nd character: data size		3rd character and later: classification	Number
%	I	Input	(Omitted)	Bit	Numerics used for detailed classification Use '.' (period) to delimit the numbers from the subsequent numbers. A period may be omitted.	Number corresponding to the device number (decimal notation)
	Q	Output	X	Bit		
	M	Internal	W	Word (16 bits)		
			D	Double word (32 bits)		
			L	Long word (64 bits)[*1]		

*1: Not supported with the FXCPU.

● Position

Position is a major class indicating the position to which data are allocated in three types: input, output, and internal.
The following shows the format rules corresponding to the device format.

Example)

%IX0   %MX1 . 863

Position Data Classification Number
          size

- X, J\X (X device) : I (input)

- Y, J\Y (Y device) : Q (output)

- Other devices   : M (internal)
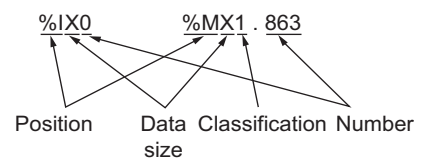
● Data size

Data size is a class indicating the size of data.
The following shows the format rules corresponding to the device format.

- Bit device   : X (bit)

- Word device   : W (word), D (double word), L (long word)

● Classification

Classification is a minor class indicating the type of a device that cannot be identified only by its position and size.
Devices X and Y do not support classification.
For the format corresponding to the device format, refer to the following section.

☞ Section 4.5.3 Correspondence between devices and addresses

⊠POINT

Long words are used in double-precision real operation instructions of the Universal model QCPU/LCPU.

4.5.3 Correspondence between devices and addresses

This section explains the correspondence between devices and addresses.

(1) Correspondence between devices and addresses
     The following table shows the correspondence between devices and addresses.

(a) QCPU (Q mode)/LCPU

Table 4.5.3-1 Correspondence between devices and addresses (1/2)

Device			Expressing method		Example of correspondence between device and address	
			Device	Address	Device	Address
Input		X	Xn	%IXn	X7FF	%IX2047
Output		Y	Yn	%QXn	Y7FF	%QX2047
Internal relay		M	Mn	%MX0.n	M2047	%MX0.2047
Latch relay		L	Ln	%MX8.n	L2047	%MX8.2047
Annunciator		F	Fn	%MX7.n	F1023	%MX7.1023
Special relay		SM	SMn	%MX10.n	SM1023	%MX10.1023
Function input		FX	FXn	None	FX10	None
Function output		FY	FYn	None	FY10	None
Edge relay		V	Vn	%MX9.n	V1023	%MX9.1023
Direct access input		DX	DXn	%IX1.n	DX7FF	%IX1.2047
Direct access output		DY	DYn	%QX1.n	DY7FF	%QX1.2047
Timer	Contact	TS	Tn	%MX3.n	TS511	%MX3.511
	Coil	TC	Tn	%MX5.n	TC511	%MX5.511
	Current value	TN	Tn	%MW3.n %MD3.n	TN511 T511	%MW3.511 %MD3.511
Counter	Contact	CS	Cn	%MX4.n	CS511	%MX4.511
	Coil	CC	Cn	%MX6.n	CC511	%MX6.511
	Current value	CN	Cn	%MW4.n %MD4.n	CN511 C511	%MW4.511 %MD4.511
Retentive timer	Contact	STS	STn	%MX13.n	STS511	%MX13.511
	Coil	STC	STn	%MX15.n	STC511	%MX15.511
	Current value	STN	STn	%MW13.n %MD13.n	STN511 ST511	%MW13.511 %MD13.511
Data register		D	Dn	%MW0.n %MD0.n	D11135	%MW0.11135 %MD0.11135
Special register		SD	SDn	%MW10.n %MD10.n	SD1023	%MW10.1023 %MD10.1023
Function register		FD	FDn	None	FD0	None
Link relay		B	Bn	%MX1.n	B7FF	%MX1.2047
Link special relay		SB	SBn	%MX11.n	SB3FF	%MX11.1023
Link register		W	Wn	%MW1.n %MD1.n	W7FF	%MW1.2047 %MD1.2047
Link special register		SW	SWn	%MW11.n %MD11.n	SW3FF	%MW11.1023 %MD11.1023
Intelligent function module device		G	Ux\Gn	%MW14.x.n %MD14.x.n	U0\G65535	%MW14.0.65535 %MD14.0.65535
File register		R	Rn	%MW2.n %MD2.n	R32767	%MW2.32767 %MD2.32767
Pointer		P	Pn	"" (Null character)	P299	None
Interrupt pointer		I	In	None	-	-
Nesting		N	Nn	None	-	-
Index register		Z	Zn	%MW7.n %MD7.n	Z9	%MW7.9 %MD7.9

4

PROGRAM
CONFIGURATION

Table 4.5.3-1 Correspondence between devices and addresses (2/2)

Device		Expressing method		Example of correspondence between device and address	
		Device	Address	Device	Address
Step relay	S	Sn	%MX2.n	S127	%MX2.127
SFC transition device	TR	TRn	%MX18.n	TR3	%MX18.3
SFC block device	BL	BLn	%MX17.n	BL3	%MX17.3
Link input	J	Jx\Xn	%IX16.x.n	J1\X1FFF	%IX16.1.8191
Link output		Jx\Yn	%QX16.x.n	J1\Y1FFF	%QX16.1.8191
Link relay		Jx\Bn	%MX16.x.1.n	J2\B3FFF	%MX16.2.1.16383
Link register		Jx\Wn	%MW16.x.1.n %MD16.x.1.n	J2\W3FFF	%MW16.2.1.16383 %MD16.2.1.16383
Link special relay		Jx\SBn	%MX16.x.11.n	J2\SB1FF	%MX16.2.11.511
Link special register		Jx\SWn	%MW16.x.11.n %MD16.x.11.n	J2\SW1FF	%MW16.2.11.511
File register	ZR	ZRn	%MW12.n %MD12.n	ZR32767	%MW12.32767 %MD12.32767

(b) FXCPU

Table 4.5.3-2 Correspondence between devices and addresses

Device			Expressing method		Example of correspondence between device and address	
			Device	Address	Device	Address
Input		X	Xn	%IXn	X367	%IX247
Output		Y	Yn	%QXn	Y367	%QX247
Auxiliary relay		M	Mn	%MX0.n	M499	%MX0.499
Timer	Contact	TS	Tn	%MX3.n	TS191	%MX3.191
	Coil	TC	Tn	%MX5.n	TC191	%MX5.191
	Current value	TN	Tn	%MW3.n %MD3.n	TN191 T190	%MW3.191 %MD3.190
Counter	Contact	CS	Cn	%MX4.n	CS99	%MX4.99
	Coil	CC	Cn	%MX6.n	CC99	%MX6.99
	Current value	CN	Cn	%MW4.n %MD4.n	CN99 C98	%MW4.99 %MD4.98
Data register		D	Dn	%MW0.n %MD0.n	D199 D198	%MW0.199 %MD0.198
Intelligent function module device		G	Ux\Gn	%MW14.x.n %MD14.x.n	U0\G09	%MW14.0.10 %MD14.0.9
Extension register		R	Rn	%MW2.n %MD2.n	R32767 R32766	%MW2.32767 %MD2.32766
Extension file register		ER	ERn	None	–	–
Pointer		P	Pn	"" (Null character)	P4095	None
Interrupt pointer		I	In	None	–	–
Nesting		N	Nn	None	–	–
Index register		Z	Zn	%MW7.n %MD7.n	Z7 Z6	%MW7.7 %MD7.6
		V	Vn	%MV6.n	V7	%MW6.7
State		S	Sn	%MX2.n	S4095	%MX2.4095

(2) Digit specification for bit devices

The following table shows the correspondence between devices and addresses when a digit is specified for a bit device.

Table 4.5.3-3 Correspondence of formats with digit specification

Device	Address
K[Number of digits][Device name][Device number] (Number of digits: 1 to 8)	%[Position of memory area][Data size]19.[Number of digits].[Classification].[Number] (Number of digits: 1 to 8)

• Correspondence examples

Device	Address
K1X0	%IW19.1.0
K4M100	%MW19.4.0.100
K8M100	%MD19.8.0.100
K2Y7E0	%QW19.2.2016

(3) Bit specification for word devices

The following table shows the correspondence between devices and addresses when a bit is specified for a word device.

Table 4.5.3-4 Correspondence of formats with bit specification

Device	Address
[Device name][Device number].[Bit number] (Bit number: 0 to F)	%[Position of memory area]X[Classification].[Device number].[Bit number]

• Correspondence examples

Device	Address
D11135.C	%MX0.11135.12
SD1023.F	%MX10.1023.15

☒**POINT**

• Index setting, digit specification, and bit specification
  Index setting, digit specification, and bit specification cannot be applied to labels.

4.6 Index Setting

(1) Overview of the index setting

    (a) The index setting is an indirect setting that uses index registers.
When the index setting is used in a sequence program, the device consists of 'directly specified device number' + 'content of index register'.
For example, when D2Z2 is specified and the value of Z2 is 3, D(2+3)=D5 is set as the target.

    (b) For Universal model QCPU, LCPU, and FXCPU, indexes can be set in 32-bit range in addition to 16-bit range.

(2) 16-bit index setting

    (a) Setting an index in 16-bit range

        Values from -32768 to 32767 can be set to index registers.
The following shows how the index is set.
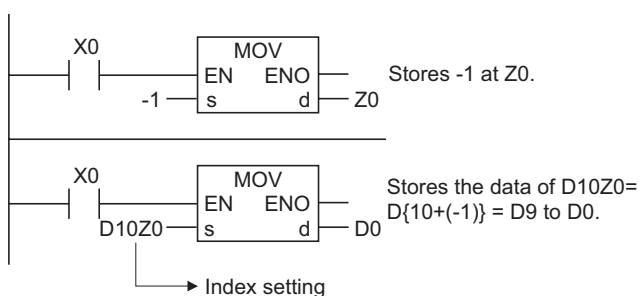
    (b) Devices that can be used for the index setting (for QCPU (Q mode), LCPU)

        The index setting can be applied to devices used by contacts, coils, basic instructions, and application instructions except for the restrictions shown in the tables below. The index setting cannot be applied to labels.

        1) Devices that cannot be used for the index setting

Device	Description
E	Floating-point data
$	Character string data
☐.☐ (D0.1 etc.)	Bit specification for word devices
FX, FY, FD	Function devices
P	Pointers used as labels
I	Interrupt pointers used as labels
Z	Index registers
S	Step relays
TR	SFC transition devices[1]
BL	SFC block devices[1]

*1: SFC transition devices and SFC block devices are devices for SFC programs. For details, refer to the following manual.
    • MELSEC-Q/L/QnA Programming Manual (SFC)

2) Devices with restrictions on index registers

Device	Description	Example
T	Only Z0 or Z1 can be used for contacts or coils of the timer.	
C	Only Z0 or Z1 can be used for contacts or coils of the counter.	

(c) Devices that can be used for the index setting (for FXCPU)

The following table shows the devices that can be used for the index setting.

Device	Description
M, S, T, C, D, R, KnM, KnS, P, K	Decimal devices, values
X, Y, KnX, KnY	Octal devices
H	Hexadecimal values

1) Devices with restrictions on index registers

When using FXCPU, note the following precautions.

• The index setting for devices used in the basic instructions is available for FX3U and FX3UC only.

• The index setting cannot be applied to 32-bit counter and special auxiliary relay.

Remark •

There are no usage restrictions on index register numbers for current values of the timer and counter.
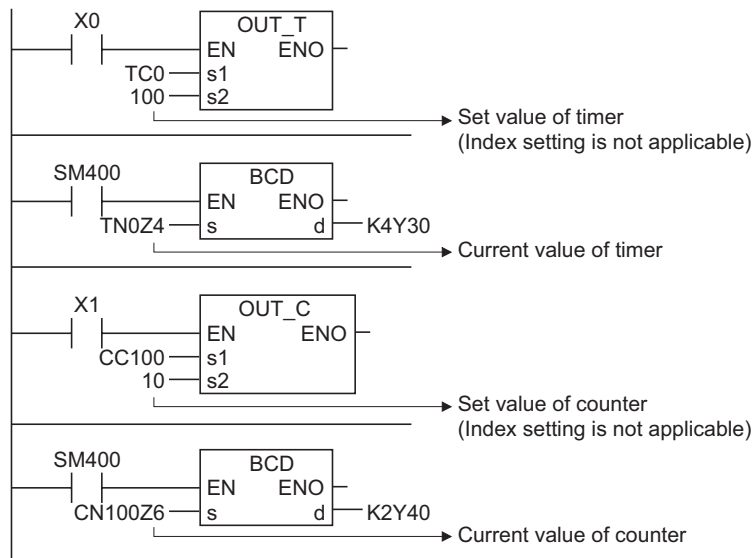
(d) The following figure shows the examples of index setting and their actual processing devices.
(With the setting of Z0=20 and Z1=−5)

Ladder example	Actual processing device
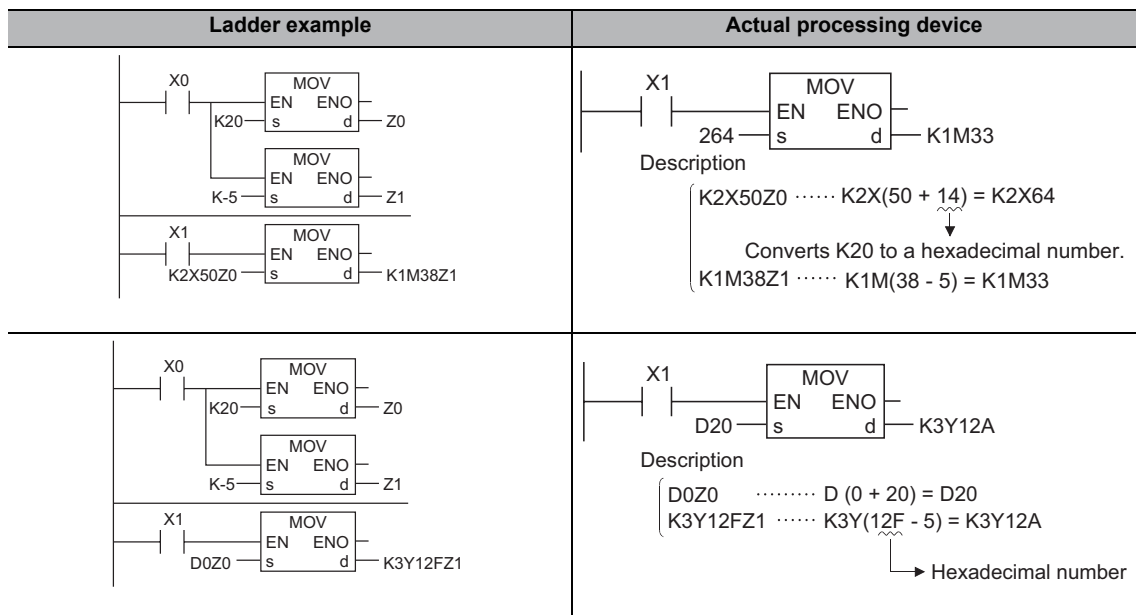
Figure. 4.6-1 Ladder examples and actual processing devices

(3) 32-bit index setting (for Universal model QCPU (excluding Q00UJCPU), LCPU, and FXCPU)

For Universal model QCPU (excluding Q00UJCPU) and LCPU, either of the following two methods can be selected to specify index registers used for a 32-bit index setting.

・ Specify a range of index registers used for a 32-bit index setting.

・ Specify a 32-bit index setting using 'ZZ'.

For FXCPU, combine index registers V (from V0) and Z (from Z0) for a 32-bit index setting.

⊠POINT

32-bit index settings using 'ZZ' can be used for the following CPU modules only.

・QnU(D)(H)CPU with a serial number whose first five digits are '10042' or higher (excluding Q00UJCPU)

・QnUDE(H)CPU

・LCPU

(a) Specifying a range of index registers used for a 32-bit index setting

    1) Values from -2147483648 to 2147483647 can be set to index registers. The following shows how the index is set.
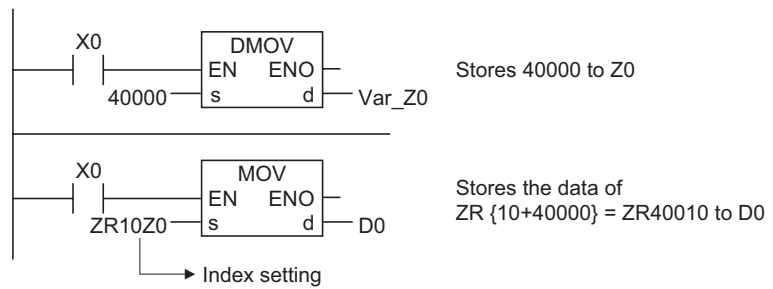
2) Specification method

When setting indexes in 32-bit range, specify the start number of index registers to be used in "Indexing Setting for ZR Device" setting in the <<Device>> tab of the PLC parameter.
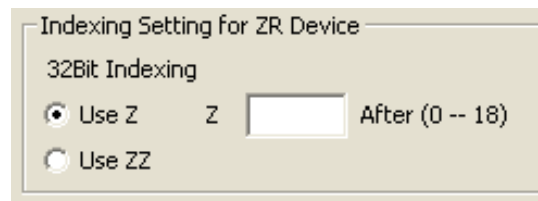
Figure 4.6-2 Index setting for ZR device on the parameter setting screen

⊠**POINT**

When changing the start number of index registers to be used in the device setting of the PLC parameter, do not change nor write only parameters to the programmable controller. Always write parameters along with the program to the programmable controller.

If data are forcibly written, the operation error "CAN'T EXE. PRG." (error code: 2500) occurs.

3) Devices that can be used for index settings

Only the following devices can be used for index settings.

Device	Description
ZR	Serial number access file register
D	Extended data register
W	Extended link register

4) Usage range of index registers

The following table shows the usage range of index registers when setting indexes in 32-bit range.

Since the specified index register (Zn) and next index register (Zn+1) are used for index setting in 32-bit range, make sure not to overlap index registers being used.

Setting value	Index register	Setting value	Index register
Z0	Z0, Z1	Z10	Z10, Z11
Z1	Z1, Z2	Z11	Z11, Z12
Z2	Z2, Z3	Z12	Z12, Z13
Z3	Z3, Z4	Z13	Z13, Z14
Z4	Z4, Z5	Z14	Z14, Z15
Z5	Z5, Z6	Z15	Z15, Z16
Z6	Z6, Z7	Z16	Z16, Z17
Z7	Z7, Z8	Z17	Z17, Z18
Z8	Z8, Z9	Z18	Z18, Z19
Z9	Z9, Z10	Z19	Not applicable

5) The following figure shows the examples of index setting and their actual processing devices.

(With the setting of Z0 (32 bits) =100000 and Z2 (16 bits) =-20)

Ladder example	Actual processing device
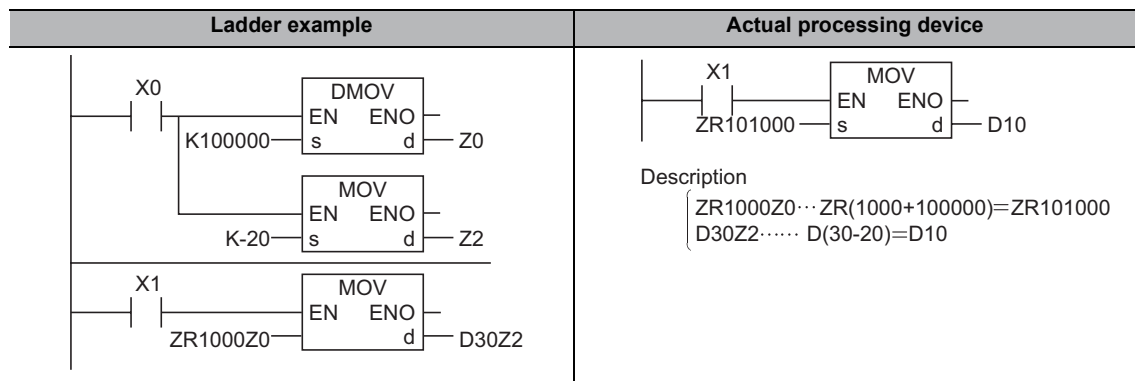
Figure 4.6-3 Ladder examples and actual processing devices

(b) Specifying a 32-bit index setting using 'ZZ'

1) A 32-bit index can be specified to the index register by specifying an index using 'ZZ', for instance, 'ZR0ZZ4'.

The following figure shows the 32-bit index setting using 'ZZ'.

```
   M0          DMOVP
 --| |--      EN   ENO--        Set 100000 to Z4 and Z5.
     K100000--s     d --Z4

   M0           MOVP
 --| |--       EN   ENO--       Set 32-bit (Z4, Z5) index to ZR.
     K100--    s     d --ZR0ZZ4  ZR(0+100000) indicates ZR100000.
```
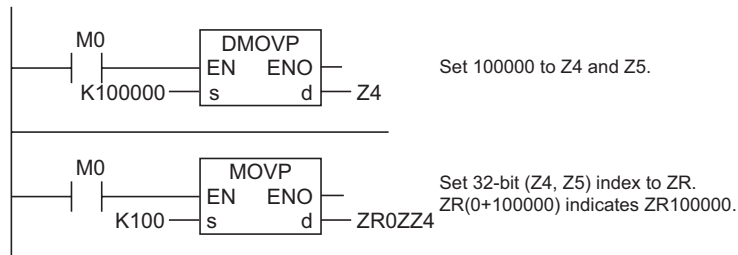
2) Specification method

When specifying a 32-bit index setting using 'ZZ', select "Use ZZ" in the "Indexing Setting for ZR Device" setting in the <<Device>> tab of the PLC parameter.
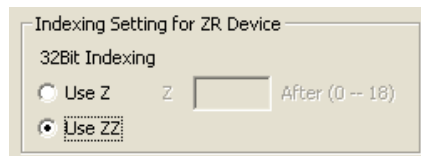
Figure 4.6-4 Index setting for ZR device on the parameter setting screen

3) Devices that can be used for the index setting

Only the following devices can be used for the index setting.

Device	Description
ZR	Serial number access file register
D	Extended data register
W	Extended link register

4) Usage range of index registers

The following table shows the usage range of index registers when specifying 32-bit index setting using 'ZZ'.

When specifying a 32-bit index setting using 'ZZ', specify a device as a form of ZRmZZn.

The device number of ZRm is indexed with 32 bits (Zn, Zn+1) by specifying ZRmZZn.

'ZZ'[*2]	Index register	'ZZ'[*2]	Index register
⬚ZZ0	Z0, Z1	⬚ZZ10	Z10, Z11
⬚ZZ1	Z1, Z2	⬚ZZ11	Z11, Z12
⬚ZZ2	Z2, Z3	⬚ZZ12	Z12, Z13
⬚ZZ3	Z3, Z4	⬚ZZ13	Z13, Z14
⬚ZZ4	Z4, Z5	⬚ZZ14	Z14, Z15
⬚ZZ5	Z5, Z6	⬚ZZ15	Z15, Z16
⬚ZZ6	Z6, Z7	⬚ZZ16	Z16, Z17
⬚ZZ7	Z7, Z8	⬚ZZ17	Z17, Z18
⬚ZZ8	Z8, Z9	⬚ZZ18	Z18, Z19
⬚ZZ9	Z9, Z10	⬚ZZ19	Not applicable

*2: ⬚ Indicates the device name (ZR, D, W) to be indexed.

5) The following figure shows the examples of 32-bit index setting using 'ZZ' and their actual processing devices.

Ladder example	Actual processing device
X0 — DMOV (EN ENO) K100000—s d—Z0 MOV (EN ENO) K-20—s d—Z2 X1 — MOV (EN ENO) ZR1000ZZ0—s d—D30Z2	X1 — MOV (EN ENO) ZR101000—s d—D10 Description $\begin{bmatrix} \text{ZR1000ZZ0} \cdots \text{ZR(1000+100000)} = \text{ZR101000} \\ \text{D30Z2} \cdots \cdots \text{D(30-20)} = \text{D10} \end{bmatrix}$
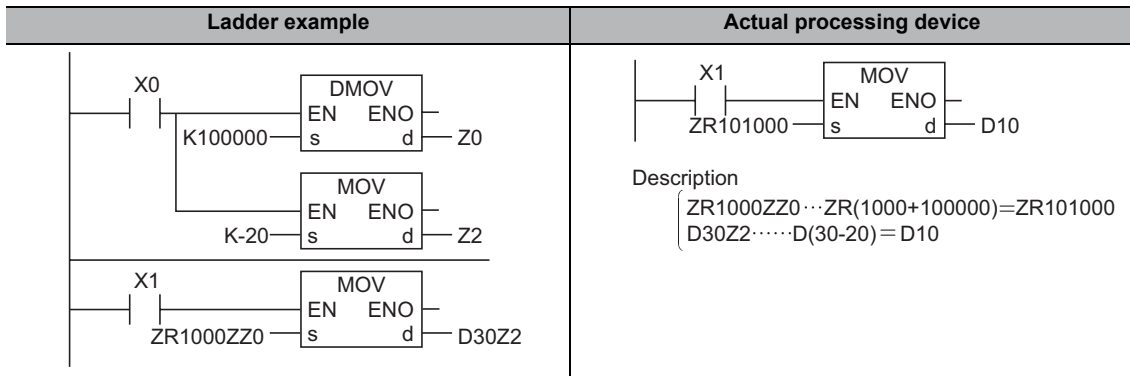
Figure 4.6-5 Ladder examples and actual processing devices

6) Functions that can use 'ZZ'

32-bit index settings using 'ZZ' can be used in the following functions.

No.	Description
1	Device specification of instructions in a program
2	Watch function
3	Current value change function
4	Executional conditioned device test
5	Sampling trace (Trace setting (Data get timing setting), trace target devices)

⊠ POINT

ZZn cannot be used individually such as 'DMOV K100000 ZZ0'. When setting a value to index registers to specify a 32-bit index setting using 'ZZ', set a value to Zn (Z0 to Z19).
ZZn cannot be entered individually in the functions.

(c) 32-bit index setting for FXCPU

Combine index registers V (from V0) and Z (from Z0) for a 32-bit index setting.
V is used for high order and Z is used for low order. With the combination of the specified Z and the corresponding V, the device can be used as a 32-bit register.
Note that the index setting is not applied by specifying the high order V.
Example: When specifying Z4, V4 and Z4 are used as a 32-bit register.

Setting value	Index register
Z0	V0, Z0
Z1	V1, Z1
Z2	V2, Z2
Z3	V3, Z3
Z4	V4, Z4
Z5	V5, Z5
Z6	V6, Z6
Z7	V7, Z7

(4) Applying index settings to extended data registers (D) and extended link registers (W) (for Universal model QCPU (excluding Q00UJCPU), and LCPU)
As an index setting can be applied to internal user devices, data registers (D) and link registers (W), the device specification by the index setting can be used within the range of extended data registers (D) and extended link registers (W).
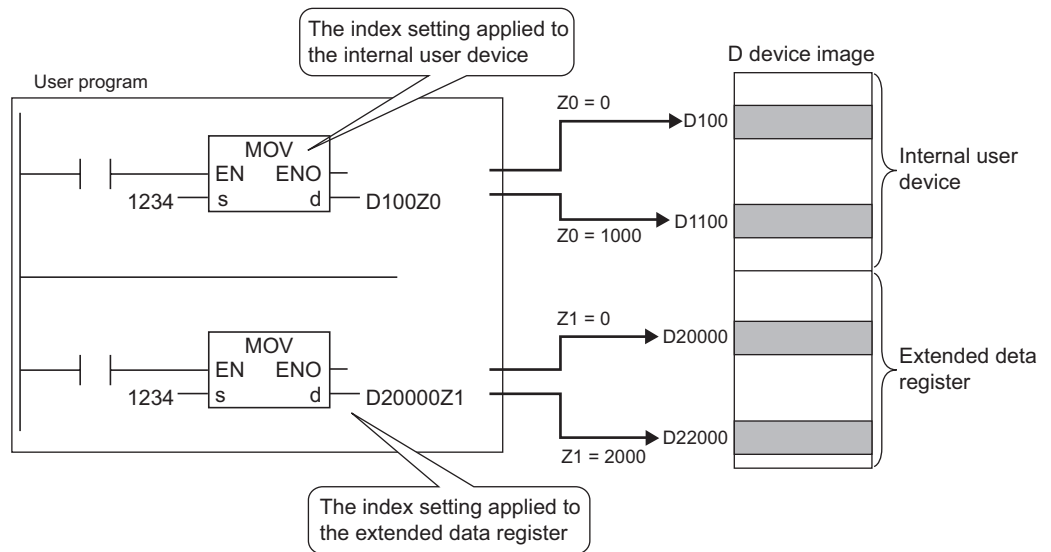
1) Index settings that cross internal user devices and extended data registers (D)/extended link registers (W)

An index setting that crosses internal user devices and extended data registers (D)/extended link registers (W) cannot be applied. If the device range check is enabled at the index setting, an error occurs. (Error code: 4101)
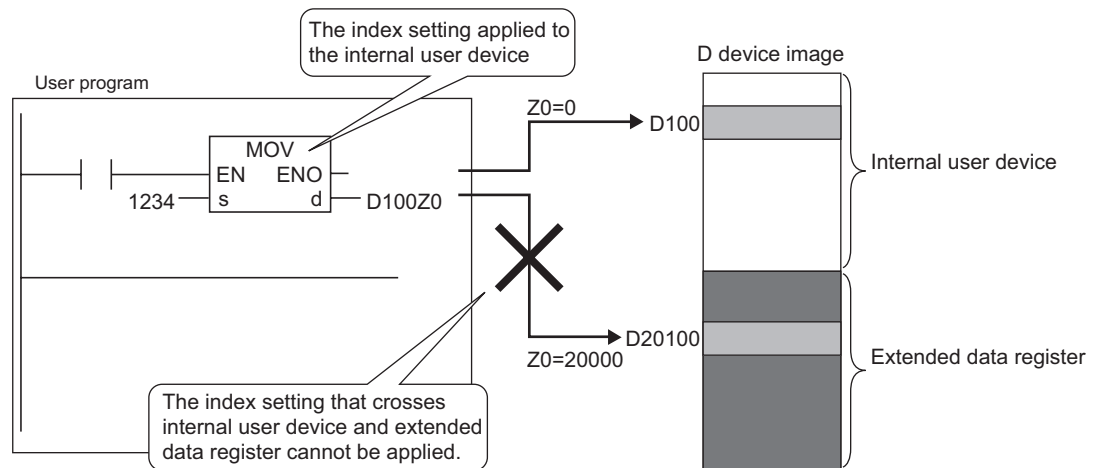
2) Index settings that cross file registers (ZR), extended data registers (D), and extended link registers (W)

Even when an index setting that crosses file registers (ZR), extended data registers (D), and extended link registers (W) is applied, an error does not occur.

However, if the result of the index setting applied to file registers (ZR), extended data registers (D) or extended link registers (W) exceeds the range of the file register files, an error occurs. (Error code: 4101)
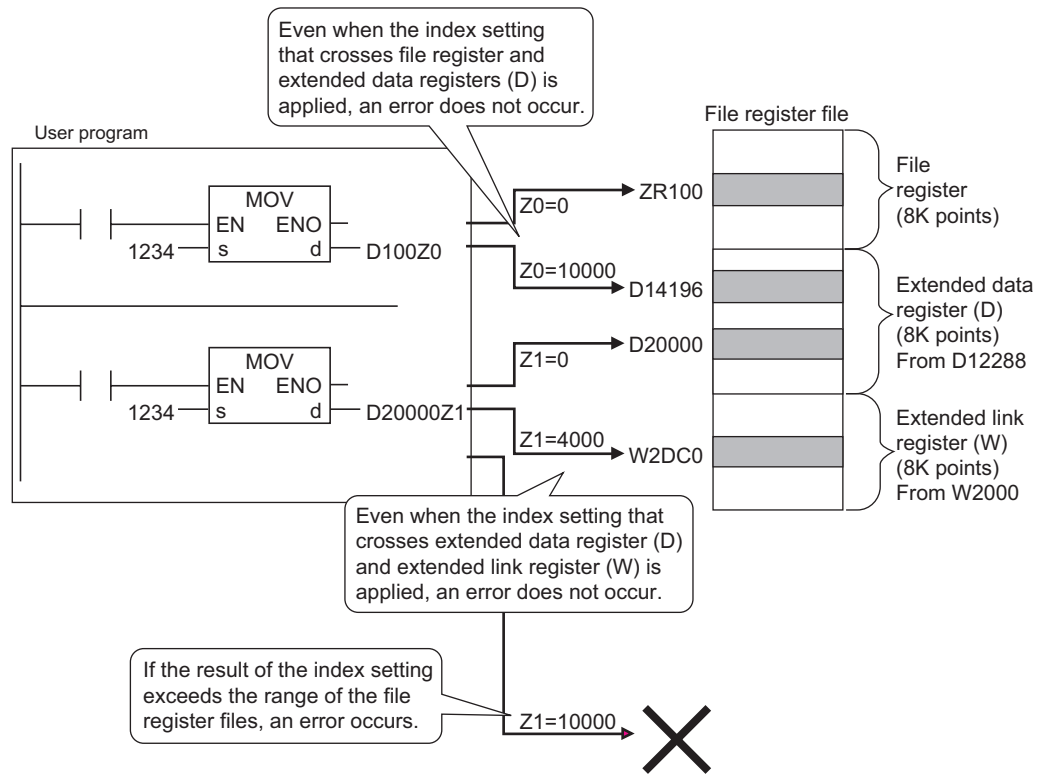
(5) Other applicable data

(a) Bit data

An index setting is applicable to device numbers whose digits are specified.
Note that an index setting is not applicable to the specified digits.
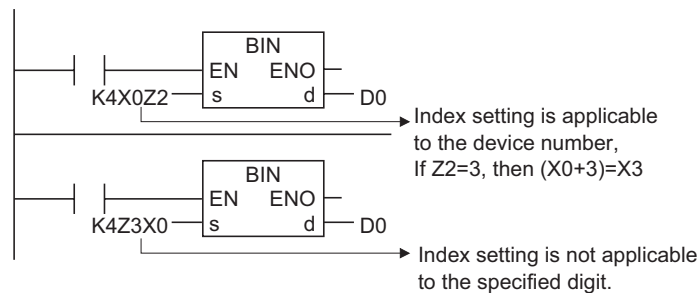
(b) An index setting is applicable to both start I/O numbers of the intelligent function module and buffer memory addresses for intelligent function module devices[3].

```
          ┌─────────┐
          │   MOV   │
     ┌┤ ├─┤EN    ENO├
     │   │s       d ├─ D0
 U10Z1\G0Z2└─────────┘
                        ──► If Z1=2 and Z2=8,
                             then U(10+2)\G(0+8)=U12\G8
```
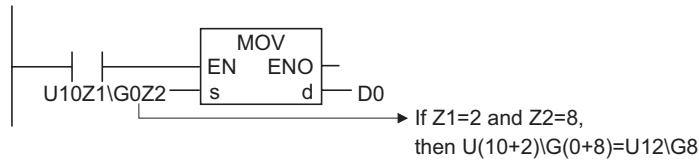
(c) An index setting is applicable to both network numbers and device numbers for link direct devices[3].

```
          ┌─────────┐
          │   MOV   │
     ┌┤ ├─┤EN    ENO├
     │   │s       d ├─ D0
 J1Z1\K4X0Z2└─────────┘
                        ──► If Z1=2 and Z2=8,
                             then J(1+2)\K4X(0+8)=J3\K4X8
```
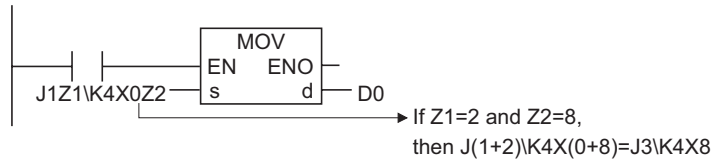
(d) An index setting is applicable to both start I/O numbers of the CPU module and CPU shared memory addresses for multiple CPU area devices[4].

```
          ┌─────────┐
          │   MOV   │
     ┌┤ ├─┤EN    ENO├
     │   │s       d ├─ D0
 U3E0Z1\G0Z2└─────────┘
```
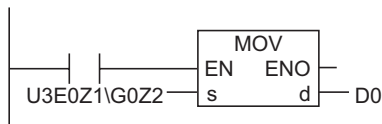
*3: For intelligent function module devices and link direct devices, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used.

*4: For multiple CPU area devices, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module to be used.

(e) A 32-bit index setting is applicable to extended data register (D) and extended link register (W)
(for Universal model QCPU (excluding Q00UJCPU), and LCPU)
When applying an index setting to extended data registers (D) or extended link registers (W), it can be applied in 32-bit range as applying an index setting to file registers (ZR) in the following two methods.

• Specify a range of index registers used for a 32-bit index setting.

• Specify a 32-bit index setting using 'ZZ'.

⊠POINT

32-bit index settings using 'ZZ' can be used for the following CPU modules only.
• QnU(D)(H)CPU with a serial number whose first five digits are '10042' or higher (excluding Q00UJCPU)
• QnUDE(H)CPU
• LCPU

(6) Precautions

(a) Applying the index setting within the FOR to NEXT instruction loop
The pulses can be output by using edge relays (V) within the FOR to NEXT instruction loop.
Note that the pulses cannot be output by the PLS, PLF or pulse (□ P) instruction.

[When using an edge relay]

(M0Z1 pulse is output normally.)

```
SM400          MOV
 ─┤ ├─────┤EN    ENO├
          0─┤s      d├─ Z1

               FOR
         ─────┤EN    ENO├
         10─┤n

X0Z1          EGP              OUT
 ─┤ ├─────┤EN    ENO├─────┤EN    ENO├
               d├─ V0Z1         d├─ M0Z1

SM400          INC
 ─┤ ├─────┤EN    ENO├
               d├─ Z1

               NEXT
         ─────┤EN    ENO├
```

[When not using an edge relay]

(M0Z1 pulse is not output normally.)

```
SM400          MOV
 ─┤ ├─────┤EN    ENO├
          0─┤s      d├─ Z1

               FOR
         ─────┤EN    ENO├
         10─┤n

X0Z1          PLS
 ─┤ ├─────┤EN    ENO├
               d├─ M0Z1

SM400          INC
 ─┤ ├─────┤EN    ENO├
               d├─ Z1

               NEXT
         ─────┤EN    ENO├
```
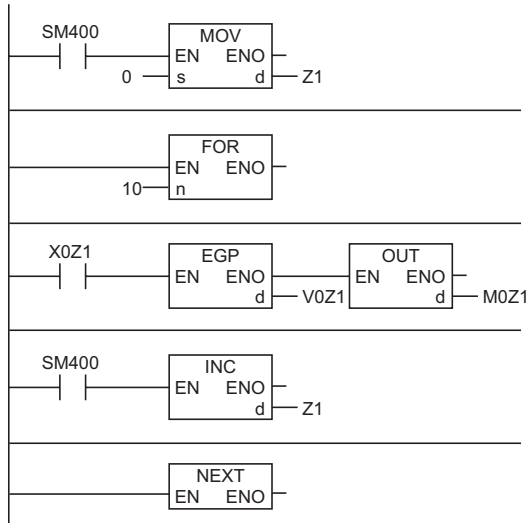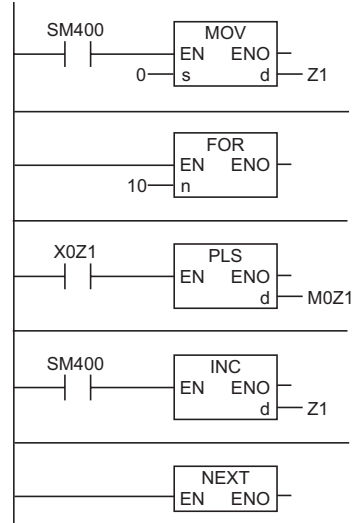
Remark •

The ON/OFF information of X0Z1 is stored to the edge relay V0Z1.
For example, the ON/OFF data of X0 is stored to V0 and the ON/OFF data of X1 is stored to V1.

• •

⊠ **POINT** ───────────────────────────────────────

Z0 and Z1 cannot be used when labels are used for array indexes within the FOR to NEXT instruction loop.

(b) Applying the index setting in the CALL instruction
The pulse can be output by using edge relays (V) with the CALL instruction. Note that the pulse cannot be output by the PLS, PLF or pulse (☐ P) instruction.
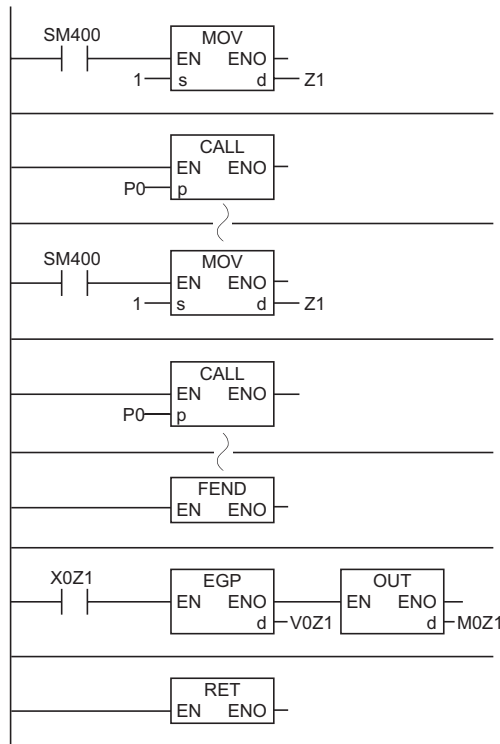
[When using an edge relay]   [When not using an edge relay]
(M0Z1 pulse is output normally.)   (M0Z1 pulse is not output normally.)
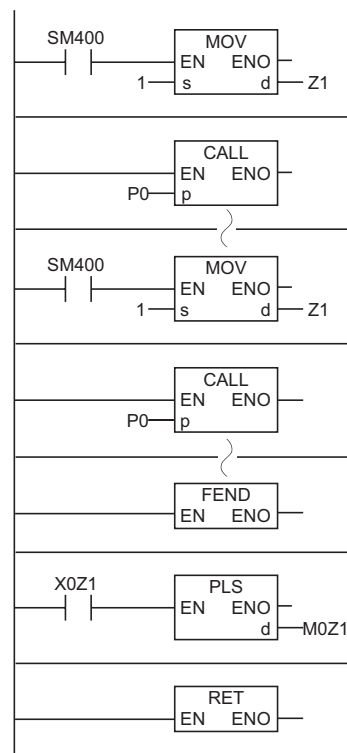
(c) Device range check when the index setting is applied

1) For Basic model QCPU, High Performance model QCPU, and FXCPU
The device range is not checked when the index setting is applied.
For Basic model QCPU and High Performance model QCPU, if the result of the index setting exceeds the device range specified by a user, an error does not occur and the data are written to other devices. (Note that if the result of the index setting exceeds the device range specified by a user and the data are written to devices for the system, an error occurs. (Error code: 1103))
For FXCPU, an operation error occurs. (Error code: 6706)
Create a program with caution when applying the index setting.

2) For Universal model QCPU, and LCPU
The device range is checked when the index setting is applied.
By changing the settings of the PLC parameter, the device range is not checked.

(d) Switching between 16-bit and 32-bit range of the index setting
When switching between 16-bit and 32-bit range, check the positions of the index setting in the program.
Since the specified index register (Zn) and next index register (Zn+1) are used for index setting in 32-bit range, make sure not to overlap index registers being used.

4.7 Libraries

A library is an aggregation of data including POUs, global labels, and structures organized in a single file to be utilized in multiple projects.

The followings are the advantages of using libraries.

- Data in library files can be utilized in multiple projects by installing them to each project.
- Since library data can be created according to the functions of components, data to be reused can be easily confirmed.
- If components registered in a library are modified, the modification is applied to projects that use the modified data.

The following figure shows the data flow when using library components in a project.
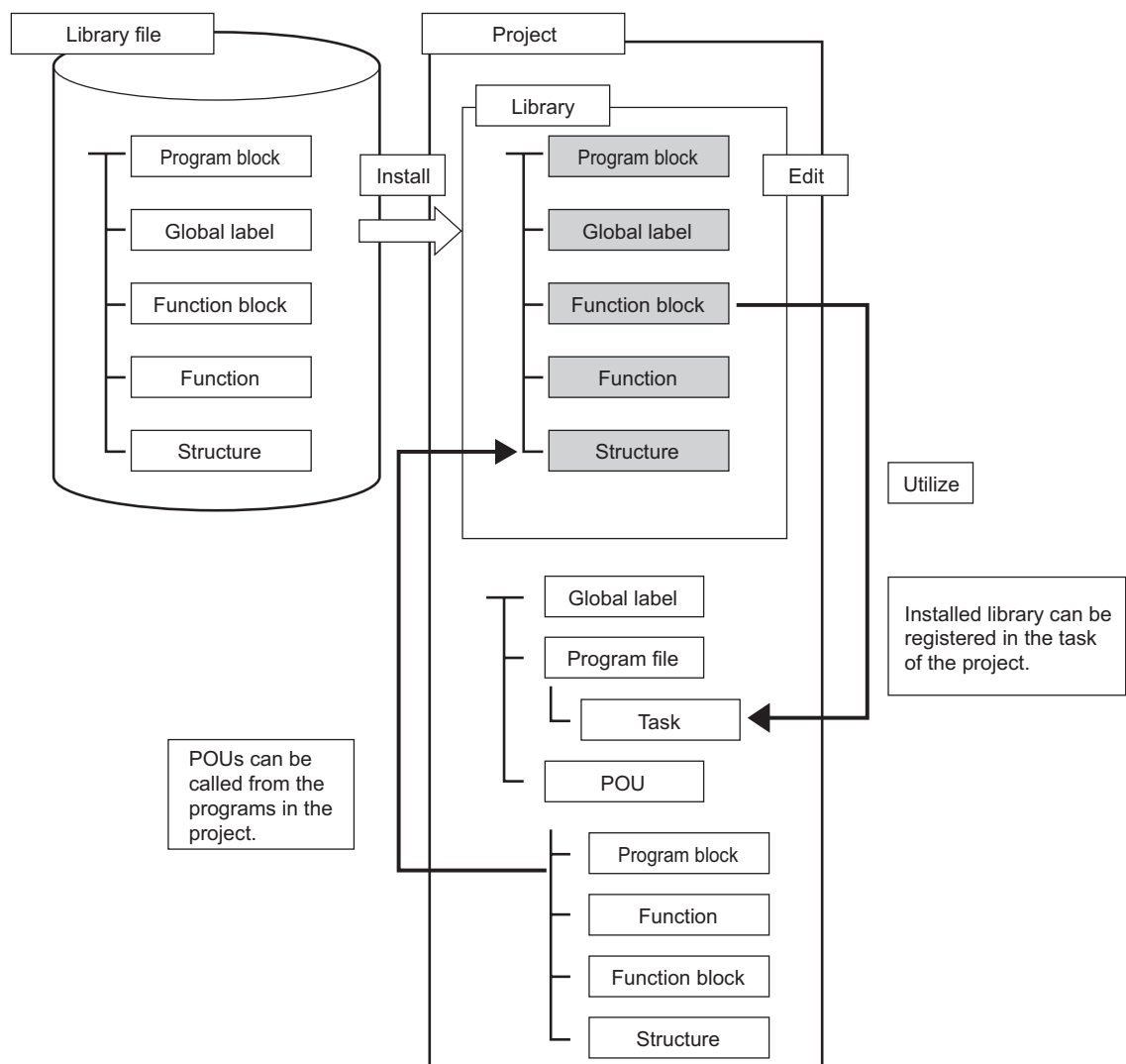
4.7.1 User libraries

A user library is a library for storing created structures, global labels, POUs, and other data that can be used in other projects.

(1) Composition of a user library

The following table shows data that can be registered in a user library.

Table 4.7.1-1 Composition of a user library

Name	Description
Structure	Stores definitions of structures used in POU folders of library, or definitions of structures used in programs of a project.
Global label	Stores definitions of global labels used in POU folders of library.
POU	Stores program blocks, functions, and function blocks that can be used as libraries.

4

PROGRAM
CONFIGURATION

4.8 Precautions on Assigning a Name

This section explains the conditions for assigning a name to a label, function block instance or structure label.

- Conditions

(1) Specify a name within 32 characters.

(2) Do not use reserved words.
For reserved words, refer to the following section.

☞ Appendix 2 Character Strings that cannot be Used in Label Names and Data Names

(3) Use alphanumeric and underscores (_).

(4) Do not use an underscore at the end of the name.
Do not use two or more underscores in succession.

(5) Do not use spaces.

(6) Do not use a number for the initial character.

(7) Constants cannot be used.
(An identifier that begins with 'H' or 'h' and an expression where a hexadecimal (0 to F) immediately follows 'H' or 'h' (maximum 9 digits including 'H' or 'h' (excluding 0 that immediately follows 'H' or 'h')) are also treated as a constant. (Example: 'hab0'))

(8) Elementary data type names cannot be used.

(9) Function/FB names cannot be used.

5 WRITING PROGRAMS

1
OVERVIEW

2
STRUCTURED DESIGN OF SEQUENCE PROGRAMS

3
PROCEDURE FOR CREATING PROGRAMS

4
PROGRAM CONFIGURATION

5
WRITING PROGRAMS

A
APPENDIX

I
INDEX

5.1 ST Language

The ST language is a text language with a similar grammatical structure to the C language.

Controls such as conditional judgement and repetition process written in syntaxes can be described.

This language is suitable for programming complicated processes that cannot be easily described by a graphic language (structured ladder language).

5.1.1 Standard format

```
intV2 := ABS( intV1);                                          ← [Assignment syntax]
IF M1 THEN
    btn01 :=TRUE;
ELSE                                                           ← [syntaxes]
    btn01 :=FALSE;
END_IF;

Output_ENO :=ENEG(btn01, Input1);                              ← [Calling the function]

LadderFBInstance(input1:= bool1,Input2:= bool2, Input3:=bool3);  ← [Calling the function block]
(* user fuction block  *)                                      ← [Comment]
```
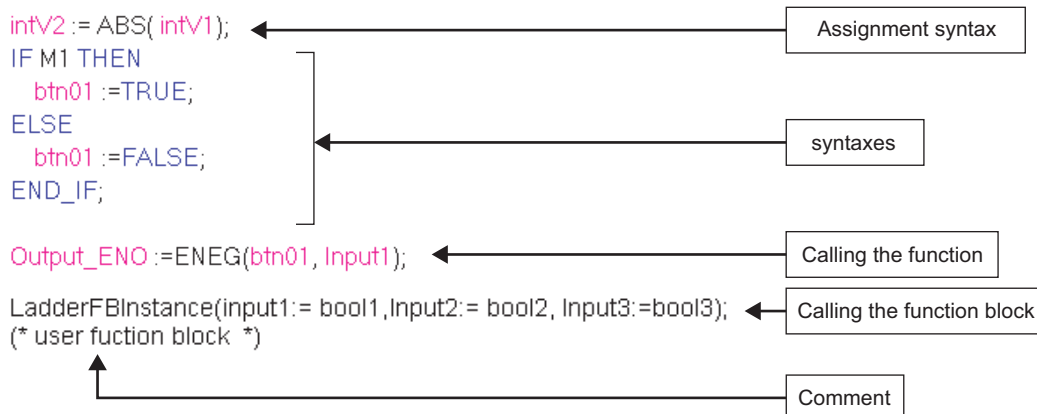
Operators and syntaxes are used for programming in the ST language.

Syntaxes must end with ';'.

```
intV1 :=0;        → [Enter ';' at the end.]
intV2 :=2;
```

Spaces, tabs, and line feeds can be inserted anywhere between a keyword and an identifier.

```
intV1 := 0;       [Space]
intV2:=           [Tab]
2;                [Line feed]
```
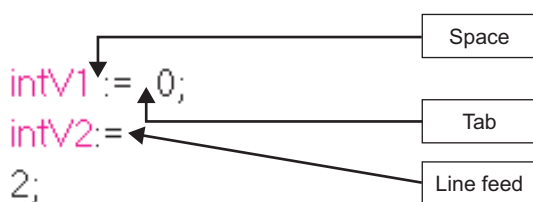
Comments can be inserted in a program. Describe '(*' in front of a comment and '*)' in back of a comment.

```
intV1 :=0;
(*Substitution*)    ← [Comment]
intV2 :=2;
```

5.1.2 Operators in ST language

The following table shows the operators used in the ST language and their priorities.

Table 5.1.2-1 Operators in the ST language

Operator	Description	Example	Priority
( )	Parenthesized expression	(1+2)*(3+4)	Highest
Function ( )	Function (Parameter list)	ADD_E(bo01, in01, in02, in03)	
**	Exponentiation	re01:= 2.0 ** 4.4	
NOT	Inverted bit value	NOT bo01	
*	Multiplication	3 * 4	
/	Division	12 / 3	
MOD	Modulus operation	13 MOD 3	
+	Addition	in01 + in02	
-	Subtraction	in01 - in02	
<, >, <=, =>	Comparison	in01 < in02	
=	Equality	in01 = in02	
<>	Inequality	in01 <> in02	
AND, &	Logical AND	bo01 & bo02	
XOR	Exclusive OR	bo01 XOR bo02	
OR	Logical OR	bo01 OR bo02	Lowest

If a syntax includes multiple operators with a same priority, the operation is performed from the leftmost operator.

The following table shows the operators, applicable data types, and operation result data types.

Table 5.1.2-2 Data types used in operators

Operator	Applicable data type	Operation result data type
*, /, +, -	ANY_NUM	ANY_NUM
<, >, <=, >=, =, <>	ANY_SIMPLE	Bit
MOD	ANY_INT	ANY_INT
AND, &, XOR, OR, NOT	ANY_BIT	ANY_BIT
**	ANY_REAL (Base) ANY_NUM (Exponent)	ANY_REAL

5.1.3 Syntaxes in ST language

The following table shows the syntaxes that can be used in the ST language.

Table 5.1.3-1 Syntaxes in the ST language

Type of syntax	Description
Assignment syntax	Assignment syntax
Conditional syntax	IF THEN conditional syntax, IF ELSE conditional syntax, and IF ELSIF conditional syntax
	CASE conditional syntax
Iteration syntax	FOR DO syntax
	WHILE DO syntax
	REPEAT UNTIL syntax
Other control syntax	RETURN syntax
	EXIT syntax

(1) Assignment syntax

    (b) Format

        <Left side> := <Right side>;

    (c) Description

        The assignment syntax 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 data type of the left side need to obtain the same data when using the assignment syntax.

    (d) Example

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

(2) IF THEN conditional syntax

    (a) Format

```
IF <Boolean expression> THEN
<Syntax ...>;
END_IF;
```

    (b) Description

The syntax is executed when the value of Boolean expression (conditional formula) is TRUE. The syntax 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.

    (c) Example

```
IF bool1 THEN
    intV1:=intV1 +1;
END_IF;
```

(3) IF ...ELSE conditional syntax

    (a) Format

```
IF <Boolean expression> THEN
<Syntax 1 ...>;
ELSE
<Syntax 2 ...>;
END_IF;
```

    (b) Description

Syntax 1 is executed when the value of Boolean expression (conditional formula) is TRUE.
Syntax 2 is executed when the value of Boolean expression is FALSE.

    (c) Example

```
IF bool1 THEN
    intV3 :=intV3 +1;
ELSE
    intV4 :=intV4 +1;
END_IF;
```

(4) IF ...ELSIF conditional syntax

    (a) Format

```
IF <Boolean expression 1> THEN
<Syntax 1 ...>;
ELSIF <Boolean expression 2> THEN
<Syntax 2 ...>;
ELSIF <Boolean expression 3> THEN
<Syntax 3 ...>;
END_IF;
```

    (b) Description

Syntax 1 is executed when the value of Boolean expression (conditional formula) 1 is TRUE. Syntax 2 is executed when the value of Boolean expression 1 is FALSE and the value of Boolean expression 2 is TRUE.

Syntax 3 is executed when the value of Boolean expression 1 and 2 are FALSE and the value of Boolean expression 3 is TRUE.

    (c) Example

```
IF bool1 THEN
   intV1 : = intV1 +1;
ELSIF bool2 THEN
   intV2 : = intV2 +2;
ELSIF bool3 THEN
   intV3 : = intV3 +3;
END_IF;
```

(5) CASE conditional syntax

    (a) Format

```
CASE <Integer expression> OF
<Integer selection 1> : <Syntax 1 ...>;
<Integer selection 2> : <Syntax 2 ...>;
.
.
.
<Integer selection n> : <Syntax n ...>;
ELSE
<Syntax n+1 ...>;
END_CASE;
```

    (b) Description

The result of the CASE conditional expression is returned as an integer value. The CASE conditional syntax is used to execute a selection syntax by a single integer value or an integer value as the result of a complicated expression.
When the syntax that has the integer selection value that matches with the value of integer expression is executed, and if no integer selection value is matched with the expression value, the syntax that follows the ELSE syntax is executed.

    (c) Example

```
CASE intV1 OF
    1:bool1 :=TRUE;
    2:bool2 :=TRUE;
ELSE
    intV1 :=intV1 +1;
END_CASE;
```

(6) FOR...DO syntax

    (a) Format

```
FOR <Repeat variable initialization>
TO <Last value>
BY <Incremental expression> DO
<Syntax ...>;
END_FOR;
```

    (b) Description

The FOR...DO syntax repeats the execution of several syntaxes according to the value of a repeat variable.

    (c) Example

```
FOR intV1 := 0
    TO 30
    BY 1 DO
    intV3 :=intV1 +1;
END_FOR;
```

(7) WHILE...DO syntax

    (a) Format

```
WHILE <Boolean expression> DO
<Syntax ...>;
END_WHILE;
```

    (b) Description

The WHILE...DO syntax executes one or more syntaxes while the value of Boolean expression (conditional formula) is TRUE.

The Boolean expression is evaluated before the execution of the syntax. If the value of Boolean expression is FALSE, the syntax in the WHILE...DO syntax is not executed.

Since a return result of the Boolean expression in the WHILE syntax requires only TRUE or FALSE, any Boolean expression that can be specified in the IF conditional syntax can be used.

    (c) Example

```
WHILE intV1 := 30 DO
    intV1 :=intV1 +1;
END_WHILE;
```

(8) REPEAT...UNTIL syntax

    (a) Format

```
REPEAT
<Syntax ...>;
UNTIL <Boolean expression>
END_REPEAT;
```

    (b) Description

The REPEAT...UNTIL syntax executes one or more syntaxes while the value of Boolean expression (conditional formula) is FALSE.

The Boolean expression is evaluated after the execution of the syntax. If the value of Boolean expression is TRUE, the syntaxes in the REPEAT...UNTIL syntax are not executed.

Since a return result of the Boolean expression in the REPEAT syntax requires only TRUE or FALSE, any Boolean expression that can be specified in the IF conditional syntax can be used.

    (c) Example

```
REPEAT
    intV1 :=intV1 +1;
    UNTIL intV1 = 30
END_REPEAT;
```

(9) RETURN syntax

    (a) Format

        RETURN;

    (b) Description

        The RETURN syntax is used to end a program in a middle of the process.
        When the RETURN syntax is used in a program, the process jumps from the RETURN syntax execution step to the last line of the program, ignoring all the remaining steps after the RETURN syntax.

    (c) Example

```
IF bool1 THEN
    RETURN;
END_IF;
```

(10) EXIT syntax

    (a) Format

        EXIT;

    (b) Description

        The EXIT syntax is used only in iteration syntaxes to end the iteration syntax in a middle of the process.
        When the EXIT syntax is reached during the execution of the iteration loop, the iteration loop process after the EXIT syntax is not executed. The process continues from the line after the one where the iteration syntax is ended.

    (c) Example

```
FOR intV1 := 0
    TO 10
    BY 1 DO
    IF intV1 > 10 THEN
        EXIT;
    END_IF;
END_FOR:
```

5.1.4 Calling functions in ST language

The following description is used to call a function in the ST language.

Function name (Variable1, Variable2, ...);

Enclose the arguments by '( )' after the function name.

When using multiple variables, delimit them by ','.

The execution result of the function is stored by assigning the result to the variables.

1) Calling a function with one input variable (Example: ABS)

> Output1 := ABS(Input1);

2) Calling a function with three input variables (Example: MAX)

> Output1 := MAX(Input1, Input2, Input3);

5.1.5 Calling function blocks in ST language

The following description is used to call a function block in the ST language.

> Instance name(Input variable1:= Variable1, ... Output variable1: = Variable2, ...);

Enclose the assignment syntaxes that assigns variables to the input variable and output variable by '( )' after the instance name.

When using multiple variables, delimit assignment syntaxes by ',' (comma).

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.

1) Calling a function block with one input variable and one output variable

> FB definition
>
> FB Name: FBADD
>
> FB instance name: FBADD1
>
> Input variable1: IN1
>
> Output variable1: OUT1

The following is the description to call the function block above.

> FBADD1(IN1:=Input1);
>
> Output1:=FBADD1.OUT1;

2) Calling a function block with three input variables and two output variables

> FB definition
>
> FB Name: FBADD
>
> FB instance name: FBADD1
>
> Input variable1: IN1
>
> Input variable2: IN2
>
> Input variable3: IN3
>
> Output variable1: OUT1
>
> Output variable2: OUT2

The following is the description to call the function block above.

> FBADD1(IN1:=Input1, IN2:=Input2, IN3:= Input3);
>
> Output1:=FBADD1.OUT1;
>
> Output2:=FBADD1.OUT2;

5.2 Structured Ladder Language

The structured ladder language is a graphic language for writing programs using ladder symbols such as contacts, coils, functions, and function blocks.

Write programs on the assumption that the power is supplied from the baseline at the left edge of the editor to the connected ladders.

5.2.1 Standard format
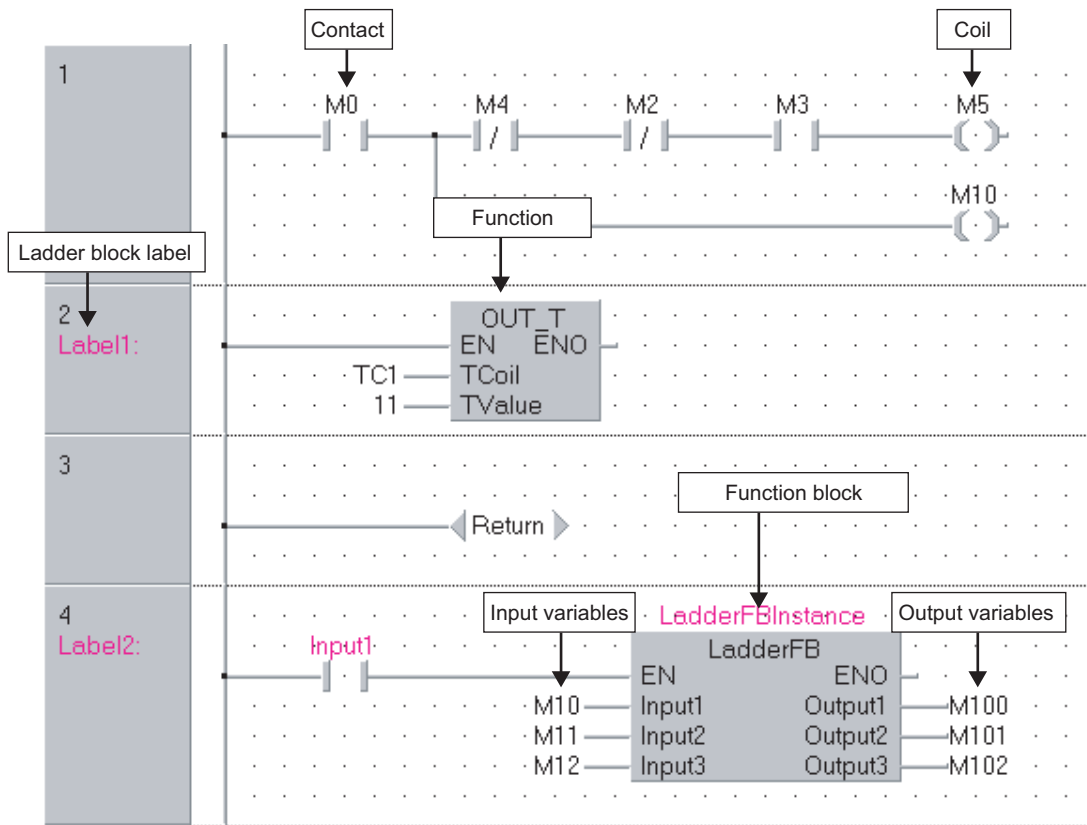
In the structured ladder language, units of ladder blocks are used for programming.

The operation order in a ladder block is from the baseline to the right and from the top to the bottom.

5.2.2 Ladder symbols in structured ladder language

The following table shows the ladder symbols that can be used in the structured ladder language.

For details, refer to the following manual.

☞ MELSEC-Q/L Structured Programming Manual (Common Instructions)

Table 5.2.2-1 Ladder symbols in the structured ladder language (1/2)

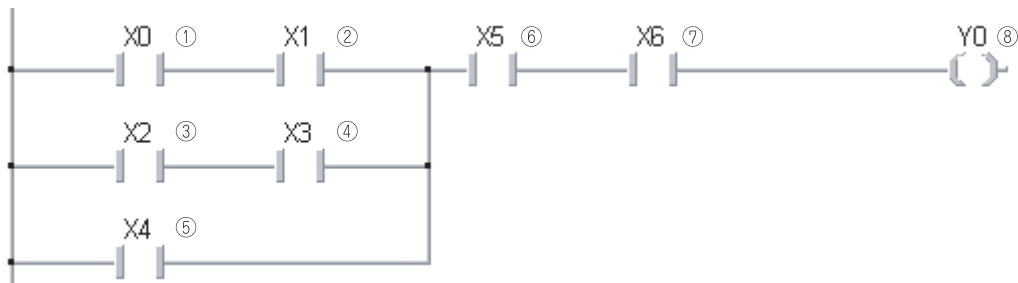| Element	Ladder symbol	Description
Normal [1]	-| |-	Turns ON when a specified device or label is ON.
Negation [1]	-| / |-	Turns OFF when a specified device or label is OFF.
Rising edge [1],[2]	-| ↑ |-	Turns ON at the rising edge (OFF to ON) of a specified device or label.
Falling edge [1],[2]	-| ↓ |-	Turns ON at the falling edge (ON to OFF) of a specified device or label.
Negated rising edge [1],[2]	-| ↟ |-	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 [1],[2]	-| ↡ |-	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.
Normal	——( )—	Outputs the operation result to a specified device or label.
Negation	——(/)—	A specified device or label turns ON when the operation result turns OFF.
Set	——(S)—	A specified device or label turns ON when the operation result turns ON. Once the device or label turns ON, it remains ON even when the operation result turns OFF.
Reset	——(R)—	A specified device or label turns OFF when the operation result turns ON. If the operation result is OFF, the status of the device or label does not change.

[1] A contact performs an AND operation or OR operation according to the connection of a ladder block and reflects in the operation result.
   • For a series connection, it performs an AND operation with the operation result up to that point, and takes the resulting value as the operation result.
   • For a parallel connection, it performs an OR operation with the operation result up to that point, and takes the resulting value as the operation result.

Series-connection contact

Parallel-connection contact
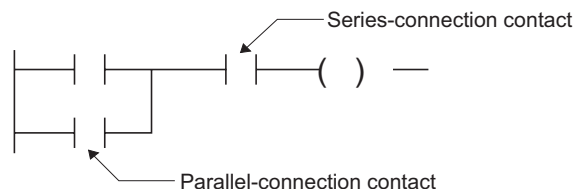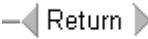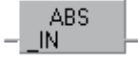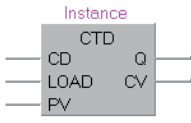
[2] Supported with GX Works2 Version 1.15R or later.
   For the confirmation method of the version of GX Works2, refer to the following manual.

   ☞ GX Works2 Version1 Operating Manual (Common)

Table 5.2.2-1 Ladder symbols in the structured ladder language (2/2)

Element	Network element	Description
Jump	—▶Label	Pointer branch instruction Unconditionally executes the program at the specified pointer number in the same program file.
Return	—◀ Return ▶	Indicates the end of a subroutine program. Returns the step to the next step after the instruction which called the subroutine program.
Function	ABS _IN	Executes a function.
Function block	Instance CTD CD Q LOAD CV PV	Executes a function block.
Function argument input	? —	Inputs an argument to a function or function block.
Function return value output	—?	Outputs the return value from a function or function block.
Function inverted argument input	? —◆	Inverts and inputs an argument to a function or function block.
Function inverted return value output	◆— ?	Inverts the return value from a function or function block and outputs it.

MEMO

APPENDIX

1 OVERVIEW

2 STRUCTURED DESIGN OF SEQUENCE PROGRAMS

3 PROCEDURE FOR CREATING PROGRAMS

4 PROGRAM CONFIGURATION

5 WRITING PROGRAMS

A APPENDIX

INDEX

Appendix 1 Correspondence between Generic Data Types and Devices

The following table shows the correspondence between generic data types and devices.

Table App. 1-1 Correspondence between generic data types and devices

Device				
Classification	**Type**	**Device name**	**Device symbol**	
Internal user device	Bit device	Input	X	
		Output	Y	
		Internal relay	M	
		Latch relay	L	
		Annunciator	F	
		Edge relay	V	
		Step relay	S	
		Link special relay	SB	
		Link relay	B	
		Timer contact[*1]	TS	
		Timer coil[*1]	TC	
		Retentive timer contact[*1]	STS	
		Retentive timer coil[*1]	STC	
		Counter contact[*1]	CS	
		Counter coil	CC	
	Word device	Timer current value	T or TN[*1]	
		Retentive timer current value	ST or STN[*1]	
		Counter current value	C or CN[*1]	
		Data register	D	
		Link register	W	
		Link special register	SW	
Internal system device	Bit device	Function input	FX	
		Function output	FY	
		Special relay	SM	
	Word device	Function register	FD	
		Special register	SD	

*1: Can be used for digit specification.
*2: Can be used for bit specification.

	Generic data type												
	ANY											ANY	
	ANY_SIMPLE												
	ANY_BIT			ANY_NUM				Time	String	Array	Structure		
				ANY_INT		ANY_REAL							
	Bit	Word (unsigned)/ 16-bit string	Double word (unsigned)/ 32-bit string	Word (signed)	Double word (signed)	Single-precision real	Double-precision real	Time	String	Array	Structure	ANY16	ANY32
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	×	×	×	×	×	×	×	×	×	×	×	×
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	×	○	×	○	×	×	×	×	×	×	×	○	×
	×	○	×	○	×	×	×	×	×	×	×	○	×
	×	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	–	–	–	–	–	–	–	–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–	–	–	–
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	–	–	–	–	–	×	×	×	×	×	×		–
	○*2	○	×	○	×	×	×	×	×	×	×	○	×

A

APPENDIX

Device				
Classification	**Type**	**Device name**	**Device symbol**	
Link direct device	Bit device	Link input	Jn\X	
		Link output	Jn\Y	
		Link relay	Jn\B	
		Link special relay	Jn\SB	
	Word device	Link register	Jn\W	
		Link special register	Jn\SW	
Intelligent function module device	Word device	Intelligent function module device	Un\G	
Index register	Word device	Index register	Z	
File register	Word device	File register	R or ZR	
Nesting	−	Nesting	N	
Pointer	−	Pointer	P	
		Interrupt pointer	I	
Constant	−	−	K, H	
			E	
String constant	−	−	'Character string' or "Character string"	

*1: Can be used for digit specification.
*2: Can be used for bit specification.

Appendix 1 Correspondence between Generic Data Types and Devices

	Generic data type												
	ANY											ANY	
	ANY_SIMPLE									Array	Structure		
	ANY_BIT			ANY_NUM									
				ANY_INT		ANY_REAL							
	Bit	Word (unsigned)/ 16-bit string	Double word (unsigned)/ 32-bit string	Word (signed)	Double word (signed)	Single-precision real	Double-precision real	Time	String	Array	Structure	ANY16	ANY32
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○	○*1	○*1	○*1	○*1	×	×	×	×	×	×	○*1	○*1
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	×	○	×	○	×	×	×	×	×	×	×	○	×
	○*2	○	×	○	×	×	×	×	×	×	×	○	×
	–	–	–	–	–	–	–	–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–	–	–	–
	–	–	–	–	–	–	–	–	–	–	–	–	–
	○	○	○	○	○	○	×	×	×	×	×	○	○
	×	×	×	×	×	○	○	×	×	×	×	×	×
	×	×	×	×	×	×	×	×	○	×	×	×	×

Appendix 2 Character Strings that cannot be Used in Label Names and Data Names

Character strings used for application function names, common instruction names, special instruction names, and instruction words are called reserved words.

These reserved words cannot be used for label names or data names. If the character string defined as a reserved word is used for a label name or data name, an error occurs during registration or compilation.

The following table shows character strings that cannot be used for label names or data names.

Table App. 2-1 Character strings that cannot be used for label names and data names (1/2)

Category	Character string
Class identifier	VAR, VAR_RETAIN, VAR_ACCESS, VAR_CONSTANT, VAR_CONSTANT_RETAIN, VAR_INPUT, VAR_INPUT_RETAIN, VAR_OUTPUT, VAR_OUTPUT_RETAIN, VAR_IN_OUT, VAR_IN_EXT, VAR_EXTERNAL, VAR_EXTERNAL_CONSTANT, VAR_EXTERNAL_CONSTANT_RETAIN, VAR_EXTERNAL_RETAIN, VAR_GLOBAL, VAR_GLOBAL_CONSTANT, VAR_GLOBAL_CONSTANT_RETAIN, VAR_GLOBAL_RETAIN
Data type	BOOL, BYTE, INT, SINT, DINT, LINT, UINT, USINT, UDINT, ULINT, WORD, DWORD, LWORD, ARRAY, REAL, LREAL, TIME, STRING
Data type hierarchy	ANY, ANY_NUM, ANY_BIT, ANY_REAL, ANY_INT, ANY_DATE, ANY_SIMPLE, ANY16, ANY32
Device name	X, Y, D, M, T, B, C, F, L, P, V, Z, W, I, N, U, J, K, H, E, A, SD, SM, SW, SB, FX, FY, DX, DY, FD, TR, BL, SG, VD, ZR, ZZ
Character string recognized as device (Device name + Numeral)	Such as X0
ST operator	NOT, MOD
IL operator	LD, LDN, ST, STN, S, S1, R, R1, AND, ANDN, OR, ORN, XOR, XORN, ADD, SUB, MUL, DIV, GT, GE, EQ, NE, LE, LT, JMP, JMPC, JMPCN, CAL, CALC, CALCN, RET, RETC, RETCN, LDI, LDP, LDPI, LDF, LDFI, ANI, ANDP, ANDPI, ANDF, ANDFI, ANB, ORI, ORP, ORPI, ORF, ORFI, ORB, MPS, MRD, MPP, INV, MEP, MEF, EGP, EGF, OUT(H), SET, RST, PLS, PLF, FF, DELTA(P), SFT(P), MC, MCR, STOP, PAGE, NOP, NOPLF
Application instruction in GX Works2	Application instructions such as DMOD, PCHK, INC(P) ☞ MELSEC-Q/L Programming Manual (Common Instructions), MELSEC-Q/L Structured Programming Manual (Common Instructions) ☞ FXCPU Structured Programming Manual (Basic & Applied Instruction), FXCPU Structured Programming Manual (Application Functions)
SFC instruction	SFCP, SFCPEND, BLOCK, BEND, TRANL, TRANO, TRANA, TRANC, TRANCA, TRANOA, SEND, TRANOC, TRANOCA, TRANCO, TRANCOC, STEPN, STEPD, STEPSC, STEPSE, STEPST, STEPR, STEPC, STEPG, STEPI, STEPID, STEPISC, STEPISE, STEPIST, STEPIR, TRANJ, TRANOJ, TRANOCJ, TRANCJ, TRANCOJ, TRANCOCJ
ST code body	RETURN, IF, THEN, ELSE, ELSIF, END_IF, CASE, OF, END_CASE, FOR, TO, BY, DO, END_FOR, WHILE, END_WHILE, REPEAT, UNTIL, END_REPEAT, EXIT, TYPE, END_TYPE, STRUCT, END_STRUCT, RETAIN, VAR_ACCESS, END_VAR, FUNCTION, END_FUNCTION, FUCTION_BLOCK, END_FUCTION_BLOCK, STEP, INITIAL_STEP, END_STEP, TRANSITION, END_TRANSITION, FROM, TO, UNTILWHILE
Standard function name	Function names in application functions such as AND_E, NOT_E

Table App. 2-1 Character strings that cannot be used for label names and data names (2/2)

Category	Character string
Standard function block name	Function block names in application functions such as CTD, CTU
Symbol	/, \, *, ?, <, >, |, ", :, [,] , ;, , , =, +, %, ', ~, @, {, }, !, #, $, &
Date and time literal	DATE, DATE_AND_TIME, DT, TIME, TIME_OF_DAY, TOD
Others	ACTION, END_ACTION, CONFIGURATION, END_CONFIGURATION, CONSTANT, F_EDGE, R_EDGE, AT, PROGRAM, WITH, END_PROGRAM, TRUE, FALSE, READ_ONLY, READ_WRITE, RESOURCE, END_RESOURCE, ON, TASK, EN, ENO, BODY_CCE, BODY_FBD, BODY_IL, BODY_LD, BODY_SFC, BODY_ST, END_BODY, END_PARAMETER_SECTION, PARAM_FILE_PATH, PARAMETER_SECTION, SINGLE, TRUE, FALSE, RETAIN, INTERVAL, L, P
String that starts with K1 to K8	Such as K1AAA
Address	Such as %IX0
Statement in ladder language	;FB BLK START, ;FB START, ;FB END, ;FB BLK END, ;FB IN, ;FB OUT, ;FB_NAME;,INSTANCE_NAME, ;FB, ;INSTANCE
Common instruction	Such as MOV
Windows reserved word	COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, LPT9, AUX, CON, PRN, NUL

(1) Precautions on using labels

- For label names and instance names, the same name as the one used for data name of task, structure, POU and the like cannot be used.

- A space cannot be used.

- A numeral cannot be used in the first character of a label name.

- A label name is not case-sensitive. An error may occur at compilation.

- For structured ladder or ST programs, a same label name can be used for a global label and a local label according to the following option setting of GX Works2[1].

  *1: Check the "Use the same label name in global label and local label" item under [Tool] ⇒ [Options] ⇒ "Compile" ⇒ "Structured Ladder/ST" ⇒ "Compile Condition1".

Appendix 3 Recreating Ladder Programs

This section provides an example of creating a structured program same as the program created in the ladder programming language using GX Works2.

Appendix 3.1 Procedure for creating a structured program

The following explains the basic procedure for creating a structured program based on the program created in the ladder programming language.

(1) Replacing devices with labels

Procedure
Labels include global labels and local labels.
Determine the type of labels (global label or local label) to replace devices.

(2) Setting labels

Procedure
Global labels and local labels to be used in the program must be defined.
Define all labels to be used in the program.

(3) Creating a program
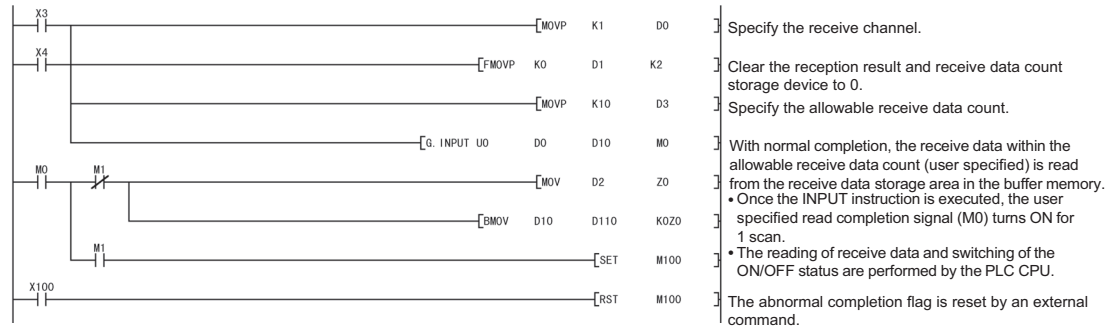
Procedure
Create a structured program in the programming language to be used.

Appendix 3.2 Example of creating a structured program

This section shows an example of creating a sequence program same as the program created in GX Developer using GX Works2.

The following examples explain the method for creating a structured program same as the data receive program for a Q-compatible serial communication module, using the structured ladder and ST languages.

The following shows the original program.

X3	──[MOVP K1 D0]──	Specify the receive channel.
X4	──[FMOVP K0 D1 K2]──	Clear the reception result and receive data count storage device to 0.
	──[MOVP K10 D3]──	Specify the allowable receive data count.
	──[G. INPUT U0 D0 D10 M0]──	With normal completion, the receive data within the allowable receive data count (user specified) is read from the receive data storage area in the buffer memory.
M0 M1	──[MOV D2 Z0]──	• Once the INPUT instruction is executed, the user specified read completion signal (M0) turns ON for 1 scan.
	──[BMOV D10 D110 K0Z0]──	• The reading of receive data and switching of the ON/OFF status are performed by the PLC CPU.
M1	──[SET M100]──	
X100	──[RST M100]──	The abnormal completion flag is reset by an external command.

(1) Replacing devices with labels

Replace devices of the original program with labels.
Replace input/output devices with global labels. For devices such as internal relays, replace them with local labels.

Table App. 3.2-1 Examples of replacement from devices to labels

Device	Purpose		Label	
			Data type	Label name
X3	CH1 reception data read request		Bit	CH1ReadRequest
X4	CH1 reception abnormal detection		Bit	CH1AbnormalDetection
D0	Control data	Reception channel	Word (unsigned)/16-bit string [0] to [3]	ControlData
D1		Reception result		
D2		Number of reception data		
D3		Number of allowable reception data		
D10 to D109	Reception data		Word (unsigned)/16-bit string [0] to [99]	RecieveData
D110 to D209	Reception data storage area		Word (unsigned)/16-bit string [0] to [99]	Data
M0	Data reception completion flag	Completion flag	Bit [0] to [1]	Completion
M1		Status flag at completion		
M100	Abnormal completion flag		Bit	AbnormalCompletion
X100	Abnormal completion flag reset command		Bit	ResetAbnormalCompletion

(2) Setting labels

Set global labels and local labels.

  • Setting examples of global labels

	Class		Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	▼	CH1 ReadRequest1	Bit	...		X3	%IX3
2	VAR_GLOBAL	▼	CH1 AbnormalDetection	Bit	...		X4	%IX4
3	VAR_GLOBAL	▼	ResetAbnormalCompletion	Bit	...		X100	%IX256

  • Setting examples of local labels[1]

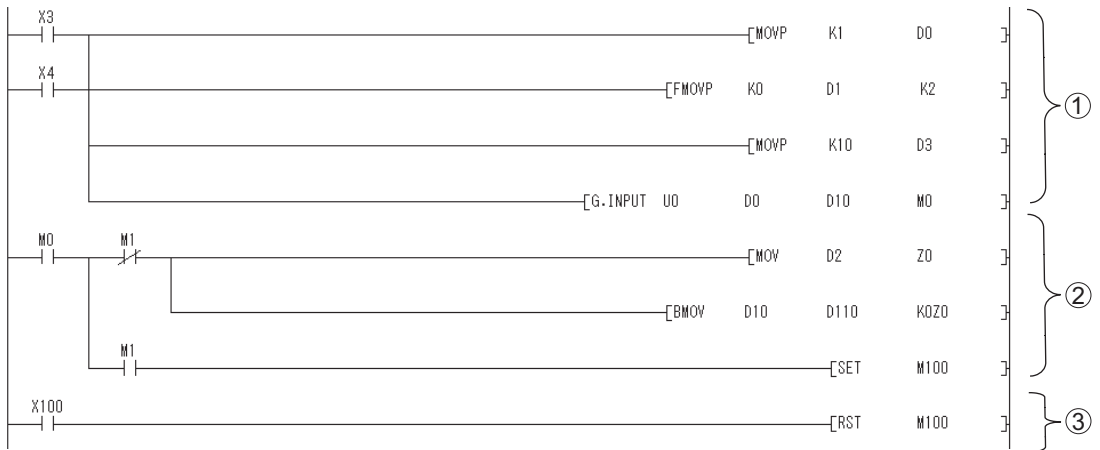| Class		Label Name	Data Type		Constant
VAR	▼	ControlData	Word[Unsigned]/Bit String[16-bit][0..3]	...	
VAR	▼	ReceiveData	Word[Unsigned]/Bit String[16-bit][0..1]	...	
VAR	▼	Completion	Bit[0..1]	...	
VAR	▼	Data	Word[Unsigned]/Bit String[16-bit][0..9]	...	
VAR	▼	AbnormalCompletion	Bit	...	

*1: Devices of local labels are automatically assigned within the range specified in the device/label automatic-assign setting in GX Works2.
To assign the same devices as those in the original ladder program, set them as global labels.
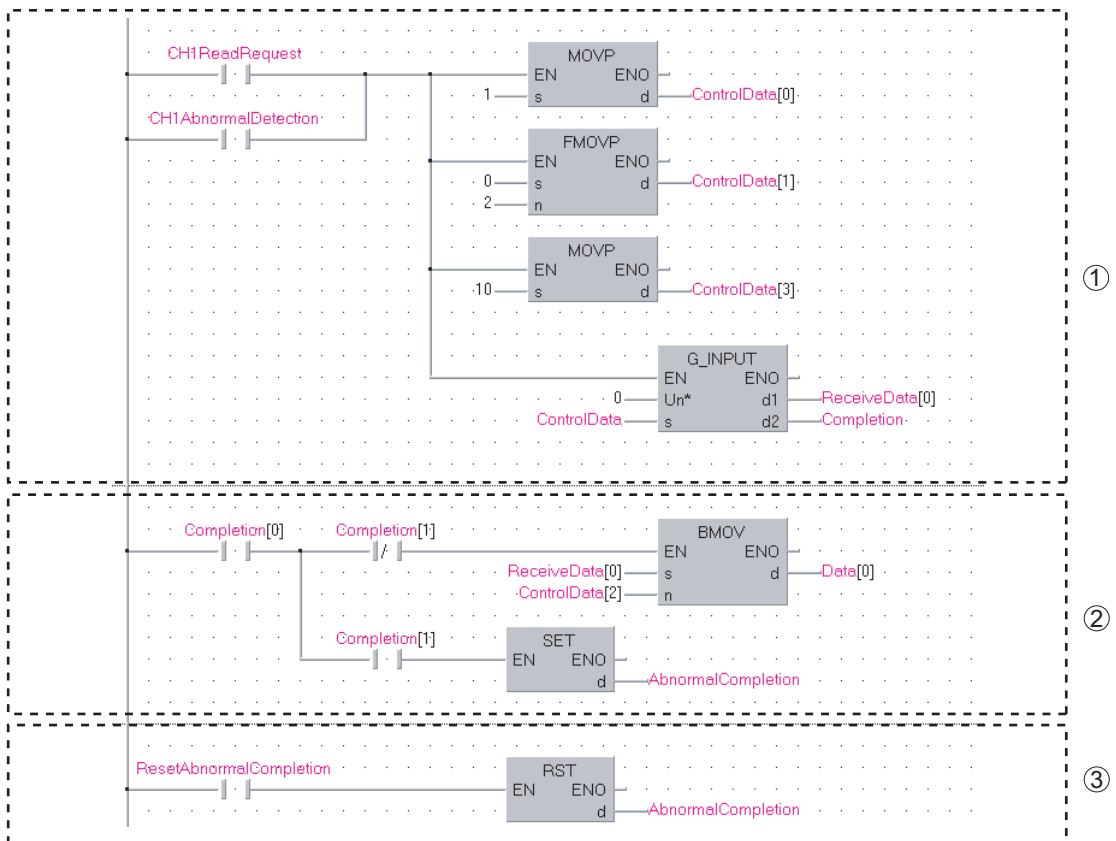
(3) Creating a structured program

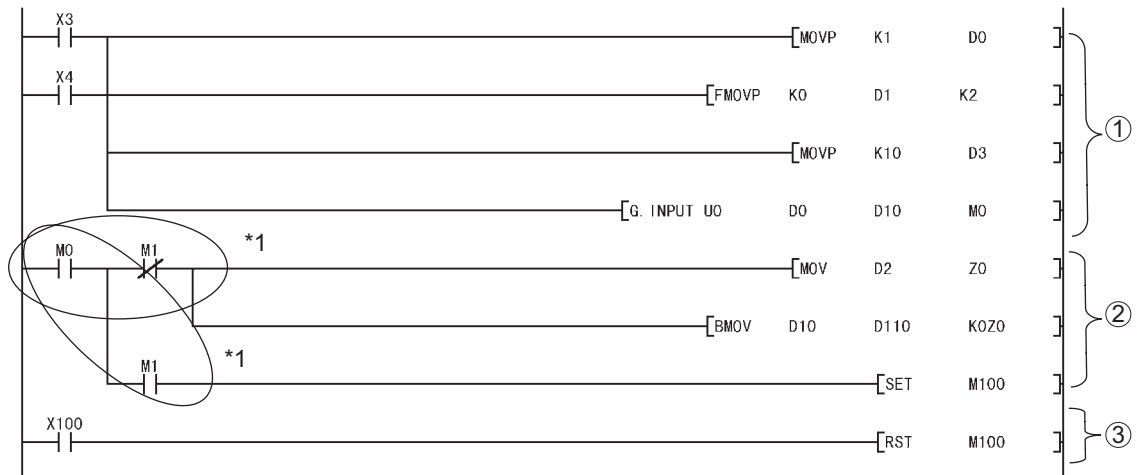The following examples show how a structured program is created based on the original program.

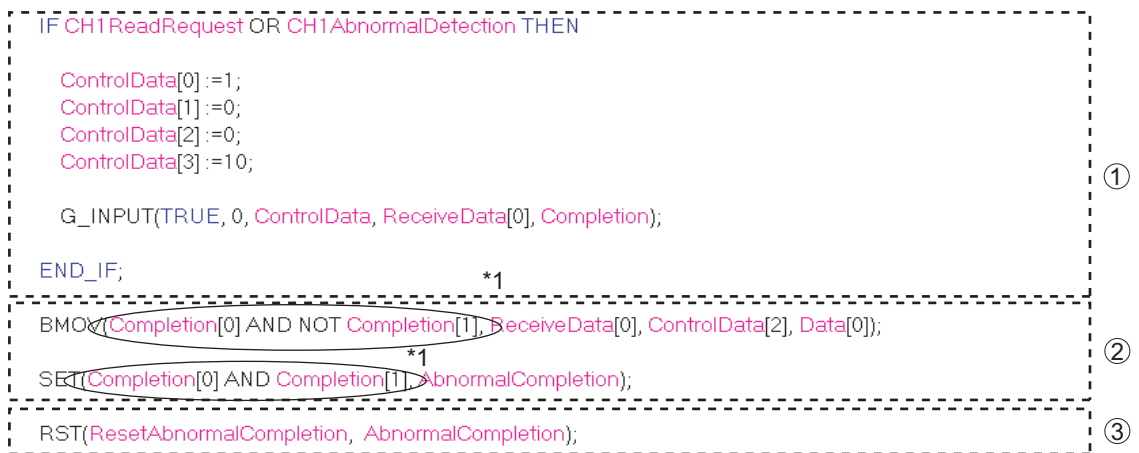• Original program (Programming language: ladder)

```
X3
─┤├──────────────────────────────────────[MOVP    K1      D0   ]─┐
X4                                                               │
─┤├──────────────────────────────────[FMOVP   K0      D1    K2 ]─┤ ①
                                        ─────────────[MOVP    K10     D3   ]─┤
                              [G.INPUT  U0      D0      D10     M0  ]─┘
M0    M1
─┤├──┤/├──────────────────────────────────[MOV     D2      Z0   ]─┐
      └──────────────────────────[BMOV    D10     D110    K0Z0]─┤ ②
      M1
      ─┤├────────────────────────────────────────[SET     M100 ]─┘
X100
─┤├──────────────────────────────────────────────[RST     M100 ]─ ③
```

• Structured program (Programming language: structured ladder)

```
CH1ReadRequest                        ┌─────MOVP─────┐
──┤├──┬─────────────────┬─────────────┤EN        ENO ├
CH1AbnormalDetection │             1 ─┤s          d  ├─ControlData[0]
──┤├──┘               │             └──────────────┘
                      │              ┌─────FMOVP────┐
                      │              ┤EN        ENO ├
                      │            0─┤s          d  ├─ControlData[1]
                      │            2─┤n            │
                      │              └──────────────┘
                      │              ┌─────MOVP─────┐
                      │              ┤EN        ENO ├
                      │          10─┤s          d  ├─ControlData[3]
                      │              └──────────────┘
                      │              ┌─────G_INPUT──┐
                      └──────────────┤EN        ENO ├
                                   0─┤Un*        d1 ├─ReceiveData[0]
                          ControlData─┤s          d2 ├─Completion
                                      └──────────────┘
```
①

```
Completion[0]   Completion[1]      ┌─────BMOV─────┐
──┤├──────────┬──┤/├────────────────┤EN        ENO ├
              │              ReceiveData[0]─┤s          d  ├─Data[0]
              │              ControlData[2]─┤n            │
              │                   └──────────────┘
              │  Completion[1]    ┌─────SET──────┐
              └──┤├────────────────┤EN        ENO ├
                                   ┤          d   ├─AbnormalCompletion
                                   └──────────────┘
```
②

```
ResetAbnormalCompletion         ┌─────RST──────┐
──┤├────────────────────────────┤EN        ENO ├
                                ┤          d   ├─AbnormalCompletion
                                └──────────────┘
```
③

• Original program (Programming language: ladder)

• Structured program (Programming language: ST)

```
IF CH1ReadRequest OR CH1AbnormalDetection THEN

    ControlData[0] :=1;
    ControlData[1] :=0;
    ControlData[2] :=0;
    ControlData[3] :=10;

    G_INPUT(TRUE, 0, ControlData, ReceiveData[0], Completion);

END_IF;                                          *1
                                                                    ①

BMOV(Completion[0] AND NOT Completion[1], ReceiveData[0], ControlData[2], Data[0]);
                                    *1                                              ②
SET(Completion[0] AND Completion[1], AbnormalCompletion);

RST(ResetAbnormalCompletion, AbnormalCompletion);                                   ③
```

*1: When using multiple contacts for execution conditions, enclose them by '( )' to be
    programmed in a group.

I INDEX

1

OVERVIEW

2

STRUCTURED DESIGN OF
SEQUENCE PROGRAMS

3

PROCEDURE FOR
CREATING PROGRAMS

4

PROGRAM
CONFIGURATION

5

WRITING
PROGRAMS

A

APPENDIX

INDEX

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 onsite that involves replacement of the failed module.

[Gratis Warranty Term]

The gratis warranty term of the product shall be for one year after the date of purchase or delivery to a designated place. Note that after manufacture and shipment from Mitsubishi, the maximum distribution period shall be six (6) months, and the longest gratis warranty term after manufacturing shall be eighteen (18) months. The gratis warranty term of repair parts shall not exceed the gratis warranty term before repairs.

[Gratis Warranty Range]

(1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.
(2) Even within the gratis warranty term, repairs shall be charged for in the following cases.
   1. Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
   2. Failure caused by unapproved modifications, etc., to the product by the user.
   3. When the Mitsubishi product is assembled into a user's device, Failure that could have been avoided if functions or structures, judged as necessary in the legal safety measures the user's device is subject to or as necessary by industry standards, had been provided.
   4. Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
   5. Failure caused by external irresistible forces such as fires or abnormal voltages, and Failure caused by force majeure such as earthquakes, lightning, wind and water damage.
   6. Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
   7. Any other failure found not to be the responsibility of Mitsubishi or that admitted not to be so by the user.

2. Onerous repair term after discontinuation of production

(1) Mitsubishi shall accept onerous product repairs for seven (7) years after production of the product is discontinued.
   Discontinuation of production shall be notified with Mitsubishi Technical Bulletins, etc.
(2) Product supply (including repair parts) is not available after production is discontinued.

3. Overseas service

Overseas, repairs shall be accepted by Mitsubishi's local overseas FA Center. Note that the repair conditions at each FA Center may differ.

4. Exclusion of loss in opportunity and secondary loss from warranty liability

Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation to damages caused by any cause found not to be the responsibility of Mitsubishi, loss in opportunity, lost profits incurred to the user by Failures of Mitsubishi products, special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products, 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.

MELSEC-Q/L/F Structured
Programming Manual

Fundamentals

MODEL	Q/FX-KP-KI-E
MODEL CODE	13JW06
SH(NA)-080782ENG-E(1001)KWIX	

MITSUBISHI ELECTRIC CORPORATION

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.