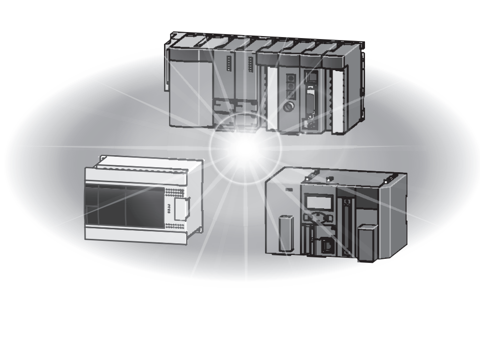
MITSUBISHI
ELECTRIC

Programmable Controller

MELSEC Q series MELSEC L series MELSEC-F

# MELSEC-Q/L/F Structured Programming Manual (Fundamentals)



# SAFETY PRECAUTIONS

(Read these precautions before using this product.)

Before using MELSEC-Q, -L, or -F series programmable controllers, please read the manuals included with each product and the relevant manuals introduced in those manuals carefully, and pay full attention to safety to handle the product correctly.

Make sure that the end users read the manuals included with each product, and keep the manuals in a safe place for future reference.

# CONDITIONS OF USE FOR THE PRODUCT

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

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

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

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

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

("Prohibited Application")

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

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

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

# INTRODUCTION

Thank you for purchasing the Mitsubishi Electric MELSEC-Q, -L, or -F series programmable controllers.
Before using this product, please read this manual and the relevant manuals carefully and develop familiarity with the programming specifications to handle the product correctly.
When applying the program examples introduced in this manual to an actual system, ensure the applicability and confirm that it will not cause system control problems.

## Applicable CPU modules

| CPU module | Model |
|---|---|
| Basic model QCPU | Q00JCPU, Q00CPU, Q01CPU |
| High Performance model QCPU | Q02CPU, Q02HCPU, Q06HCPU, Q12HCPU, Q25HCPU |
| Process CPU | Q02PHCPU, Q06PHCPU, Q12PHCPU, Q25PHCPU |
| Redundant CPU | Q12PRHCPU, Q25PRHCPU |
| Universal model QCPU | Q00UJCPU, Q00UCPU, Q01UCPU, Q02UCPU, Q03UDCPU, Q03UDVCPU, Q03UDECPU, Q04UDHCPU, Q04UDVCPU, Q04UDPVCPU, Q04UDEHCPU, Q06UDHCPU, Q06UDVCPU, Q06UDPVCPU, Q06UDEHCPU, Q10UDHCPU, Q10UDEHCPU, Q13UDHCPU, Q13UDVCPU, Q13UDPVCPU, Q13UDEHCPU, Q20UDHCPU, Q20UDEHCPU, Q26UDHCPU, Q26UDVCPU, Q26UDPVCPU, Q26UDEHCPU, Q50UDEHCPU, Q100UDEHCPU |
| LCPU | L02SCPU, L02SCPU-P, L02CPU, L02CPU-P, L06CPU, L06CPU-P, L26CPU, L26CPU-P, L26CPU-BT, L26CPU-PBT |
| FXCPU | $FX_{0S}$, $FX_{0}$, $FX_{0N}$, $FX_{1S}$, $FX_{1N}$, $FX_{1NC}$, $FX_{2}$, $FX_{2C}$, $FX_{2N}$, $FX_{2NC}$, $FX_{3S}$, $FX_{3G}$, $FX_{3GC}$, $FX_{3U}$, $FX_{3UC}$ |

## Compatible software package

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

| Software package name | Model name |
|---|---|
| GX Works2 | SW1DNC-GXW2-E, SW1DND-GXW2-E |

### ■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

# CONTENTS

# MANUALS

## Related Manuals

The manuals related to this product are listed below. Please place an order as needed.

### ■Structured programming

| Manual name<br><Manual number> | Description |
|---|---|
| MELSEC-Q/L Structured Programming Manual (Common Instructions)<br><SH-080783ENG> | Specifications and functions of common instructions, such as sequence instructions, basic instructions, and application instructions, that can be used in structured programs |
| MELSEC-Q/L Structured Programming Manual (Application Functions)<br><SH-080784ENG> | Specifications and functions of application functions that can be used in structured programs |
| MELSEC-Q/L Structured Programming Manual (Special Instructions)<br><SH-080785ENG> | Specifications and functions of special instructions, such as module dedicated instructions, PID control instructions, and built-in I/O function instructions, that can be used in structured programs |
| FXCPU Structured Programming Manual [Device & Common]<br><JY997D26001> | Devices and parameters for structured programming provided in GX Works2 |
| FXCPU Structured Programming Manual [Basic & Applied Instruction]<br><JY997D34701> | Sequence instructions for structured programming provided in GX Works2 |
| FXCPU Structured Programming Manual [Application Functions]<br><JY997D34801> | Application functions for structured programming provided in GX Works2 |

### ■Operation of GX Works2

| Manual name<br><Manual number> | Description |
|---|---|
| GX Works2 Version 1 Operating Manual (Common)<br><SH-080779ENG> | System configuration, parameter settings, and online operations of GX Works2, which are common to Simple projects and Structured projects |
| GX Works2 Version 1 Operating Manual (Structured Project)<br><SH-080781ENG> | Operations, such as programming and monitoring in Structured projects, of GX Works2 |
| GX Works2 Beginner's Manual (Structured Project)<br><SH-080788ENG> | Basic operations, such as programming, editing, and monitoring in Structured projects, of GX Works2. This manual is intended for first-time users of GX Works2. |

# TERMS

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.

| Term | Description |
|---|---|
| Application function | A generic term for the functions, such as functions and function blocks, defined in IEC 61131-3.<br>(The functions are executed with a set of common instructions in a programmable controller.) |
| Basic model QCPU | A generic term for the Q00JCPU, Q00CPU, and Q01CPU |
| Common instruction | A generic term for the sequence instructions, basic instructions, application instructions, data link instructions, multiple CPU dedicated instructions, multiple CPU high-speed transmission dedicated instructions, and redundant system instructions |
| CPU module | A generic term for the QCPU (Q mode), LCPU, and FXCPU |
| FXCPU | A generic term for MELSEC-FX series programmable controllers<br>($FX_{0S}$, $FX_0$, $FX_{0N}$, $FX_1$, $FX_{1S}$, $FX_{1N}$, $FX_{1NC}$, $FX_2$, $FX_{2C}$, $FX_{2N}$, $FX_{2NC}$, $FX_{3S}$, $FX_{3G}$, $FX_{3GC}$, $FX_{3U}$, $FX_{3UC}$) |
| GX Developer | The product name of the software package for the MELSEC programmable controllers |
| GX Works2 | |
| High Performance model QCPU | A generic term for the Q02CPU, Q02HCPU, Q06HCPU, Q12HCPU, and Q25HCPU |
| High-speed Universal model QCPU | A generic term for the Q03UDVCPU, Q04UDVCPU, Q06UDVCPU, Q13UDVCPU, and Q26UDVCPU |
| IEC 61131-3 | An abbreviation for the IEC 61131-3 international standard |
| LCPU | A generic term for the L02SCPU, L02SCPU-P, L02CPU, L02CPU-P, L06CPU, L06CPU-P, L26CPU, L26CPU-P, L26CPU-BT, and L26CPU-PBT |
| Personal computer | The generic term for personal computers where Windows® operates |
| Process CPU | A generic term for the Q02PHCPU, Q06PHCPU, Q12PHCPU, and Q25PHCPU |
| QCPU (Q mode) | A generic term for the Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU, and Universal model QCPU |
| QnU(D)(H)CPU | A generic term for the Q02UCPU, Q03UDCPU, Q04UDHCPU, Q06UDHCPU, Q10UDHCPU, Q13UDHCPU, Q20UDHCPU, and Q26UDHCPU |
| QnUDE(H)CPU | A generic term for the Q03UDECPU, Q04UDEHCPU, Q06UDEHCPU, Q10UDEHCPU, Q13UDEHCPU, Q20UDEHCPU, Q26UDEHCPU, Q50UDEHCPU, and Q100UDEHCPU |
| QnUDPVCPU | A generic term for the Q04UDPVCPU, Q06UDPVCPU, Q13UDPVCPU, and Q26UDPVCPU |
| QnUDVCPU | A generic term for the Q03UDVCPU, Q04UDVCPU, Q06UDVCPU, Q13UDVCPU, and Q26UDVCPU |
| Redundant CPU | A generic term for the Q12PRHCPU and Q25PRHCPU |
| Special instruction | A 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 |
| Universal model Process CPU | A generic term for the Q04UDPVCPU, Q06UDPVCPU, Q13UDPVCPU, and Q26UDPVCPU |
| Universal model QCPU | A generic term for the Q00UJCPU, Q00UCPU, Q01UCPU, Q02UCPU, Q03UDCPU, Q03UDVCPU, Q03UDECPU, Q04UDHCPU, Q04UDVCPU, Q04UDPVCPU, Q04UDEHCPU, Q06UDHCPU, Q06UDVCPU, Q06UDPVCPU, Q06UDEHCPU, Q10UDHCPU, Q10UDEHCPU, Q13UDHCPU, Q13UDVCPU, Q13UDPVCPU, Q13UDEHCPU, Q20UDHCPU, Q20UDEHCPU, Q26UDHCPU, Q26UDVCPU, Q26UDPVCPU, Q26UDEHCPU, Q50UDEHCPU, and Q100UDEHCPU |

# 1 OVERVIEW

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

## 1.1 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 following.
☞ Page 6 Related Manuals

### Operation of GX Works2

| Purpose | | Summary | Detail |
|---|---|---|---|
| Installation | Learning the operating environment and installation method | — | 📖 GX Works2 Installation Instructions |
| | Learning a USB driver installation method | — | 📖 GX Works2 Version 1 Operating Manual (Common) |
| Operation of GX Works2 | Learning all functions of GX Works2 | 📖 GX Works2 Version 1 Operating Manual (Common) | — |
| | Learning the project types and available languages in GX Works2 | | — |
| | Learning the basic operations and operating procedures when creating a simple project for the first time | — | 📖 GX Works2 Beginner's Manual (Simple Project) |
| | Learning the basic operations and operating procedures when creating a structured project for the first time | — | 📖 GX Works2 Beginner's Manual (Structured Project) |
| | Learning the operations of available functions regardless of project type. | — | 📖 GX Works2 Version 1 Operating Manual (Common) |
| | Learning the functions and operation methods for programming | 📖 GX Works2 Version 1 Operating Manual (Common) | 📖 GX Works2 Version 1 Operating Manual (Simple Project)<br>📖 GX Works2 Version 1 Operating Manual (Structured Project) |
| | Learning data setting methods for intelligent function module | — | 📖 GX Works2 Version 1 Operating Manual (Intelligent Function Module) |

### Operations in each programming language

For details of instructions used in each programming language, refer to the following.
☞ Page 9 Details of instructions in each programming language

| Purpose | | Summary | Detail |
|---|---|---|---|
| Simple Project | Ladder | 📖 GX Works2 Beginner's Manual (Simple Project) | 📖 GX Works2 Version 1 Operating Manual (Simple Project) |
| | SFC | 📖 GX Works2 Beginner's Manual (Simple Project)*1 | |
| | ST | 📖 GX Works2 Beginner's Manual (Structured Project) | 📖 GX Works2 Version 1 Operating Manual (Structured Project) |
| Structured Project | Ladder | 📖 GX Works2 Beginner's Manual (Simple Project) | 📖 GX Works2 Version 1 Operating Manual (Simple Project) |
| | SFC | 📖 GX Works2 Beginner's Manual (Simple Project)*1 | |
| | Structured ladder/FBD | 📖 GX Works2 Beginner's Manual (Structured Project) | 📖 GX Works2 Version 1 Operating Manual (Structured Project) |
| | ST | | |

*1 MELSAP3 and FX series SFC only

## Details of instructions in each programming language

### ■QCPU (Q mode)/LCPU

| Purpose | | Summary | Detail |
|---|---|---|---|
| All languages | Learning details of programmable controller CPU error codes, special relay areas, and special register areas | — | User's Manual (Hardware Design, Maintenance and Inspection) for the CPU module used |
| Using ladder language | Learning the types and details of common instructions | — | MELSEC-Q/L Programming Manual (Common Instruction) |
| | Learning the types and details of instructions for intelligent function modules | — | Manual for the intelligent function module used |
| | Learning the types and details of instructions for network modules | — | Manual for the network module used |
| | Learning the types and details of instructions for the PID control function | — | MELSEC-Q/L/QnA Programming Manual (PID Control Instructions) |
| | Learning the types and details of the process control instructions | — | MELSEC-Q Programming/Structured Programming Manual (Process Control Instructions) |
| Using SFC language | Learning details of specifications, functions, and instructions of SFC (MELSAP3) | — | MELSEC-Q/L/QnA Programming Manual (SFC) |
| Using structured ladder/FBD/ST language | Learning the fundamentals for creating a structured program | — | MELSEC-Q/L/F Structured Programming Manual (Fundamentals) |
| | Learning the types and details of common instructions | — | MELSEC-Q/L Structured Programming Manual (Common Instructions) |
| | Learning the types and details of instructions for intelligent function modules | MELSEC-Q/L Structured Programming Manual (Special Instructions) | Manual for the intelligent function module used |
| | Learning the types and details of instructions for network modules | | Manual for the network module used |
| | Learning the types and details of instructions for the PID control function | | MELSEC-Q/L/QnA Programming Manual (PID Control Instructions) |
| | Learning the types and details of application functions | — | MELSEC-Q/L Structured Programming Manual (Application Functions) |
| | Learning the types and details of the process control instructions | — | MELSEC-Q Programming/Structured Programming Manual (Process Control Instructions) |

### ■FXCPU

| Purpose | | Summary | Detail |
|---|---|---|---|
| Using ladder language | Learning the types and details of basic/application instructions, descriptions of devices and parameters | — | Programming manual for the FXCPU used |
| Using SFC language | Learning details of specifications, functions, and instructions of SFC | — | |
| Using structured ladder/FBD/ST language | Learning the fundamentals for creating a structured program | — | MELSEC-Q/L/F Structured Programming Manual (Fundamentals) |
| | Learning the descriptions of devices, parameters, and error codes | — | FXCPU Structured Programming Manual [Device & Common] |
| | Learning the types and details of sequence instructions | — | FXCPU Structured Programming Manual [Basic & Applied Instruction] |
| | Learning the types and details of application functions | — | FXCPU Structured Programming Manual [Application Functions] |

# 1.2 Features of Structured Programs

This section explains the features of structured programs.

## Structured design

A structured design is a method to program control content 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.

## 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 POU.

| Name | | Description |
|---|---|---|
| ST (structured text) | | A text language similar to C language, aimed for computer engineers. |
| Structured ladder/FBD | Structured ladder | A graphic language that is expressed in form of ladder by using elements such as contacts and coils. |
| | FBD | A graphic language that is expressed in form of ladder by connecting elements such as functions and function blocks with lines. |

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

☞ Page 23 Programming languages for POUs

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

☞ Page 73 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

## Improved program reusability

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

# 2 STRUCTURED DESIGN OF SEQUENCE PROGRAMS

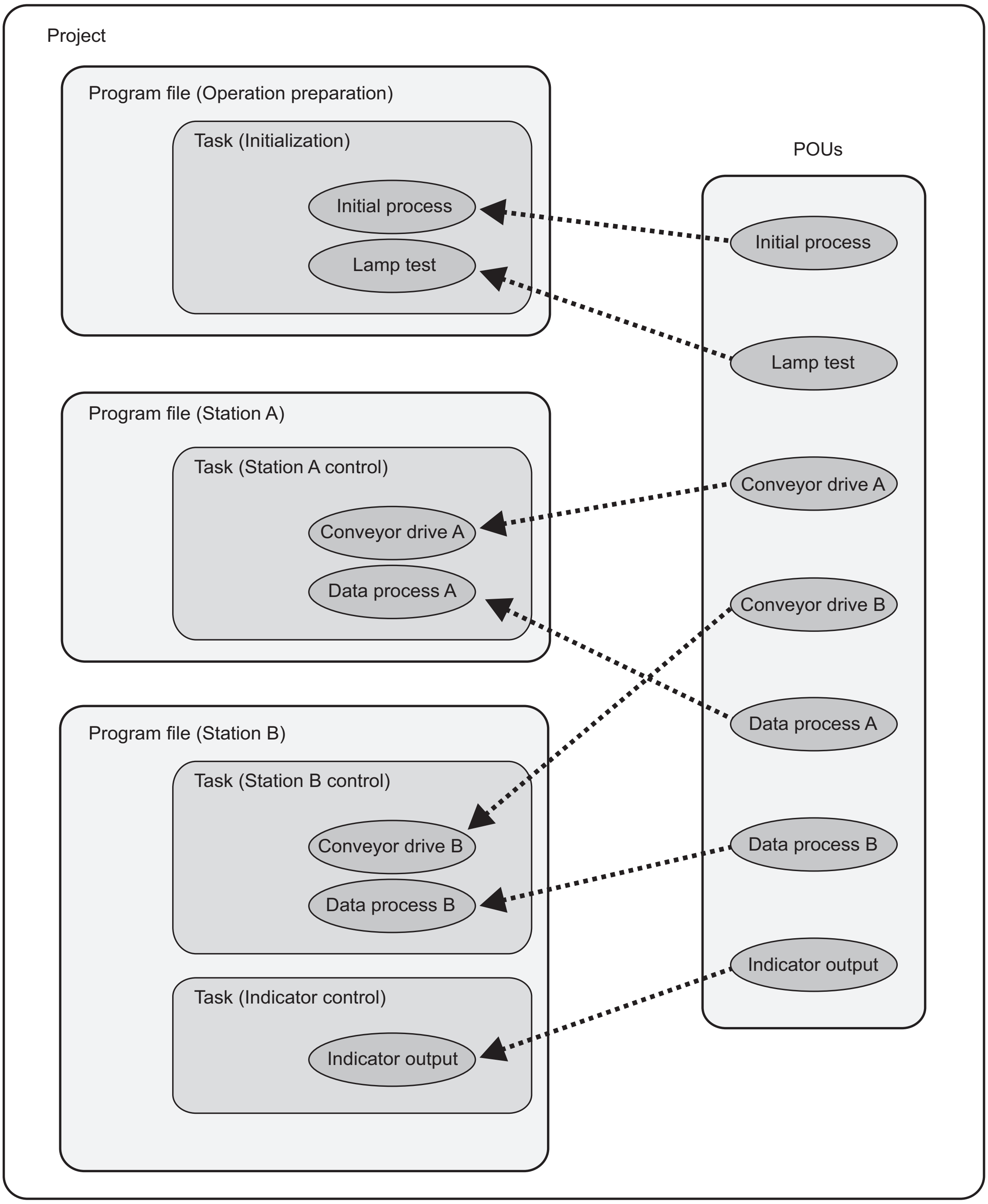
## 2.1 Hierarchical Sequence Program



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

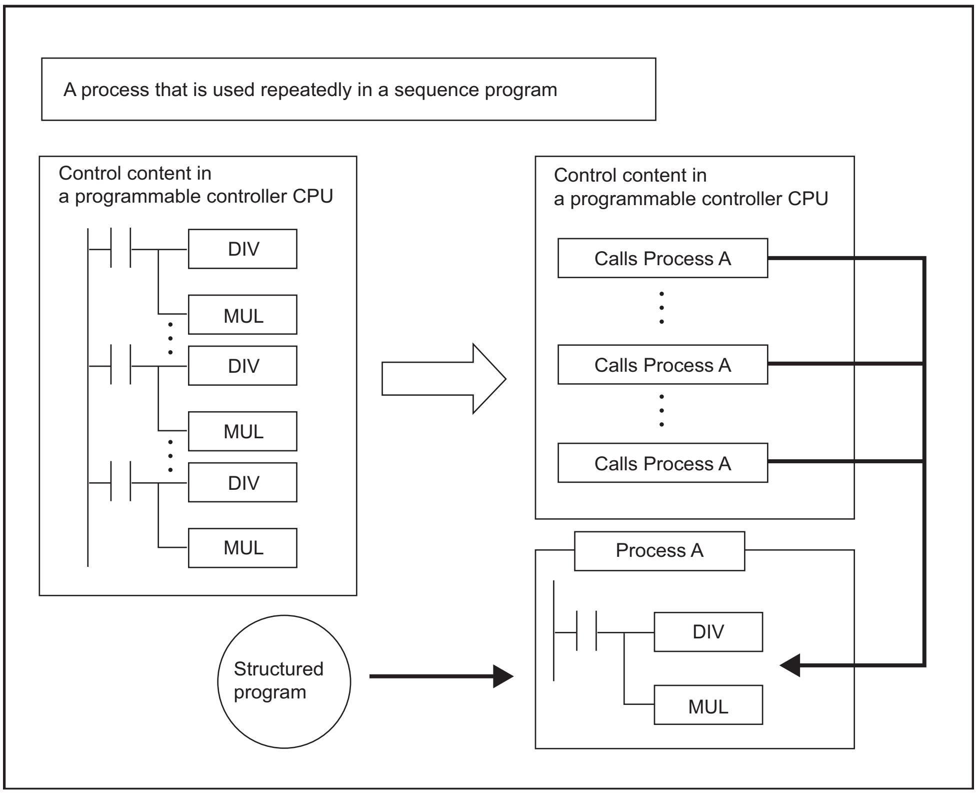
In higher levels, the processing order and timing in a fixed range is controlled. With each move from a higher level to a lower level, control content and processes are progressively subdivided within a fixed range, and specific processes are described in lower levels.

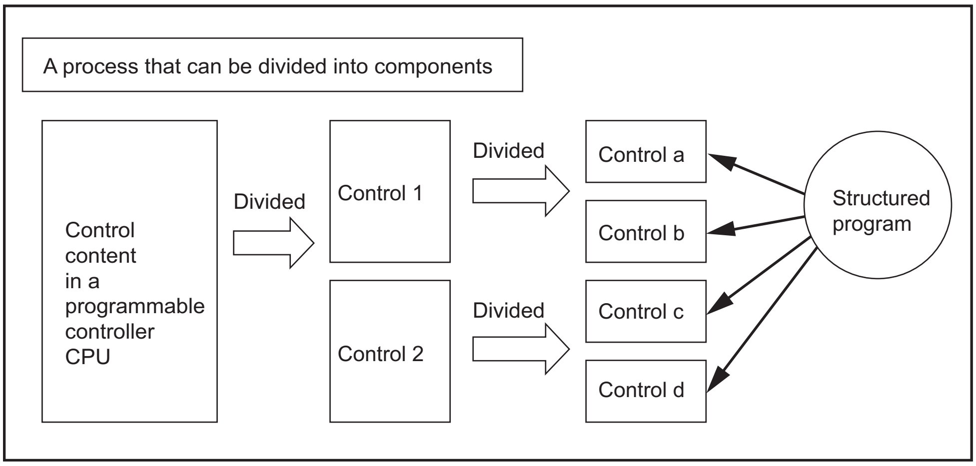
In the Structured project, hierarchical sequence programs are created with the configuration that states the highest level as the project, followed by program files, tasks, and POUs (abbreviation for Program Organization Units).



## 2.2 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.



# 3 PROCEDURE FOR CREATING PROGRAMS

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

**1.** Creating the program configuration

- Create program files.
- Create tasks.

**2.** Creating POUs

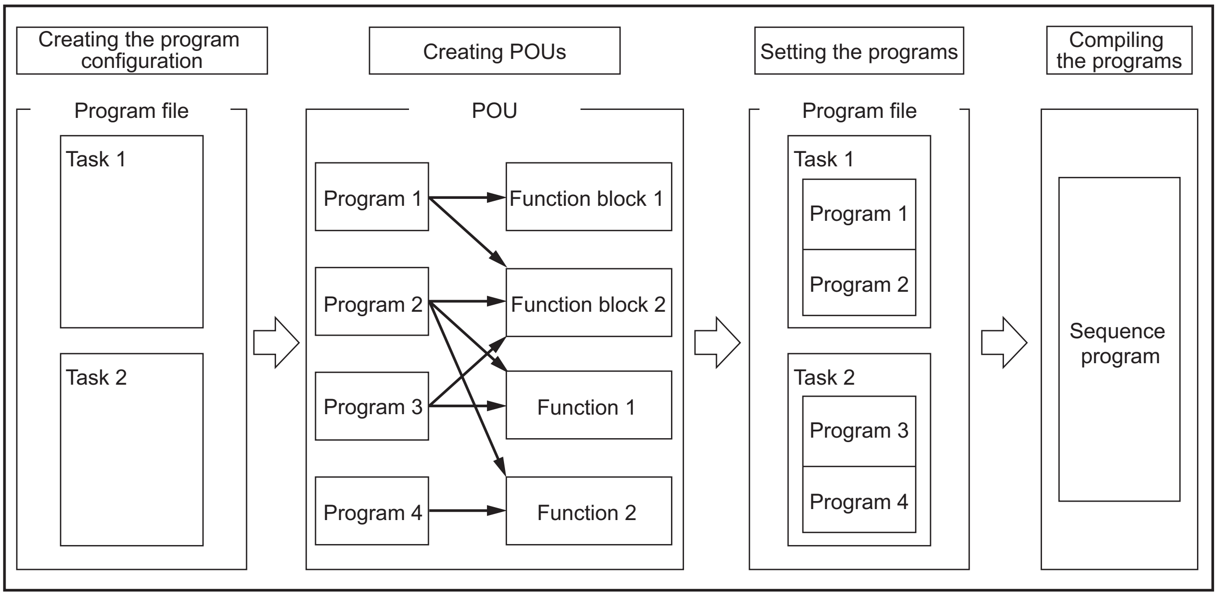
- Create POUs.
- Define global labels.
- Define local labels.
- Edit the programs of each POU.

**3.** Setting the programs

Register the POUs in the tasks.

**4.** Compiling the programs

Compile the programs.



# MEMO

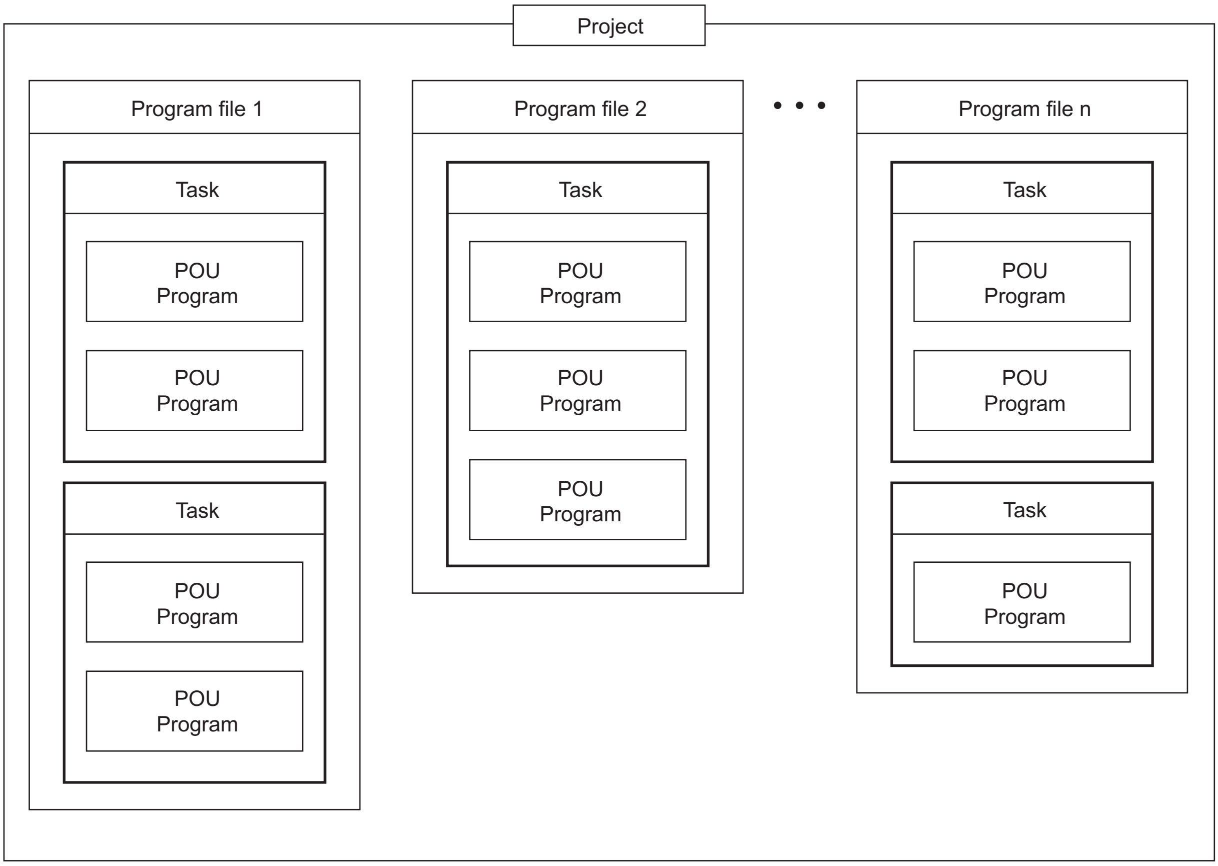
# 4 PROGRAM CONFIGURATION

## 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.

| Item | Reference |
|---|---|
| Projects | Page 16 Project |
| Program files | Page 16 Program files |
| Tasks | Page 17 Tasks |
| POUs | Page 18 POUs |

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



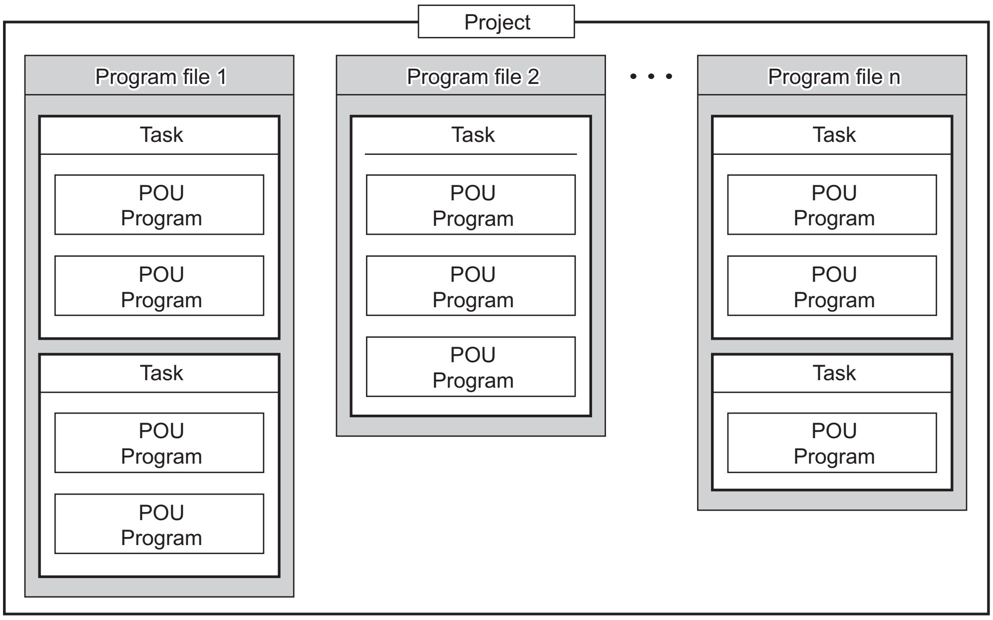

4

# 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.

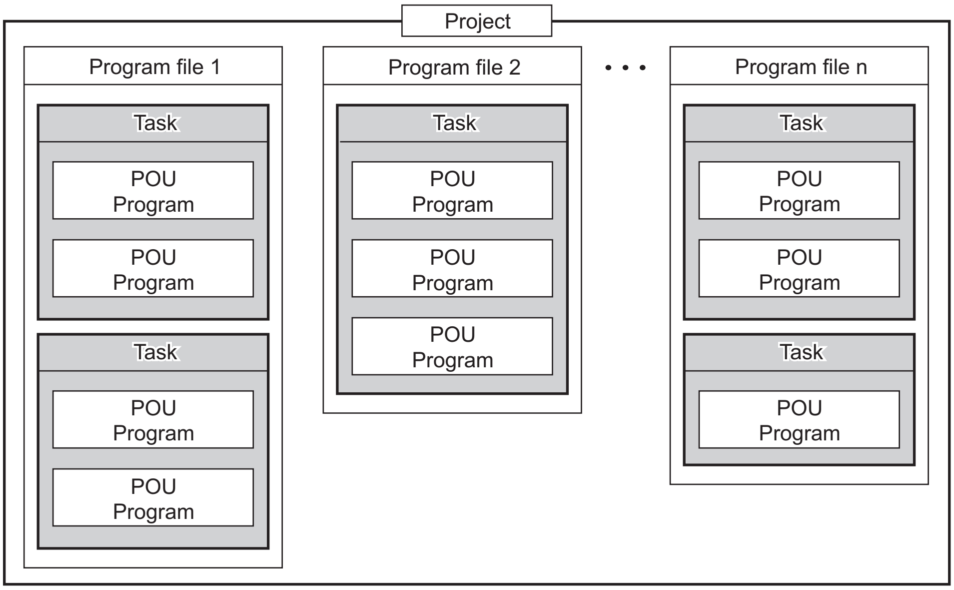
# Program files

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 of the execution types set in the parameter, refer to the user's manual for the CPU module used.



# Tasks

A task is an element that contains multiple POUs, and it is registered to a program file. One or more programs of POU need to be registered in a task. (Functions and function blocks cannot be registered in a task.)




## 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.

### ■Always (Default executing condition)

Executes registered programs for each scan.

### ■Event

Executes tasks when values are set to the corresponding devices or labels.

### ■Interval

Executes tasks in a specified cycle.

## Priority

A priority can be set for each task execution.

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.

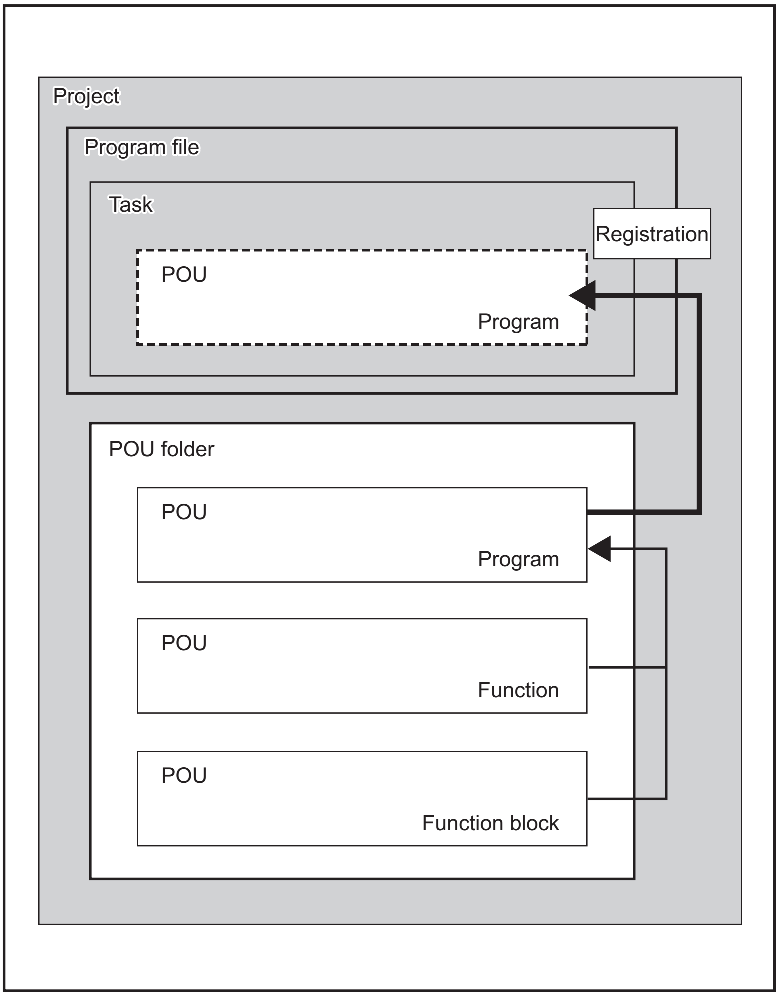
## Types of POU

The following three types can be selected for each POU according to the content to be defined.

- Program
- Function
- Function block

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

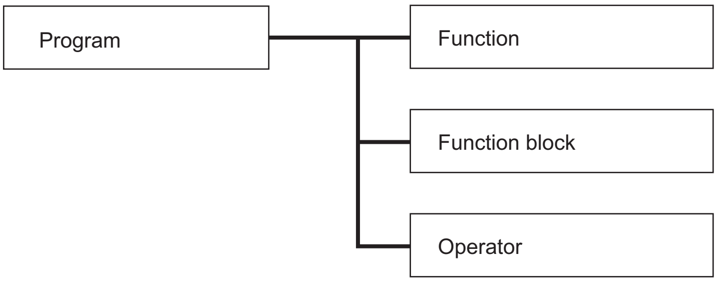
A process can be described in a programming language that suits the control function for each POU.




*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.
☞ Page 29 Local labels

## Program

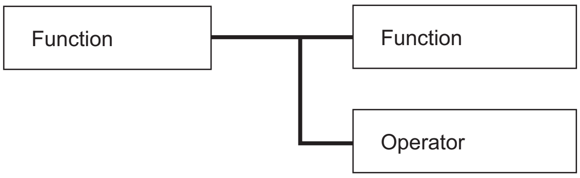
A program is an element that is stated at the highest level of POU. Functions, function blocks, and operators are used to edit programs.




Sequence programs executed in a programmable controller CPU are created by programs of POU. For a simplest sequence program, only one program needs to be created and registered to a task in order to be executed in a programmable controller CPU.
Programs can be described in the ST or structured ladder/FBD language.

## Functions

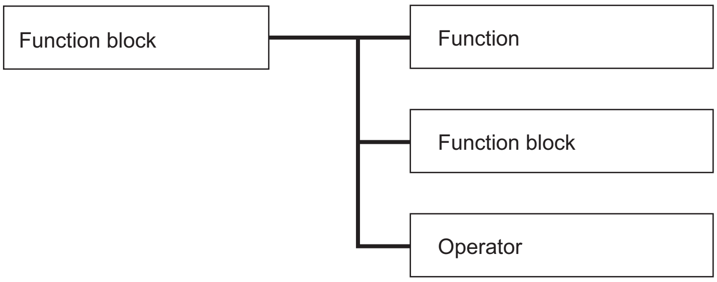
Functions and operators are used to edit functions. Functions can be used by calling them from programs, functions, or function blocks.




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/FBD language.

# Function blocks

Functions, function blocks, and operators are used to edit function blocks. Function blocks can be used by calling them from programs 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/FBD language.

## Instantiation

Function blocks need to be instantiated to be used in programs. (☞ Page 24 Functions, function blocks, and operators)

**Point**

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

## Function block specifications

### ■Nesting

- Macro type function block: 5 levels (Ladder diagram: 2 levels)
- Subroutine type function block: 16 levels

A macro type function block can be called from a subroutine type function block.
A subroutine type function block cannot be called from a macro type function block.

### ■Number of function blocks that can be called from within a function block.

- Macro type function blocks: No limit[*1]
- Subroutine type function block: No limit[*2]

*1 There is no limit as long as the memory capacity is not exceeded.
*2 There is no limit as long as the upper limit of pointer in the auto device setting (default: 2048) is not exceeded. Note, depending on the program, that the upper limit is less than usual because the pointer device is used for other than subroutine type function blocks.

## Operators

Operators can be used by calling them from programs, functions, or function blocks. Operators cannot be edited.
Operators always output same processing results for the same input values.

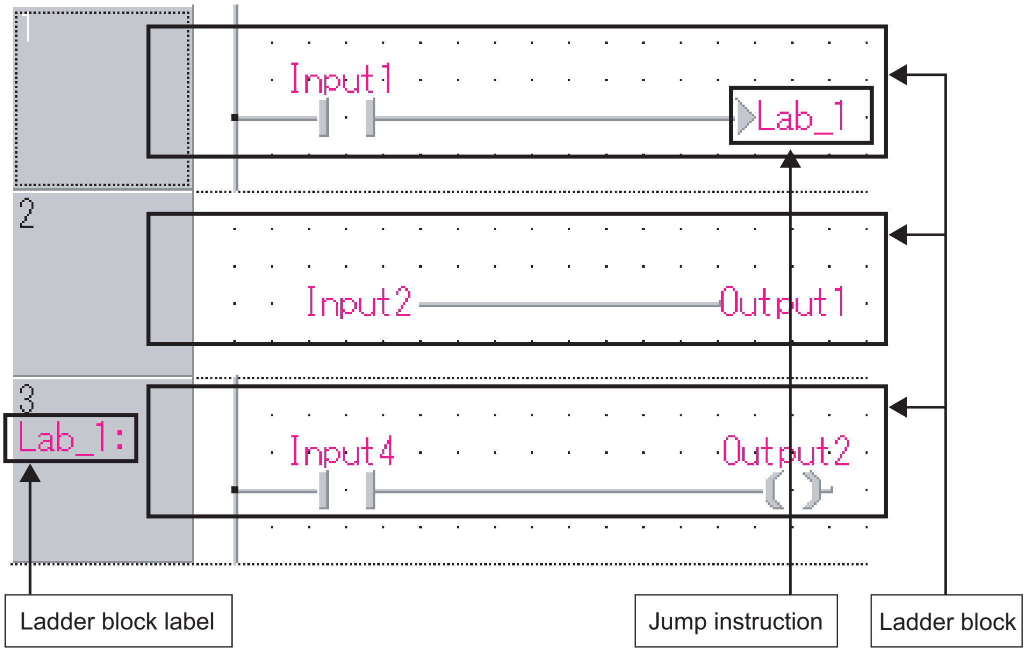
## Ladder blocks

In the structured ladder/FBD 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.



# Programming languages for POUs

Two types of programming language are available for programs of POU.
The following explains the features of each programming language.

## ST: Structured text

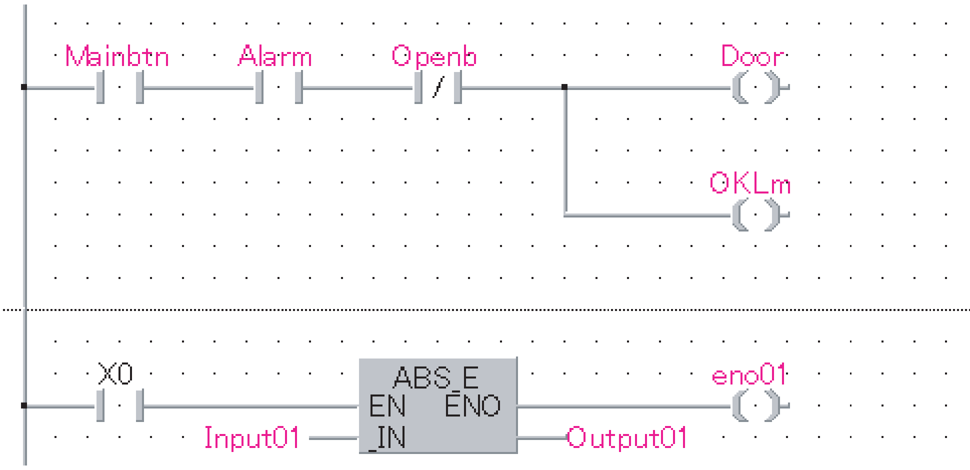
Control syntax such as selection branch by conditional syntax or repetitions by iterative syntax can be described in the structured text language, as in the high-level language such as C language. Clear and simple programs can be written by using these syntax.

```
intV2 := ABS( intV1);

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

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

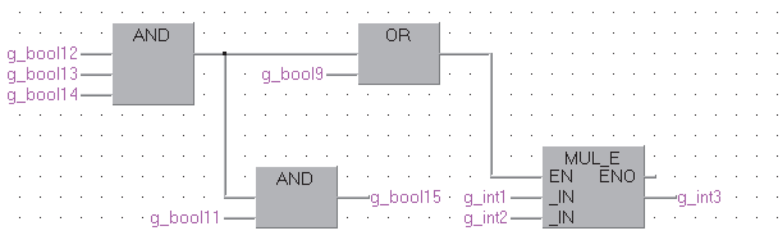
## Structured ladder/FBD: (ladder diagram)

The structured ladder or FBD is a graphical language developed based on the relay ladder programming technique. They are commonly used for the sequence programming because they can be understood intuitively.

- Structured ladder




- FBD



# Functions, function blocks, and operators

The following table shows differences among functions, function blocks, and operators.

| Item | Function | Function block | Operator |
|---|---|---|---|
| Output variable assignment | Cannot be assigned | Can be assigned | Cannot be assigned |
| Internal variable | Not used | Used | Not used |
| Creating instances | Not necessary | Necessary | Not necessary |

## Output variable assignment

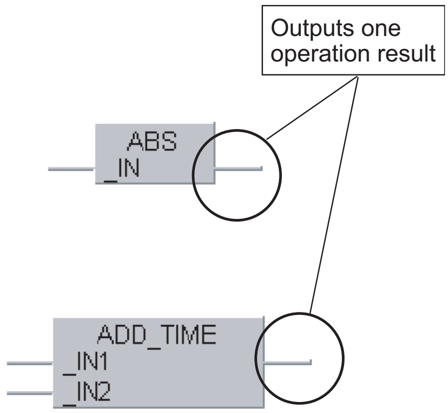
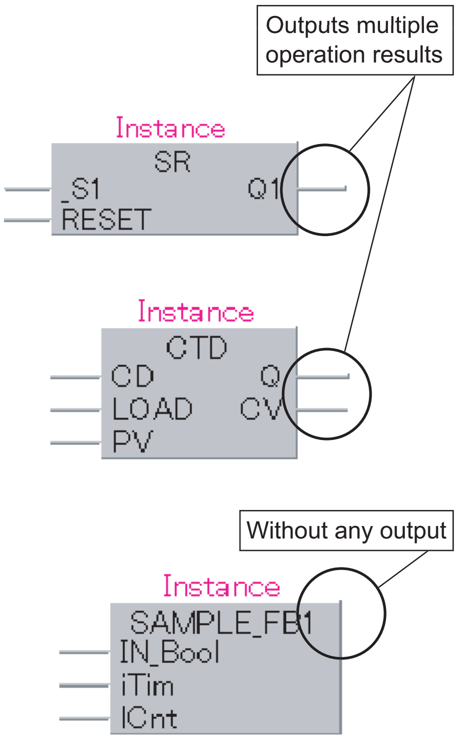
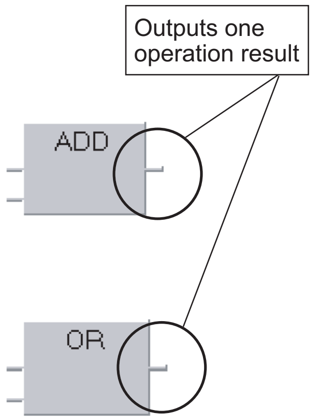
A function always outputs a single operation result. A function that does not output any operation result or outputs multiple operation results cannot be created.

A function block can output multiple operation results. It also can be created without any output.

An operator always outputs a single operation result. It cannot be edited.

**Ex.**

The following table shows the examples.

| Function | Function block | Operator |
|---|---|---|
|  |  |  |

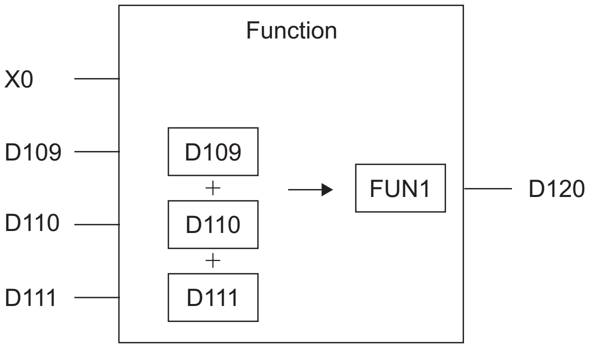
## Internal variables

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

A program that outputs the total of three input variables

Ex.

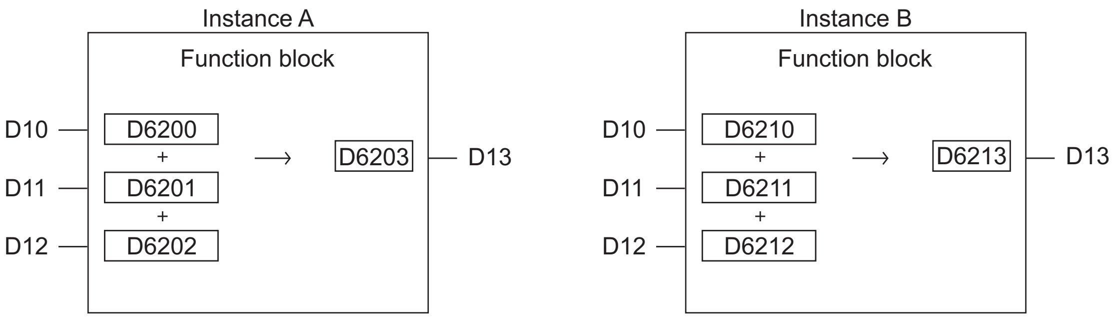
When using a function (FUN1)




A function block uses internal variables. Different devices are assigned to the internal variables for each instance of function blocks.

Ex.

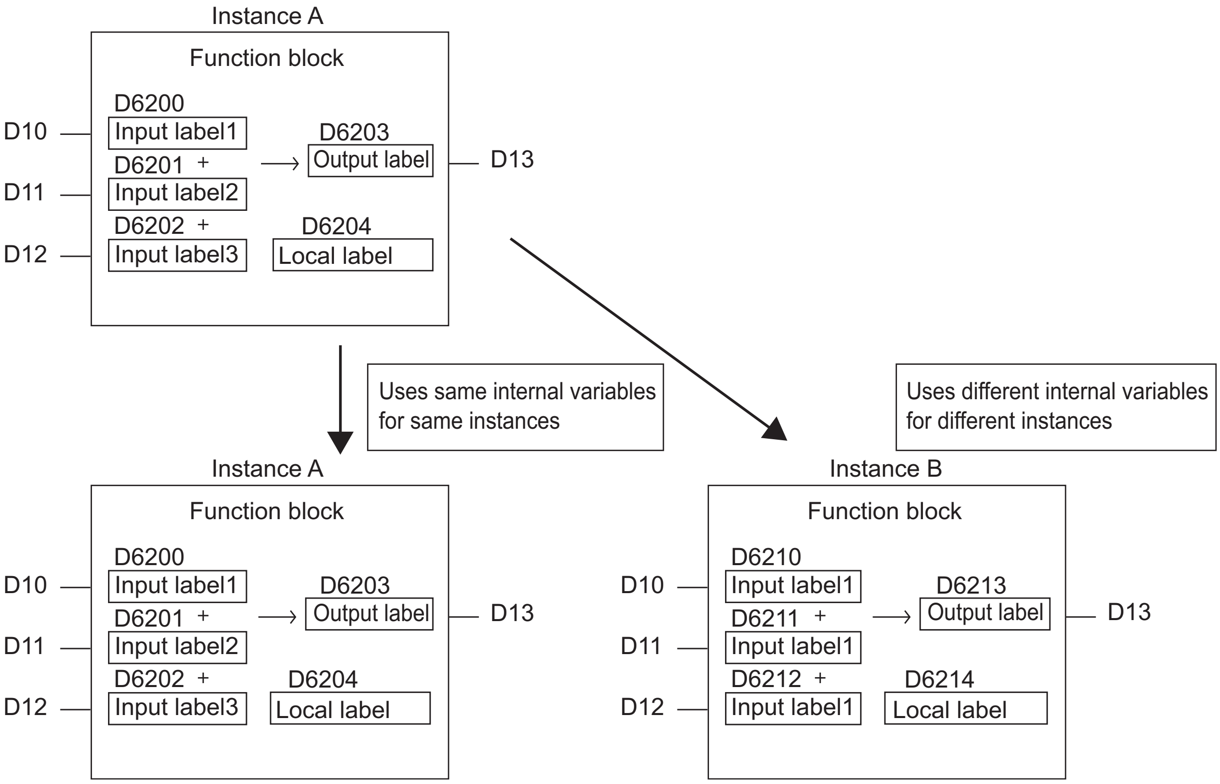
When using function blocks



## Creating instances

When using function blocks, create instances to reserve internal variables. Variables can be called from programs 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.



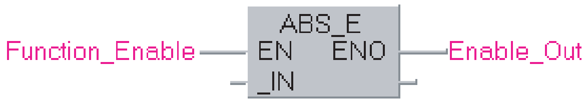

Function blocks perform operations using internal variables assigned to each instance.

**Point**

If the same function is called in the circuit multiple times, the value of internal variables or output variables is overwritten everytime the function is called. To hold the value of internal variables or output variables when the function is called, edit programs to use function blocks or to save the values as different valuables.

## 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 execution result is set to an ENO.




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

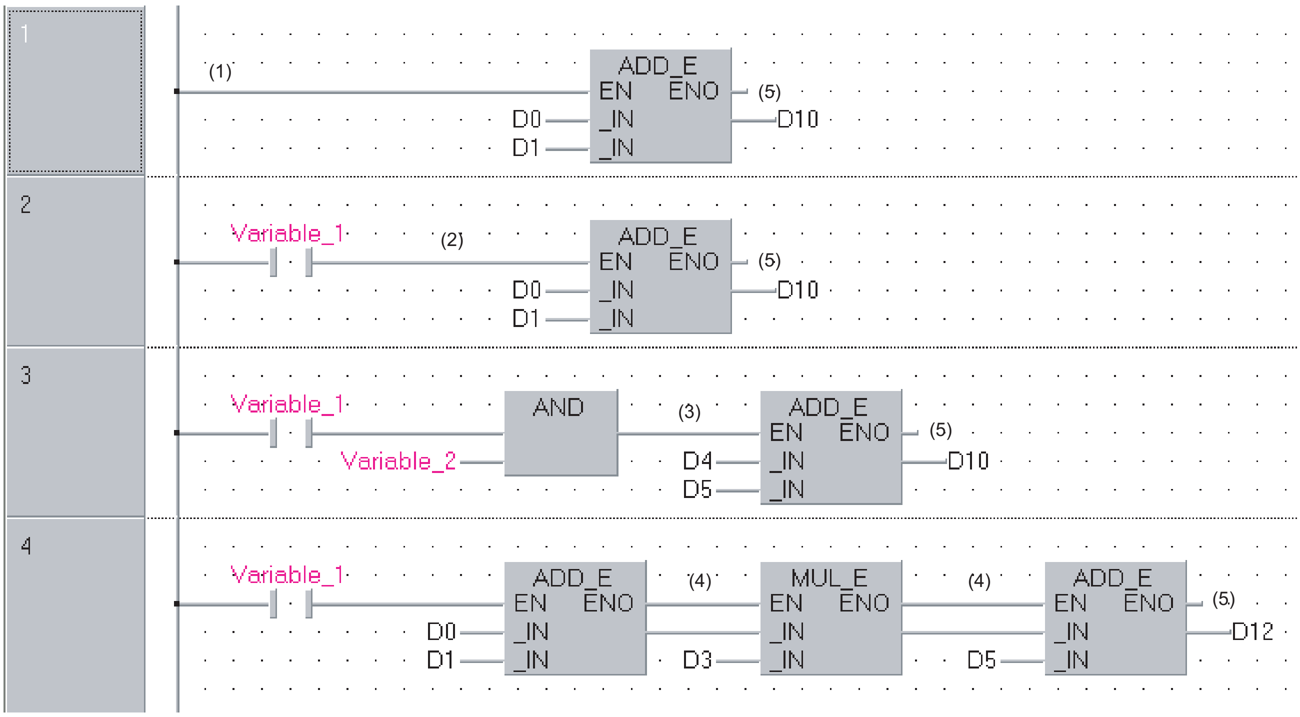
| EN | ENO | Operation result |
|---|---|---|
| TRUE (Operation execution) | TRUE (No operation error) | Operation output value |
| | FALSE (Operation error) | Undefined value |
| FALSE (Operation stop) | FALSE | Undefined value |

**Point**

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

**Ex.**

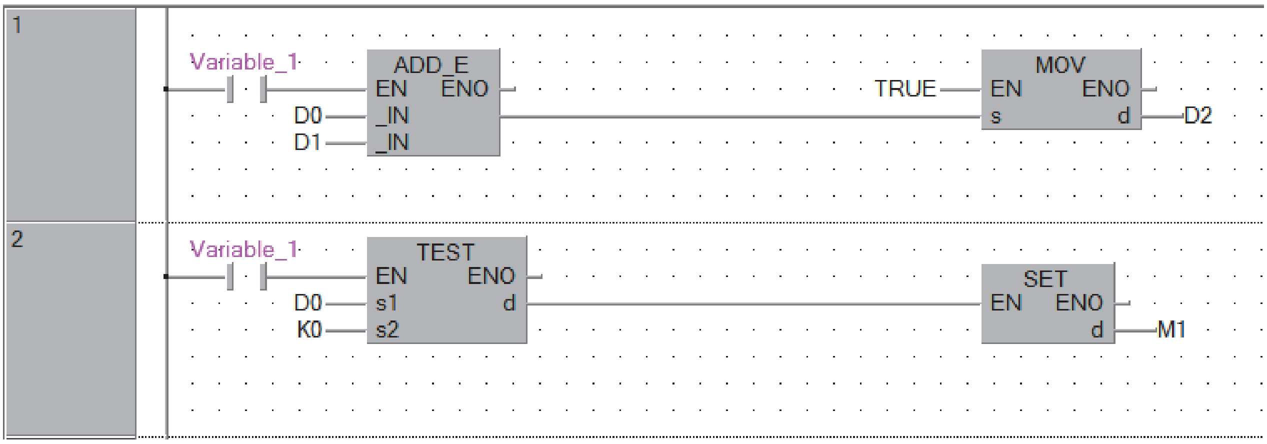
Usage example of EN and ENO



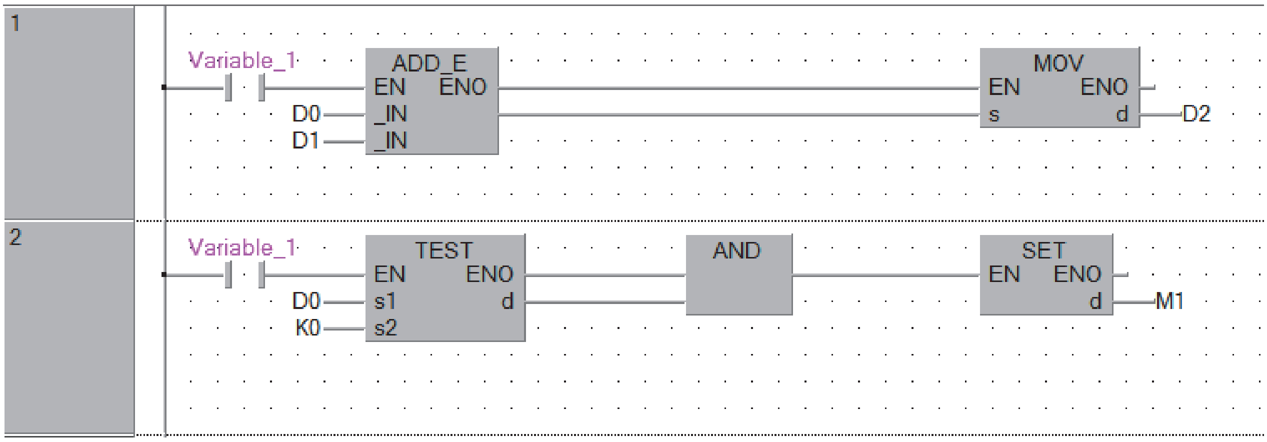

| No. | Control description |
|---|---|
| (1) | When the EN input is directly connected from the left power rail, the EN input is always TRUE and the instruction is always executed. If the ADD_E instruction is used in this manner, the operation result is the same as the ADD instruction without the EN 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 result of the instruction is not output. |

## Precautions

The following example shows that the operation results an undefined value.




When Variable_1 is OFF, the MOV or SET instruction is executed though the ADD_E or TEST instruction is not executed. Even though Variable_1 is OFF, a value may be set in D2 by the MOV instruction or M1 may turn ON by the SET instruction.




Input ENO of the first instruction to EN of the next instruction not to perform the sequential operation when EN is OFF.

# 4.3 Labels

Labels include global labels and local labels.

## Global labels

The global labels are labels that can be used in programs 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.

## 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.

# 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.

| Class | Description | Applicable POU | | |
|---|---|---|---|---|
| | | Program | Function | Function block |
| VAR_GLOBAL | Common label that can be used in programs and function blocks | ○ | × | ○ |
| VAR_GLOBAL_CONSTANT | Common constant that can be used in programs 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. | × | × | ○ |

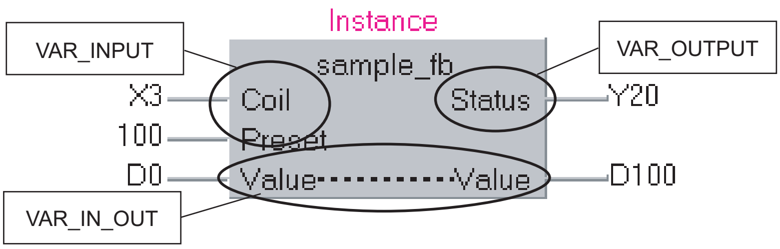
*1 Not supported by FXCPU.

**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.

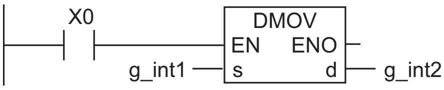
VAR_IN_OUT can be used for both input and output variables.



# Setting labels

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

The following describes setting examples of the arguments g_int1 and g_int2 of the DMOV instruction.




Ex.

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 |
|---|---|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | g_int1 | Word[Signed] | ... | | D0 | %MW0.0 | |
| 2 | VAR_GLOBAL | g_int2 | Word[Signed] | ... | | D10 | %MW0.10 | |
| 3 | | | | ... | | | | |

Ex.

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_int1 | Word[Signed] | ... | | | | |
| 2 | VAR | g_int2 | Word[Signed] | ... | | | | |
| 3 | | | | ... | | | | |

# Data types

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

The following data types are available.

- Elementary data types
- Generic data types

## 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).

| Elementary data type | Description | Value range | Bit length |
|---|---|---|---|
| Bit | Boolean | 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[*2] | Real | $-2^{128}$ to $-2^{-126}$, 0, $2^{-126}$ to $2^{128}$ | 32 bits |
| Double-precision real[*3] | Double-precision real | $-2^{1024}$ to $-2^{-1022}$, 0, $2^{-1022}$ to $2^{1024}$ | 64 bits |
| String[*4] | Character string | Maximum 255 characters | Variable |
| Time[*5] | Time value | T#-24d20h31m23s648ms to T#24d20h31m23s647ms | 32 bits |

*1 The following data types cannot be used for the structured ladder/FBD/ST language. 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 The $FX_{3S}$, $FX_{3G}$, $FX_{3GC}$, $FX_{3U}$, and $FX_{3UC}$ support this data type.

*3 The Universal model QCPU and the LCPU support this data type.

*4 The $FX_{3U}$ and $FX_{3UC}$ support this data type.

*5 This data type is used in time type operation instructions of application function. For details of the application functions, refer to the following.
📖 MELSEC-Q/L Structured Programming Manual (Application Functions)
📖 FXCPU Structured Programming Manual [Application Functions]

## Generic data types

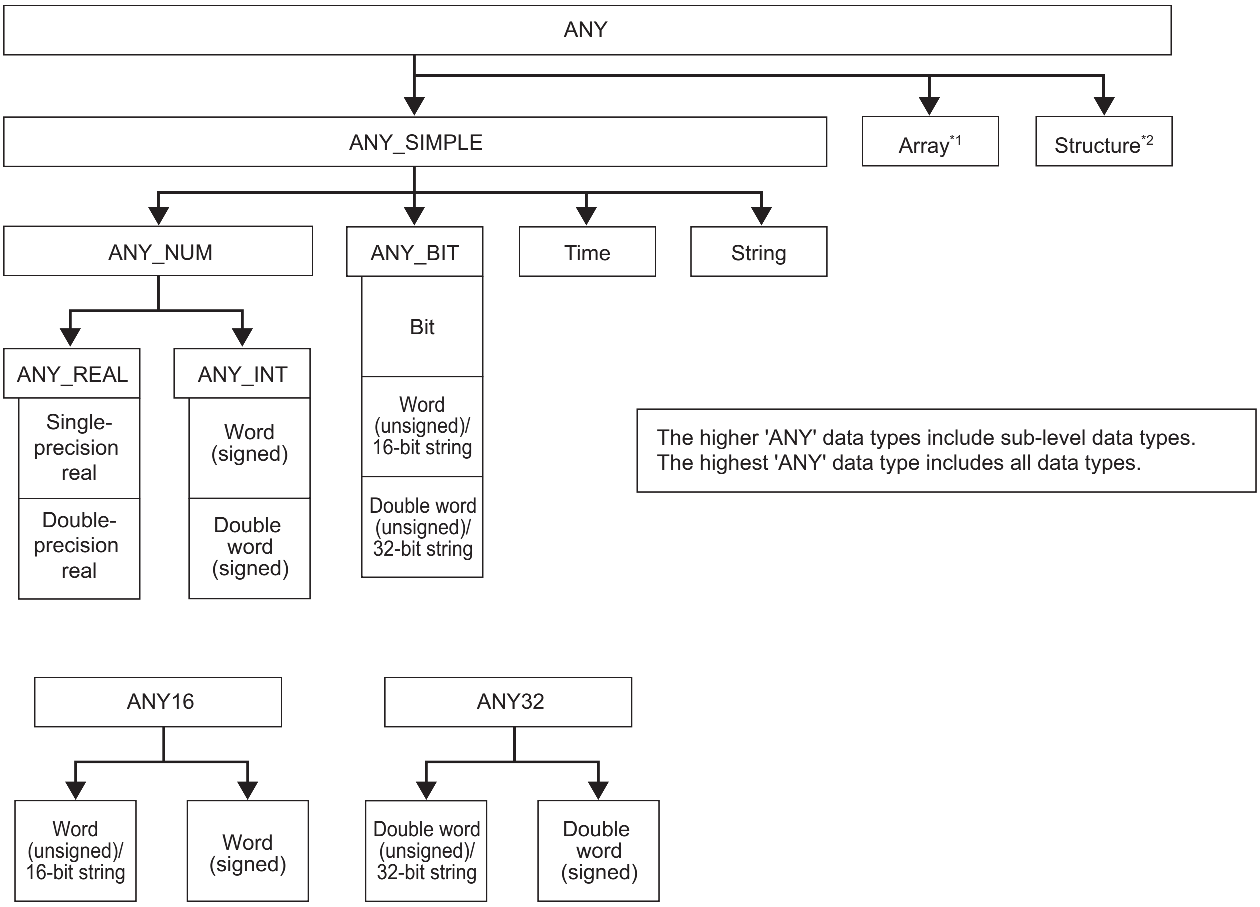
Generic data type is the data type of labels summarizing some elementary data types. Data type name starts with 'ANY'. ANY data types are used when multiple data types are allowed for function arguments and return values.

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.




*1 Arrays (☞ Page 48 Arrays)

*2 Structures (☞ Page 51 Structures)

## Expressing methods of constants

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

| 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.<br>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,<br>T#1d2h3m4s5ms |

# 4.4 Method for Specifying Data

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

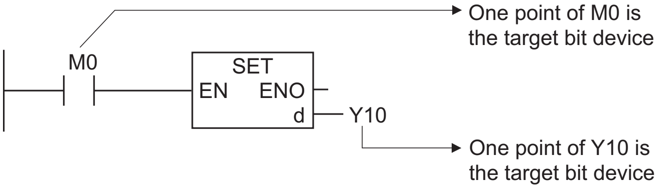
| Data that can be handled by CPU module | | | Reference |
|---|---|---|---|
| Bit data | | | Page 36 Bit data |
| Numeric data | Integer data | Word (Signed) data | Page 37 Word (16 bits) data |
| | | Double word (Signed) data | Page 39 Double word (32 bits) data |
| | Real number data | Single-precision real data | Page 42 Single-precision real (single-precision floating-point data) |
| | | Double-precision real data | Page 43 Double-precision real (double-precision floating-point data) |
| Character string data | | | Page 46 String data |
| Time data | | | Page 47 Time data |

# Bit data

Bit data are data handled in units of 1 bit, such as contacts and coils.
'Bit devices' and 'bit-specified word device' can be used as bit data.

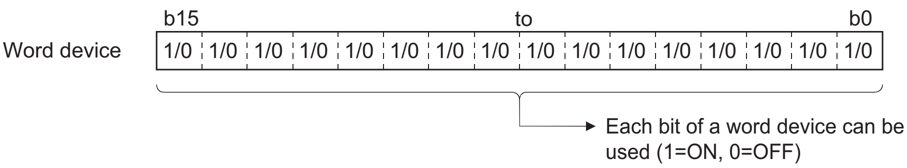
## Using bit devices

A bit device is specified in unit of one point.

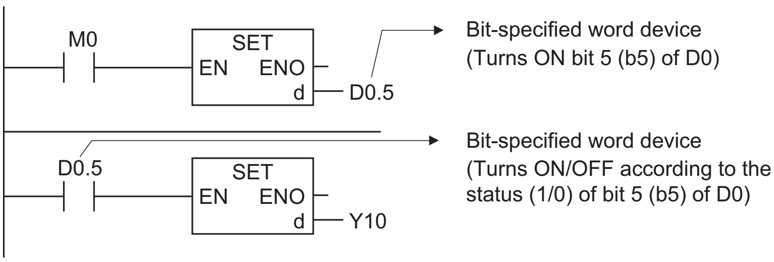



## Using word devices

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




Specify a bit device of 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).
(Example: Z0.0 is not available).




**Point**

For FXCPU, bit specification of a word device can be used for $FX_{3U}$ and $FX_{3UC}$.

## 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: K-32768 to K32767
- Hexadecimal constants: H0000 to HFFFF

For word data, word devices and digit-specified bit device 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) for the CPU module used.)



### Using bit devices

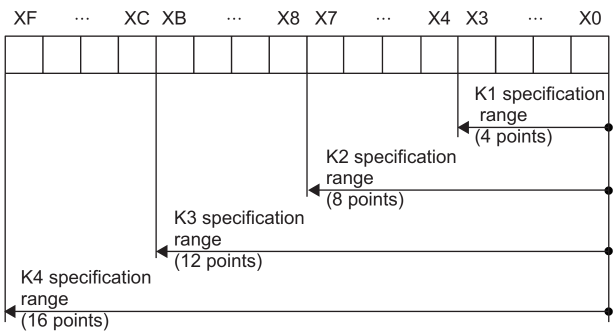
By specifying digits of bit devices, word data can be used.
Specify digits of bit data as "[Number of digits][Start number of bit device]".
Digits can be specified in the range from K1 to K4 in unit of 4 points (4 bits).
(For a link direct device, specify as "J[Network No.]\[Number of digits][Start number 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 digits are specified for X0.

- QCPU (Q mode)/LCPU

| Digit specification | Number of target points |
|---|---|
| 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 |

- For FXCPU, the device numbers of input/output (X, Y) is assigned in octal.

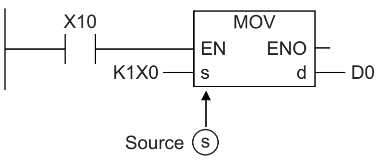
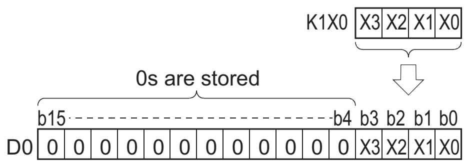
| Digit specification | Number of target points |
|---|---|
| K1X0 | 4 points of X0 to X3 |
| K2X0 | 8 points of X0 to X7 |
| K3X0 | 12 points of X0 to X13 |
| K4X0 | 16 points of X0 to X17 |



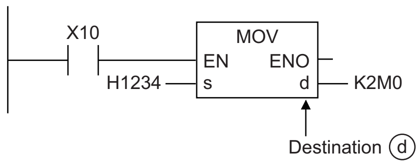
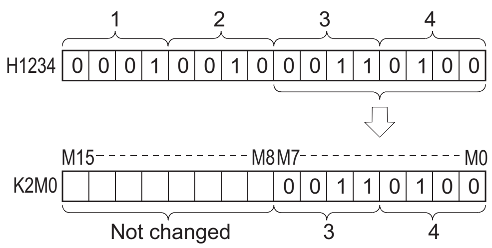
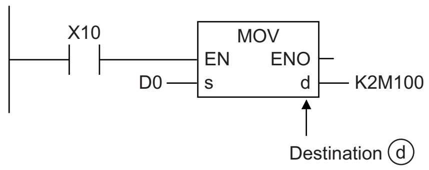
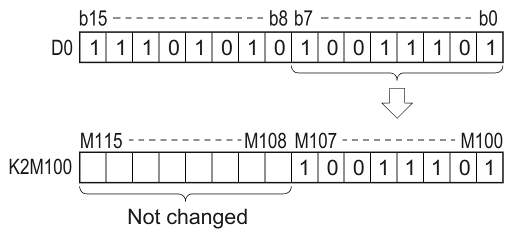
The following table shows the numeric values that can be used as source data when digits are specified at the source (s).

| 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 |

When destination (D) data is a word device, the word device for the destination becomes 0 following the bit designated by digit designation at the source.

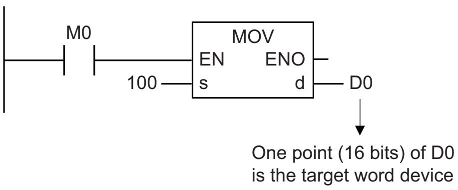
| Ladder example | Processing |
|---|---|
| • Instruction that processes 16-bit data  |  |

When digits are specified at the destination (d), the points by the specified digit are the target of destination.
The status of bit devices which follow the digit-specified bit devices is not changed.

| Ladder example | Processing |
|---|---|
| • When the source (s) is a numeric value  |  |
| • When the source (s) is a word device  |  |

## Using word devices

A word device is specified in unit of one point (16 bits).




**Point**

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

## 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: K-2147483648 to K2147483647
- Hexadecimal constants: H00000000 to HFFFFFFFF

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).

### Using bit devices

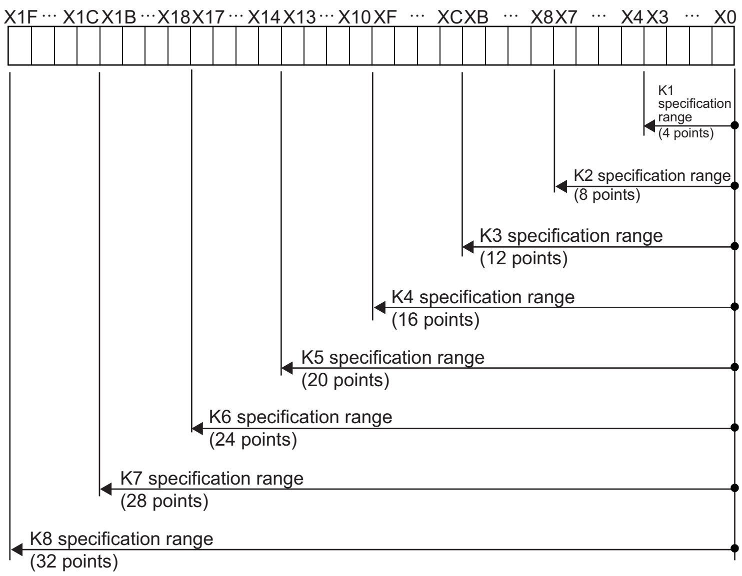
By specifying digits of bit devices, double word data can be used.
Specify digits of bit data as "[Number of digits][Start number of bit device]".
(For a link direct device, specify as "J[Network No.]\[Number of digits][Start number of bit device]". To specify X100 to X11F of Network No.2, specify as J2\K8X100.) Digits 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 digits are specified for X0.

- QCPU (Q mode)/LCPU

| Digit specification | Number of target points | Digit specification | Number of target points |
|---|---|---|---|
| K1X0 | 4 points of X0 to X3 | K5X0 | 20 points of X0 to X13 |
| K2X0 | 8 points of X0 to X7 | K6X0 | 24 points of X0 to X17 |
| K3X0 | 12 points of X0 to XB | K7X0 | 28 points of X0 to X1B |
| K4X0 | 16 points of X0 to XF | K8X0 | 32 points of X0 to X1F |

- For FXCPU, the device numbers of input/output (X, Y) is assigned in octal.

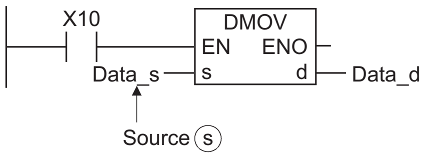
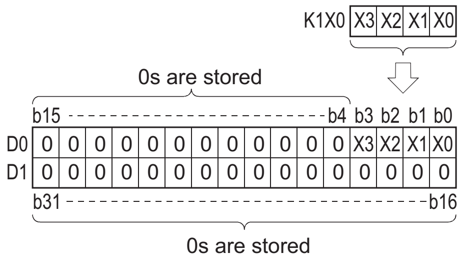
| Digit specification | Number of target points | Digit specification | Number of target points |
|---|---|---|---|
| K1X0 | 4 points of X0 to X3 | K5X0 | 20 points of X0 to X23 |
| K2X0 | 8 points of X0 to X7 | K6X0 | 24 points of X0 to X27 |
| K3X0 | 12 points of X0 to X13 | K7X0 | 28 points of X0 to X33 |
| K4X0 | 16 points of X0 to X17 | K8X0 | 32 points of X0 to X37 |



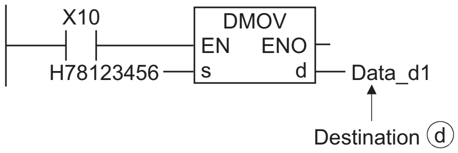
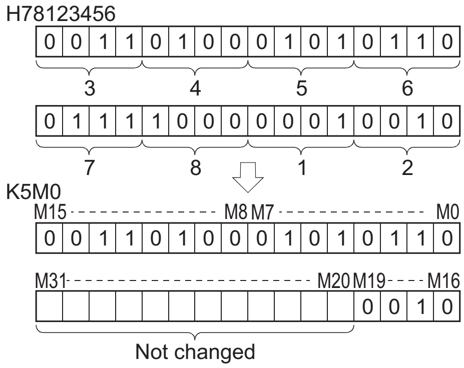
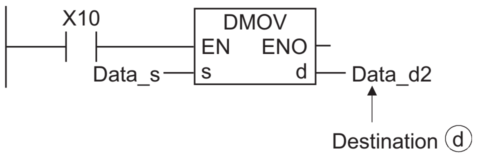
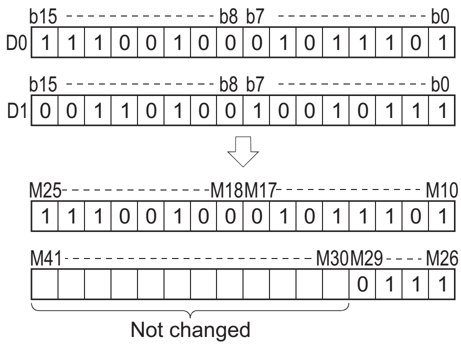
The following table shows the numeric values that can be used as source data when digits are specified at the source (s).

| 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 |

When destination (D) data is a word device, the word device for the destination becomes 0 following the bit designated by digit designation at the source. (Data_s:K1X0, Data_d:D0)

| Ladder example | Processing |
|---|---|
| • Instruction that processes 32-bit data   |   |

When digits are specified at the destination (d), the points by the specified digit are the target of destination. (Data_d1:K5M0, Data_d2:K5M10, Data_s:D0) Bit devices below the number of points designated as digits do not change.

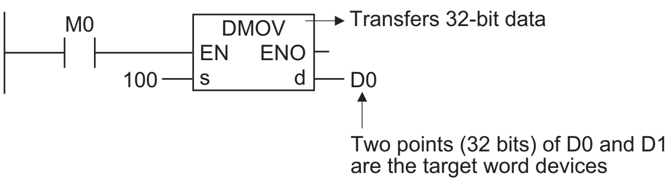
| Ladder example | Processing |
|---|---|
| • When the source (s) is a numeric value   |   |
| • When the source (s) is a word device   |   |

**Point**

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

## 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.

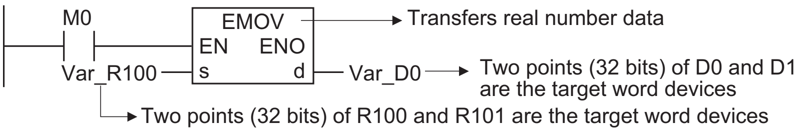


# Single-precision real/double-precision real data

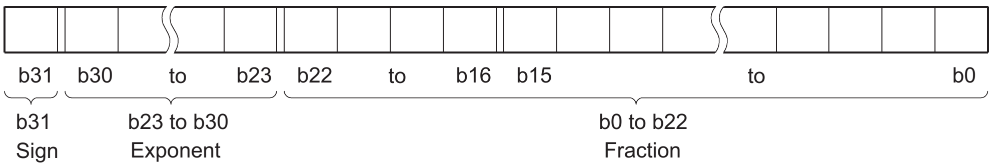
Single-precision real/double-precision real data are 32-bit floating-point data used in basic instructions and application instructions.
For FXCPU, double-precision real data is not supported.
Real number data can be stored only in word devices.

## 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'.




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.




- 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.

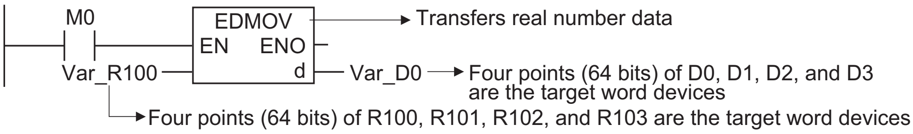
| b23 to b30 | FFH | FEH | FDH | | | 81H | 80H | 7FH | 7EH | | | 02H | 01H | 00H |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | Not used | 127 | 126 | | | 2 | 1 | 0 | -1 | | | -125 | -126 | Not used |

- Mantissa: Each of the 23 bits, b0 to b22, represents the "XXXXXX..." portion when the data is represented in binary, "1.XXXXXX...".

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

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

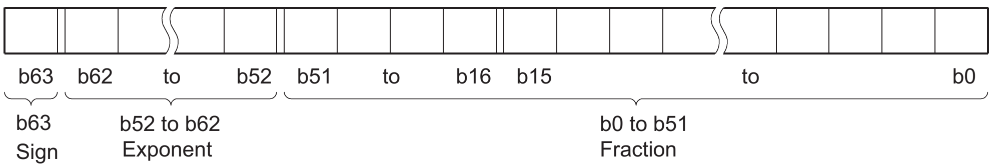
Real number data are stored in 64 bits of 'specified device number' and 'specified device number + 3'.




Floating-point data are represented by four word devices.

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

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




- Sign: The most significant bit, b63, is the sign bit.
  - 0: Positive
  - 1: Negative
- Exponent: The 11 bits, b52 to b62, represent the excess 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 | | 1 | 0 | -1 | -2 | -3 | | -1021 | -1022 | Not used |

- Mantissa: Each of the 52 bits, b0 to b51, represents the "XXXXXX..." portion when the data is represented in binary, "1.XXXXXX...".

## Precautions

Precautions when an input value of a single/double-precision real number is set using a programming tool are shown below.

### ■Single-precision real

Single-precision real data are processed as 32-bit single precision in the programming tool, and thus the number of significant figures becomes approximately 7. If the input value of single-precision real data exceeds 7 digits, the 8th digit is rounded. If the value after the rounding exceeds a value between -2147483648 and 2147483647, an operation error occurs.

Example 1: When '2147483647' is set for the input value

8th digit '6' is rounded.
The value is handled as '2147484000'.

Example 2: When 'E1.1754943562' is set for the input value

8th digit '3' is rounded.
The value is handled as 'E1.175494'.

### ■Double-precision real

Double-precision real data are processed as 64-bit double precision in the programming tool, and thus the number of significant figures becomes approximately 15. If the input value of double-precision real data exceeds 15 digits, the 16th digit is rounded. If the value after the rounding exceeds a value between -2147483648 and 2147483647, an operation error occurs.

Example 1: When '2147483646.12345678' is set for the input value

16th digit '6' is rounded.
The value is handled as '2147483646.12346'.

Example 2: When 'E1.7976931348623157+307' is set for the input value

16th digit '5' is rounded.
The value is handled as 'E1.79769313486232+307'.

Point

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

To express 0 in floating-point data, set all of the following bits to 0.

- Single-precision floating-point data: b0 to b31
- Double-precision floating-point data: b0 to b63

The setting range of floating decimal point data is as follows.*1

- Single-precision floating-point data: $-2^{128}$ < Device data $\leq -2^{-126}$, 0, $2^{-126} \leq$ Device data < $2^{128}$
- Double-precision floating-point data: $-2^{1024}$ < Device data $\leq -2^{-1022}$, 0, $2^{-1022} \leq$ Device data < $2^{1024}$

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 the following CPU modules, a floating-point operation does not result an error since -0 is converted to 0 in a CPU module 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.)
- Universal model QCPU (QnUDVCPU and QnUDPVCPU only)

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
- Process CPU
- Redundant CPU
- Universal model QCPU (excluding QnUDVCPU and QnUDPVCPU)
- LCPU
- FXCPU*4

*1 For operations when an overflow or underflow is occurred, or when a special value is input, refer to the following manuals.
QCPU (Q mode)/LCPU
User's Manuals (Function Explanation, Program Fundamentals) for the CPU module used.
FXCPU
User's manuals and Programming Manuals for the FXCPU used

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

*3 The floating point operation is supported with the Basic model QCPU with a serial number whose first five digits are '04122' or higher.

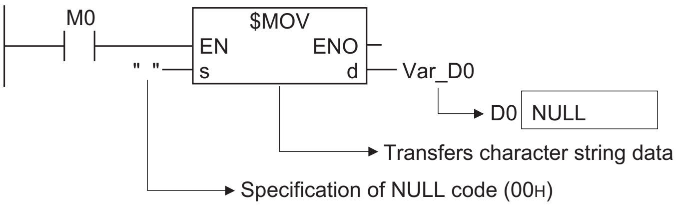
*4 Only the $FX_{2N}$, $FX_{2NC}$, $FX_{3S}$, $FX_{3G}$, $FX_{3GC}$, $FX_{3U}$, and $FX_{3UC}$ support floating point operations.

# 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.

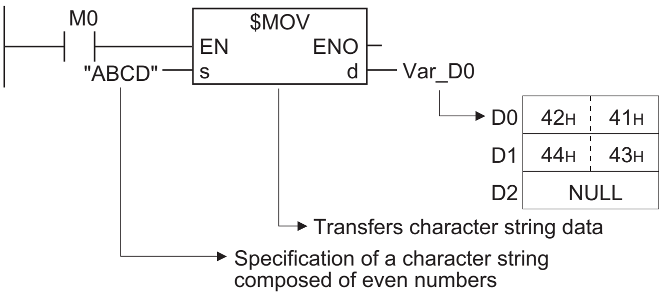
## When the specified character is NULL code

The NULL code is stored by using one word.



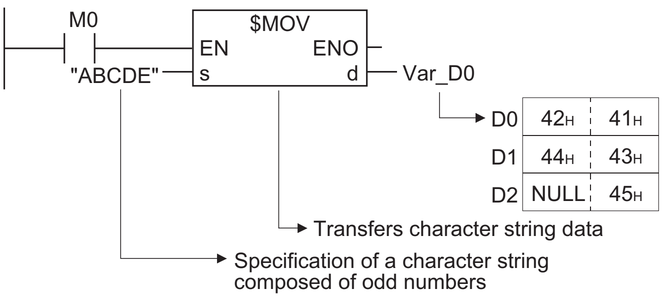

## When the number of characters is an even number

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).



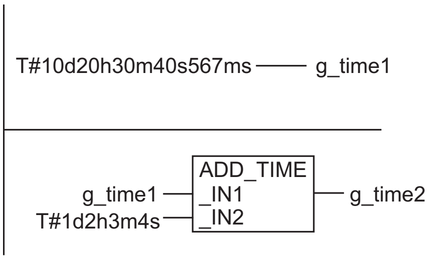

## When the number of characters is an odd number

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).



# 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".




Each value of time data can be specified within the following range.

| 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]

# Arrays

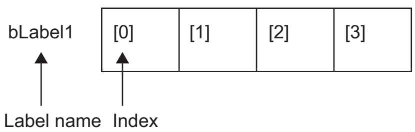
An array represents a consecutive aggregation of same data type labels.

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

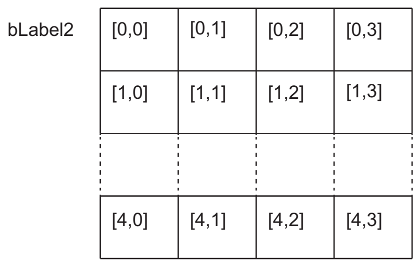
(📖 GX Works2 Version 1 Operating Manual (Structured Project))

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

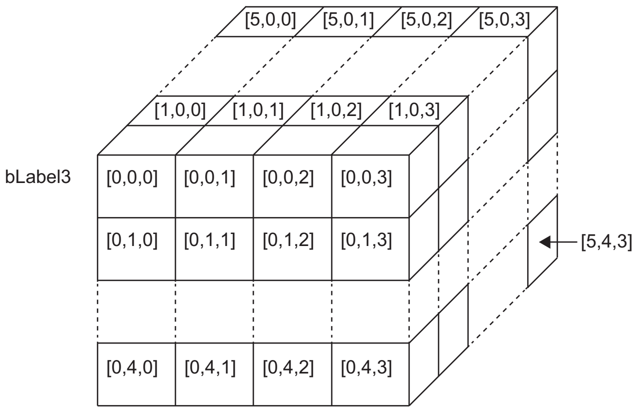
• One-dimensional array (Number of elements: 4)




• Two-dimensional array (Number of elements: 5 × 4)




• Three-dimensional array (Number of elements: 6 × 5 × 4)




• Settings in the programming tool

| | Class | Label Name | Data Type | |
|---|---|---|---|---|
| 1 | VAR | bLabel1 | Bit(0..3) | ... |
| 2 | VAR | bLabel2 | Bit(0..4,0..3) | ... |
| 3 | VAR | bLabel3 | Bit(0..5,0..4,0..3) | ... |

## Defining arrays

### ■Dimension number of multidimensional array

Up to three-dimensional array can be defined.

### ■Definition format

The following table lists definition format.

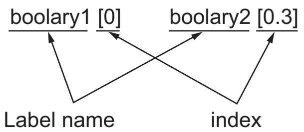
| Number of array dimensions | Format | Remarks |
|---|---|---|
| One dimension | Array of elementary data type/structure name (array start value .. array end value) | For elementary data types ☞ Page 32 Data types For structured data types ☞ Page 51 Structures |
| | (Definition example) Bit (0..2) | |
| Two dimensions | Array of elementary data type/structure name (array start value .. array end value, array start value .. array end value) | |
| | (Definition example) Bit (0..2, 0..1) | |
| Three dimensions | Array of elementary data type/structure name (array start value .. array end value, array start value .. array end value, array start value .. array end value) | |
| | (Definition example) Bit (0..2, 0..1, 0..3) | |

## 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.

For an array with two or more dimensions, delimit indexes in '[ ]' by ','.

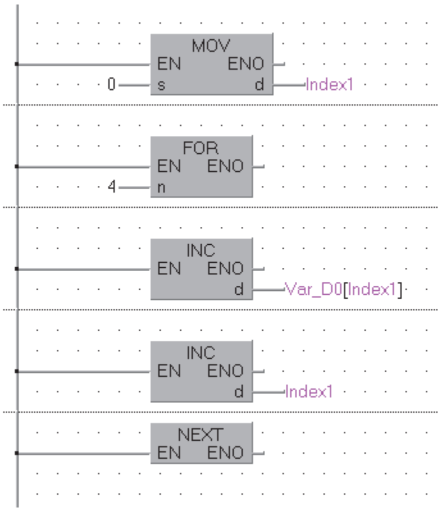
Example)




For the ST and structured ladder/FBD languages, labels (word (signed) or double word (signed) data type) can be used for indexes as shown below.

Note that Z0 or Z1 cannot be used in the programs if labels are used for indexes.

[Structured ladder/FBD]




[ST]

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

### ■Precautions

The following explains precautions for the index of an array.

- When a label or a device is specified for an array index, the operation is performed with a combination of multiple sequence instructions. Therefore, if an interruption occurs during the operation of the array label, an unintended operation result may be produced. When using interrupt programs, use interrupt disable/enable instructions (DI/EI instructions) as necessary.
- If the index which is outside of the defined range is specified for an array index*1, any of the following operations occur.
  - An operation error occurs.
  - A current value of other label is referred or written.

*1 For example, a value other than the value within 0 to 2 is used for the index of an array which is declared with the bit array (0..2).

## Maximum number of array elements

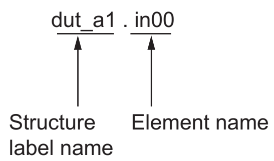
The maximum number of array elements differs depending on data types as shown below.

| 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 |

## 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 Version 1 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.

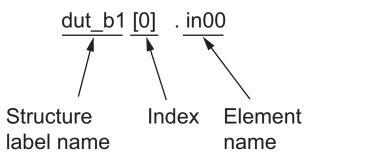
Example) When using the element
of the structured data

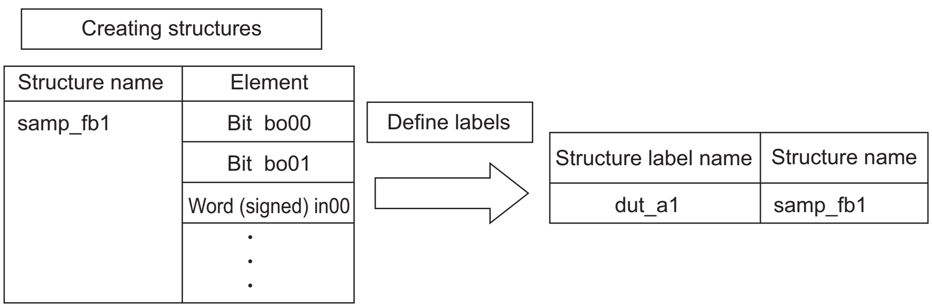


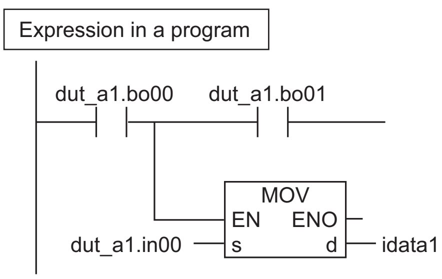



Structures can also be used as arrays. When a structure is declared as an array, append an index enclosed by '[ ]' after the structure label name. When arrays are used and accessed using array indices to specify a label or device, the maximum value in an array is 32767.
The arranged structured data can be specified as arguments of functions and function blocks. When arrays are used and accessed using array indices to specify a label or device, a bit-specified word device cannot be specified for a bit type element.

Example) When using the element
of the arranged structured data



# 4.5 Device and Address

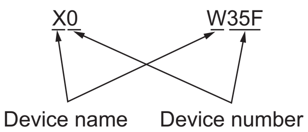
This section explains the method for expressing programmable controller CPU devices. The following two types of format are available.

- Device: This format consists of a device name and a device number.
- Address: A format defined in IEC 61131-3. In this format, a device name starts with %.

## Device

Device is a format that uses a device name and a device number.

Example)




For details of devices, refer to the following manuals.

User's Manual (Function Explanation, Program Fundamentals) for the CPU module used.

FXCPU Structured Programming Manual [Device & Common]

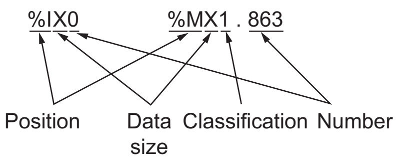
# Address

Address is a format defined in IEC 61131-3. The following table shows details of format that conforms to IEC 61131-3.

| Start | 1st character: position | 2nd character: data size | | 3rd character and later: classification | Number |
|---|---|---|---|---|---|
| % | I: Input<br>Q: Output<br>M: Internal | (Omitted) | Bit | Numeric characters 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) |
| | | X | Bit | | |
| | | W | Word (16 bits) | | |
| | | D | Double word (32 bits) | | |
| | | L | Long word (64 bits)*1 | | |

*1 Not supported by FXCPU.

Example)




## 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.

| Device | Position |
|---|---|
| X, J\X<br>(X device) | I (input) |
| Y, J\Y<br>(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.

| Device | Data size |
|---|---|
| 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.

☞ Page 54 Correspondence between devices and addresses

**Point**

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

# Correspondence between devices and addresses

This section explains the correspondence between devices and addresses.

## Correspondence between devices and addresses

The following table shows the correspondence between devices and addresses.

### ■QCPU (Q mode)/LCPU

| 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<br>%MD3.n | TN511<br>T511 | %MW3.511<br>%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<br>%MD4.n | CN511<br>C511 | %MW4.511<br>%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<br>%MD13.n | STN511<br>ST511 | %MW13.511<br>%MD13.511 |
| Data register | | D | Dn | %MW0.n<br>%MD0.n | D11135 | %MW0.11135<br>%MD0.11135 |
| Special register | | SD | SDn | %MW10.n<br>%MD10.n | SD1023 | %MW10.1023<br>%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<br>%MD1.n | W7FF | %MW1.2047<br>%MD1.2047 |
| Link special register | | SW | SWn | %MW11.n<br>%MD11.n | SW3FF | %MW11.1023<br>%MD11.1023 |
| Intelligent function module device | | G | Ux\Gn | %MW14.x.n<br>%MD14.x.n | U0\G65535 | %MW14.0.65535<br>%MD14.0.65535 |
| File register | | R | Rn | %MW2.n<br>%MD2.n | R32767 | %MW2.32767<br>%MD2.32767 |
| Pointer | | P | Pn | "" (Null character) | P299 | None |
| Interrupt pointer | | I | In | None | — | — |
| Nesting | | N | Nn | None | — | — |
| Index register | | Z | Zn | %MW7.n<br>%MD7.n | Z9 | %MW7.9<br>%MD7.9 |
| Step relay | | S | Sn | %MX2.n | S127 | %MX2.127 |
| SFC transition device | | TR | TRn | %MX18.n | TR3 | %MX18.3 |

| Device | | Expressing method | | Example of correspondence between device and address | |
|---|---|---|---|---|---|
| | | Device | Address | Device | Address |
| 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<br>%MD16.x.1.n | J2\W3FFF | %MW16.2.1.16383<br>%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<br>%MD16.x.11.n | J2\SW1FF | %MW16.2.11.511 |
| File register | ZR | ZRn | %MW12.n<br>%MD12.n | ZR32767 | %MW12.32767<br>%MD12.32767 |

### ■FXCPU

| 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<br>%MD3.n | TN191<br>T190 | %MW3.191<br>%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<br>%MD4.n | CN99<br>C98 | %MW4.99<br>%MD4.98 |
| Data register | | D | Dn | %MW0.n<br>%MD0.n | D199<br>D198 | %MW0.199<br>%MD0.198 |
| Intelligent function module device | | G | Ux\Gn | %MW14.x.n<br>%MD14.x.n | U0\G09 | %MW14.0.10<br>%MD14.0.9 |
| Extension register | | R | Rn | %MW2.n<br>%MD2.n | R32767<br>R32766 | %MW2.32767<br>%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<br>%MD7.n | Z7<br>Z6 | %MW7.7<br>%MD7.6 |
| | | V | Vn | %MV6.n | V7 | %MW6.7 |
| State | | S | Sn | %MX2.n | S4095 | %MX2.4095 |

## Digit specification of bit devices

The following table shows the correspondence between devices and addresses when specifying digits of bit devices.

| 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]<br>(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 |

## Bit specification of word device

The following table shows the correspondence between devices and addresses when specifying a bit device of word device.

| 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 of bit devices, and bit specification of word device

Index setting, digit specification of bit devices, and bit specification of word device cannot be applied to labels.

# 4.6 Index Setting

## Overview of the index setting

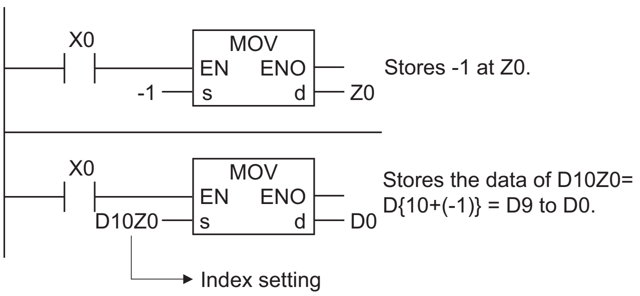
The index setting is an indirect setting that uses index registers.
When the index setting is used in a sequence program, the device consists of "directly specified device number" + "content of index register".
For example, when D2Z2 is specified and the value of Z2 is 3, D(2+3)=D5 is set as the target.
For Universal model QCPU, LCPU, and FXCPU, indexes can be set in 32-bit range in addition to 16-bit range.



## 16-bit index setting

### ■Setting an index in 16-bit range

Values from -32768 to 32767 can be set to index registers.*1
The following shows how the index is set.




*1 For the specifications of the index register, refer to the User's Manual (Function Explanation, Program Fundamentals) for the CPU module used.

### ■Devices that can be used for the index setting (for QCPU (Q mode), LCPU)

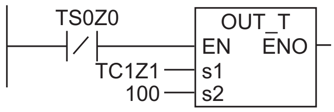
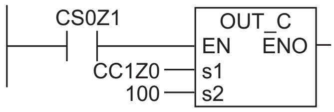
The index setting can be applied to devices used by contacts, coils, basic instructions, and application instructions except for the restrictions listed in the tables below. The index setting cannot be applied to labels.

- Devices that cannot be used for the index setting

| Device | Description |
|---|---|
| E | Floating-point data |
| $ | Character string data |
| □.□ (D0.1 etc.) | Bit-specified word device |
| FX, FY, FD | Function devices |
| P | Pointers used as labels |
| I | Interrupt pointers used as labels |
| Z | Index registers |
| S | Step relays*2 |
| TR | SFC transition devices*1 |
| BL | SFC block devices*1*2 |

*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 The SFC block devices (BL) and step relays (S) of the High-speed Universal model QCPU and Universal model Process CPU can be used for the index modification under the following ranges.
· SFC block device (BL): BL0 to BL319
· Step relay (S): Within the range set in the parameter (device settings)
When the step relays (S) in an SFC block device are selected, S0 to S511 can be used for the index setting.

• Devices with restrictions on index registers*3

| Device | Description | Example |
|---|---|---|
| T, ST | • 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. |  |

*3 The High-speed Universal model QCPU and Universal model Process CPU are excluded.

## ■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 |

• Devices with restrictions on index registers

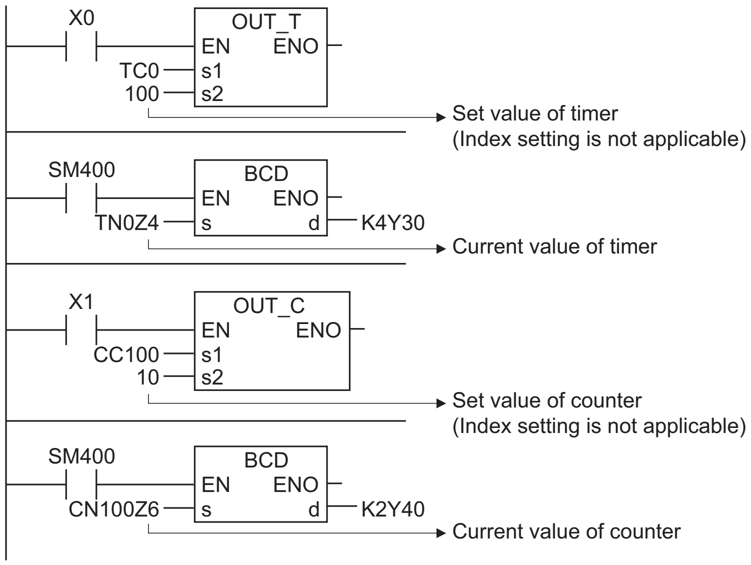
When using FXCPU, note the following precautions.

The index setting for devices used in the basic instructions is available for FX$_{3U}$ and FX$_{3UC}$ only.

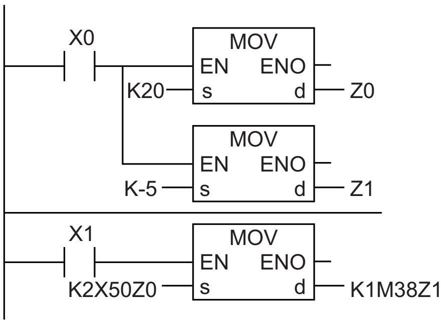
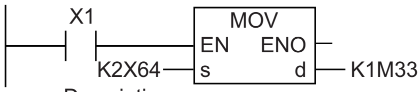
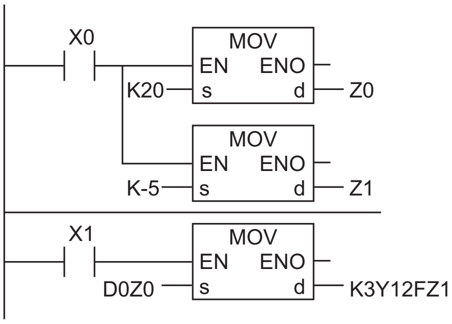
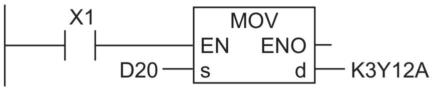
The index setting cannot be applied to 32-bit counter and special auxiliary relay.

**Point**

There are no usage restrictions on index register numbers for current values of the timer and counter.



■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 |
|---|---|
|  | <br>Description<br>K2X50Z0 ······ K2X(50 + 14) = K2X64<br>Converts K20 to a hexadecimal number.<br>K1M38Z1 ······ K1M(38 - 5) = K1M33 |
|  | <br>Description<br>D0Z0 ········· D (0 + 20) = D20<br>K3Y12FZ1 ······ K3Y(12F - 5) = K3Y12A<br>Hexadecimal number |

## 32-bit index setting

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**

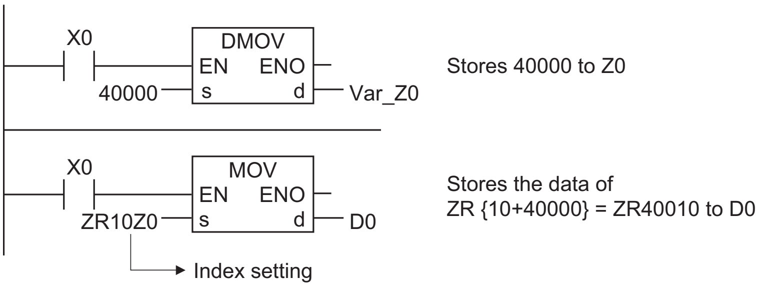
32-bit index settings using 'ZZ' can be used for the following CPU modules only. For the usable programming tool, refer to the operating manual of the programming tool.

- QnU(D)(H)CPU with a serial number whose first five digits are '10042' or higher (excluding Q00UJCPU)
- Built-in Ethernet port QCPU
- LCPU

### ■Specifying a range of index registers used for a 32-bit index setting

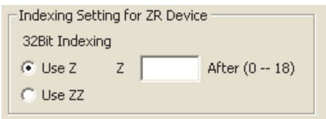
Values from -2147483648 to 2147483647 can be set to index registers.

The following shows how the index is set.




- Specification method

When setting indexes in 32-bit range, specify the start number of index registers to be used in the device setting of the PLC parameter.




**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.

- 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 |

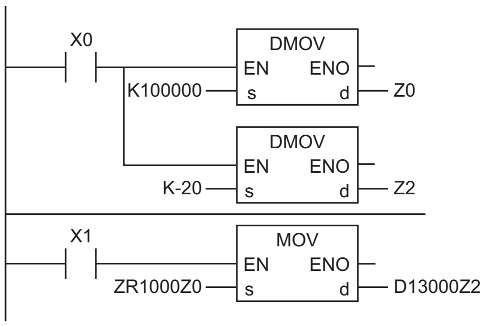
- Usage range of index registers

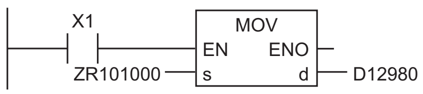
The following table lists the usage range of index registers when setting indexes in 32-bit range.

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 |

- The following figure shows the examples of index setting and their actual processing devices. (With the setting of Z0 (32 bits) = 100000 and Z2 (32 bits) = -20)

<table>
<tr><th>Ladder example</th><th>Actual processing device</th></tr>
<tr><td>


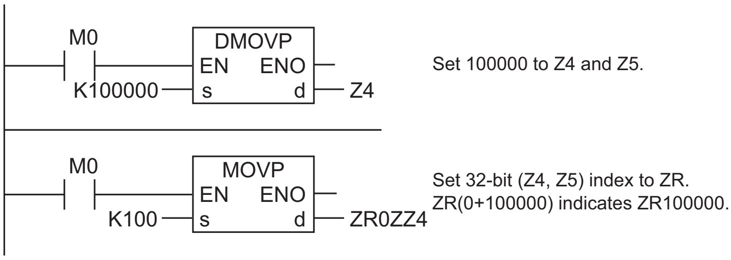
</td><td>


Description<br>
ZR1000Z0 · · · ZR(1000+100000)=ZR101000<br>
D13000Z2 · · · D(30-20)=D12980
</td></tr>
</table>

## ■Specifying a 32-bit index setting using 'ZZ'

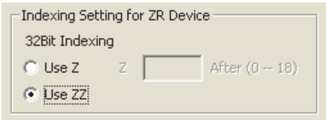
A 32-bit index can be specified to the index register by specifying an index using 'ZZ', for instance, 'ZR0ZZ4'.

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




- 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.




- Devices that can be used for the index setting

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 |
| M*1 | Internal relay |
| B*1 | Link relay |
| D*1 | Data register |
| W*1 | Link register |
| Jn\B*1 | Link relay |
| Jn\W*1 | Link register |

*1 The devices can be used for the High-speed Universal model QCPU and Universal model Process CPU only.

- Usage range of index registers

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

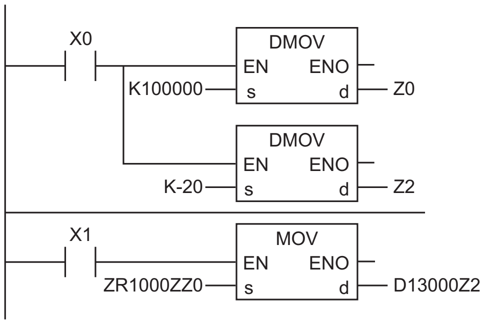
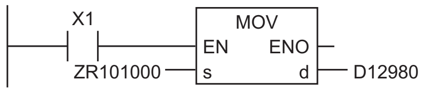
When specifying a 32-bit index setting using 'ZZ', specify a device as a form of ZRmZZn.

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

- The following figure shows the examples of 32-bit index setting using 'ZZ' and their actual processing devices. (With the setting of Z0 (32 bits) =100000 and Z2 (32 bits)=-20)

<table>
<tr><th>Ladder example</th><th>Actual processing device</th></tr>
<tr><td>
</td><td>

Description<br>ZR1000Z0 ··· ZR(1000+100000)=ZR101000<br>D13000Z2 ··· D(30-20)=D12980</td></tr>
</table>

- Functions that can use 'ZZ'

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

| No. | Description |
|---|---|
| 1 | Device specification with an instruction in a program |
| 2 | Monitoring device registrations |
| 3 | Device test |
| 4 | Device test with an execution condition |
| 5 | Setting monitoring conditions |
| 6 | Sampling trace (trace point (device specification), trace target devices) |
| 7 | Data logging function (sampling interval (device specification), logging target data) |

**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.

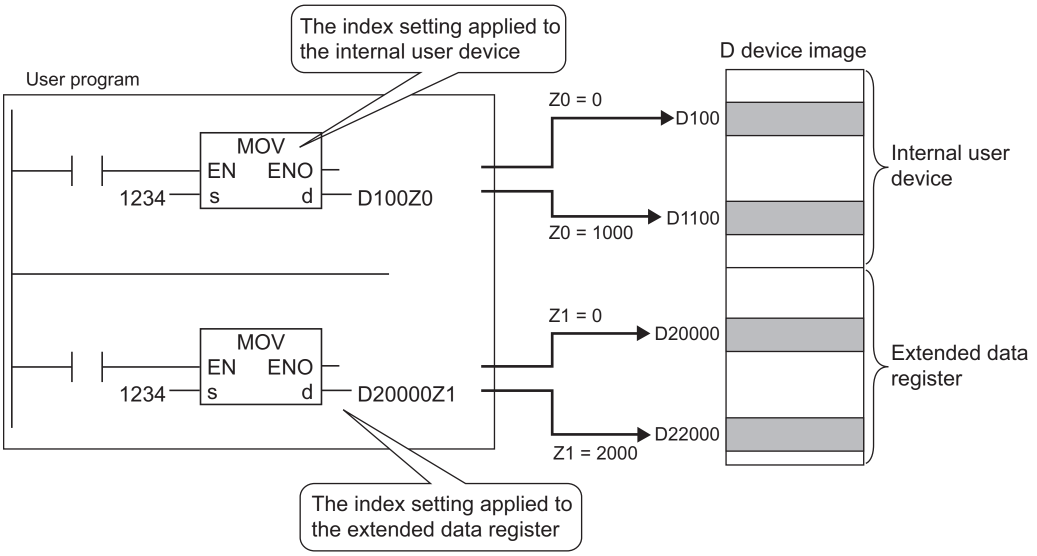
### ■32-bit index setting for FXCPU

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 |

## Applying index settings to extended data registers (D) and extended link registers (W)*1

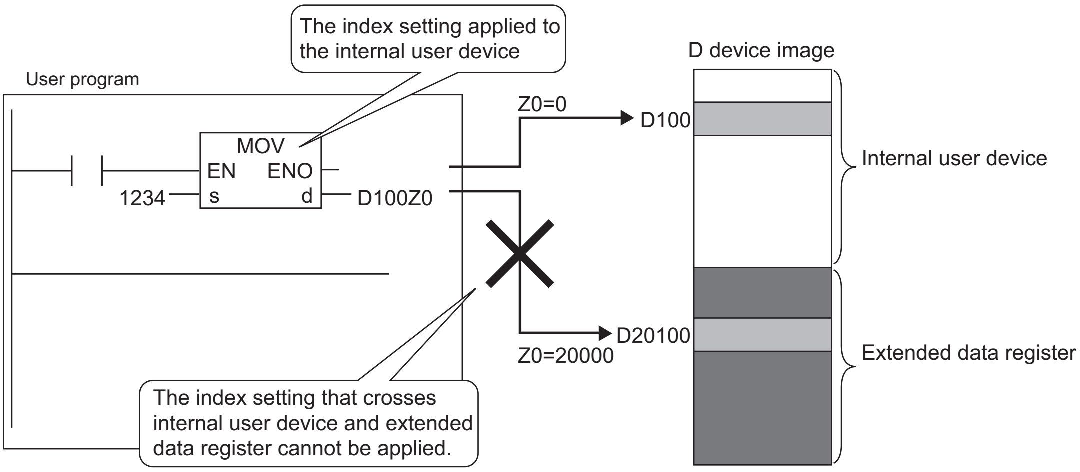
As an index setting can be applied to internal user devices, data registers (D) and link registers (W), the device specification by the index setting can be used within the range of extended data registers (D) and extended link registers (W).




*1 For Universal model QCPU (excluding Q00UJCPU), and LCPU

### ■Index settings that cross internal user devices and extended data registers (D)/extended link registers (W)

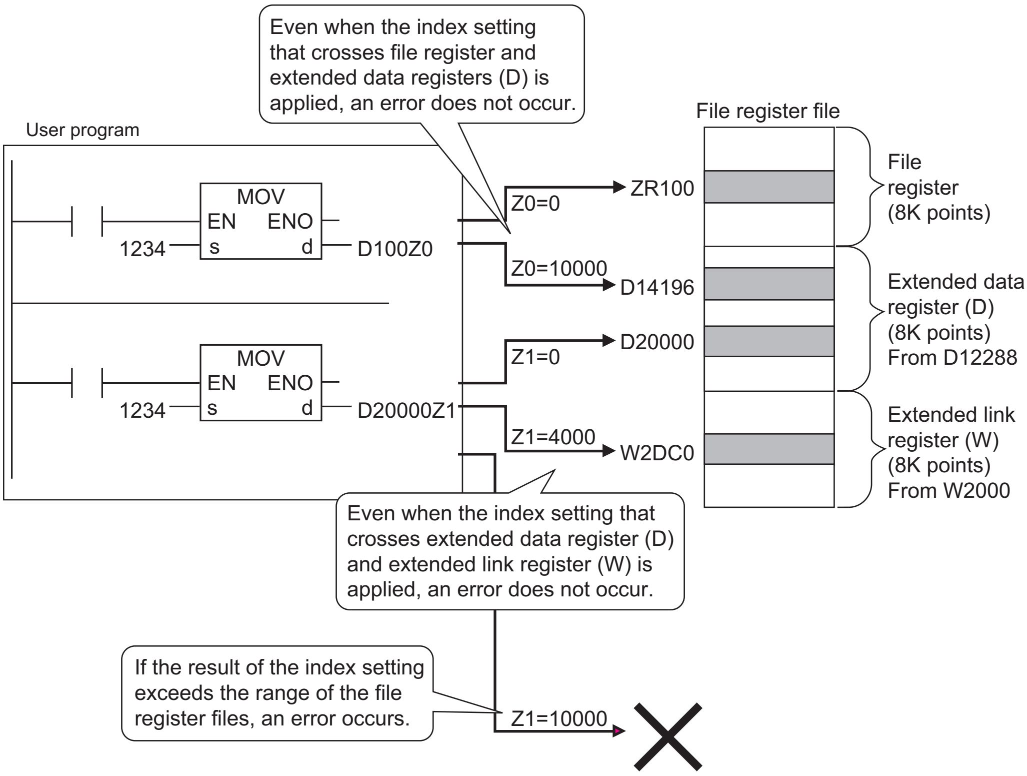
An index setting that crosses internal user devices and extended data registers (D)/extended link registers (W) cannot be applied. If the device range check is enabled at the index setting, an error occurs. (Error code: 4101)



## ■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)

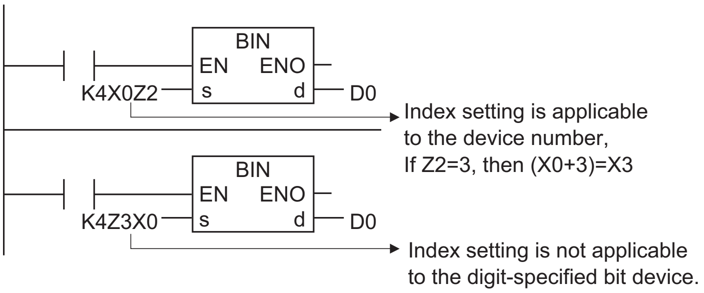



4

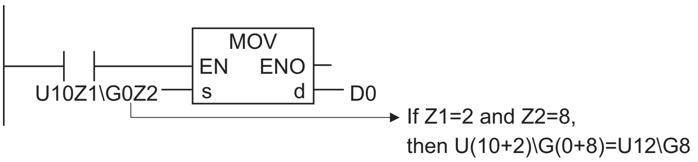
## Other applicable data

For bit data, an index setting is applicable to device numbers whose digits are specified.

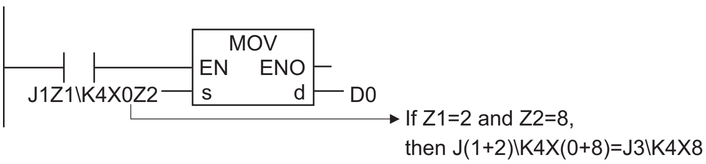
Note that an index setting is not applicable to the digit-specified bit device.



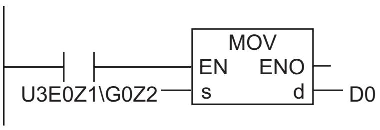

For intelligent function module devices[*1], an index setting is applicable to both start I/O numbers of the intelligent function module and buffer memory addresses.




For link direct devices[*1], an index setting is applicable to both network numbers and device numbers.




For multiple CPU area devices[*2], an index setting is applicable to both start I/O numbers of the CPU module and CPU shared memory addresses.




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.[*3]

- Specify a range of index registers used for a 32-bit index setting.
- Specify a 32-bit index setting using 'ZZ'.

*1 For intelligent function module devices and link direct devices, refer to the User's Manual (Function Explanation, Program Fundamentals) of the CPU module used.

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

*3 For Universal model QCPU (excluding Q00UJCPU), and LCPU

**Point**

32-bit index settings using 'ZZ' can be used for the following CPU modules only. For the usable programming tool, refer to the operating manual of the programming tool.

- QnU(D)(H)CPU with a serial number whose first five digits are '10042' or higher (excluding Q00UJCPU)
- Built-in Ethernet port QCPU
- LCPU

## Precautions

### ■Using the index setting for arguments of instruction/application function/function/function block

When "Use ZZ" is checked in "Indexing Setting for ZR Device" setting in the <<Device>> tab of the PLC parameter, and Z device is used for the argument of instruction/application function/function/function block, the expression is converted to "ZZ" at the compilation.

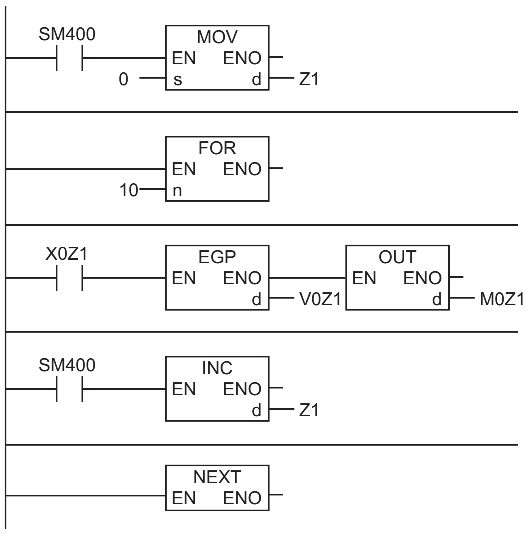
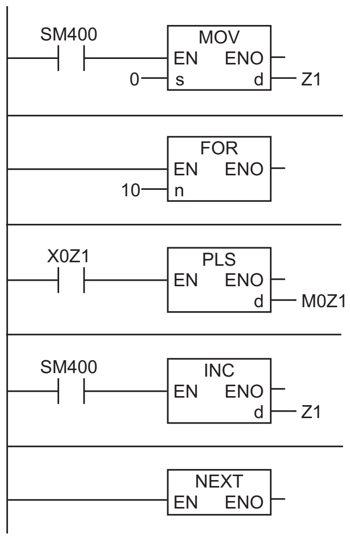
This may cause unintended device accesses.

When "Use ZZ" is checked, use ZZ devices for arguments of instruction/application function/function/function block.

### ■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 | When not using an edge relay |
|---|---|
| M0Z1 pulse is output normally. | M0Z1 pulse is not output normally. |
|  |  |





**Point**

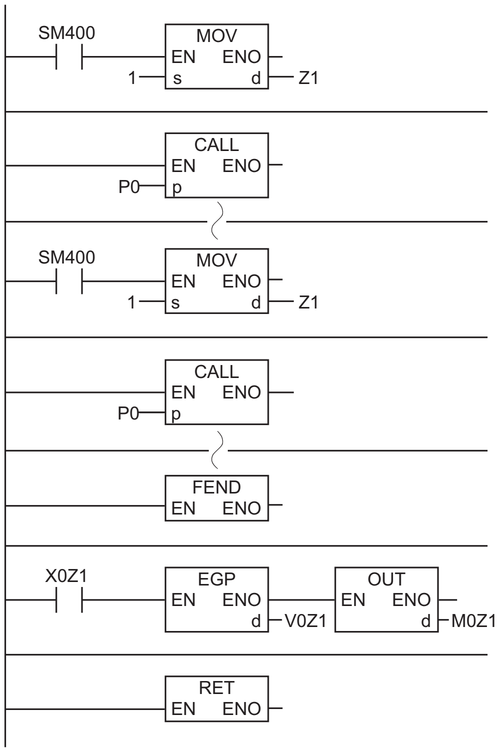
- The ON/OFF information of X0Z1 is stored to the edge relay V0Z1. For example, the ON/OFF data of X0 is stored to V0 and the ON/OFF data of X1 is stored to V1.
- Z0 and Z1 cannot be used when labels are used for array indexes within the FOR to NEXT instruction loop.

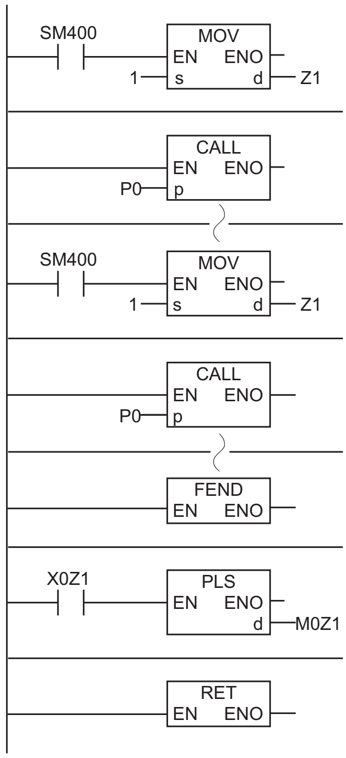
## ■Applying the index setting in the CALL instruction

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. |




## ■Device range check when the index setting is applied

- For Basic model QCPU, High Performance model QCPU, Process CPU, Redundant CPU, and FXCPU

The device range is not checked when the index setting is applied.

For Basic model QCPU, High Performance model QCPU, Process CPU, and Redundant CPU, 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.

- For the QnU(D)(H)CPU, QnUDE(H)CPU, 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.

The timings for checking the device range during index modification are shown below:

| Instruction | Timings for checking |
|---|---|
| Contact Instructions | Always*1 |
| Association instruction | |
| Comparison operation instruction (LD□) | |
| Comparison operation instruction (AND□) | When previous conditions are ON*1 |
| Comparison operation instruction (OR□) | When previous conditions are OFF*1 |
| Instructions other than the above | It follows the execution conditions for the instruction.*1*2*3 |

*1 When the data after index modification exceed the user specified device range, it may cause an error. (Error code: 4101).

*2 For the executions conditions for each instruction, refer to the descriptions page for each instruction.

*3 The PLS instruction and PLF instruction are excluded. (The PLS instruction and PLF instruction always check the device range during index modification.)

- For the QnUDVCPU and QnUDPVCPU:

The device range is checked during index modification.

It is also possible not to allow checking the device range using the parameters.

The timings for checking the device change during index modification are shown below.

| Instruction | Timings for checking |
|---|---|
| Contact Instructions | Always*4 |
| Association instruction | |
| Comparison operation instruction (LD□) | |
| Comparison operation instruction (AND□) | |
| Comparison operation instruction (OR□) | |
| Instructions other than the above | It follows the execution conditions for the instruction.*5*6*7 |

*4 When the data after index modification exceed the user specified device range, the operation results in OFF without causing an error.

*5 When the data after index modification exceed the user specified device range, it may cause an error. (Error code: 4101).

*6 For the executions conditions for each instruction, refer to the descriptions page for each instruction.

*7 The PLS instruction and PLF instruction are excluded. (The PLS instruction and PLF instruction always check the device range during index modification.)

### ■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.

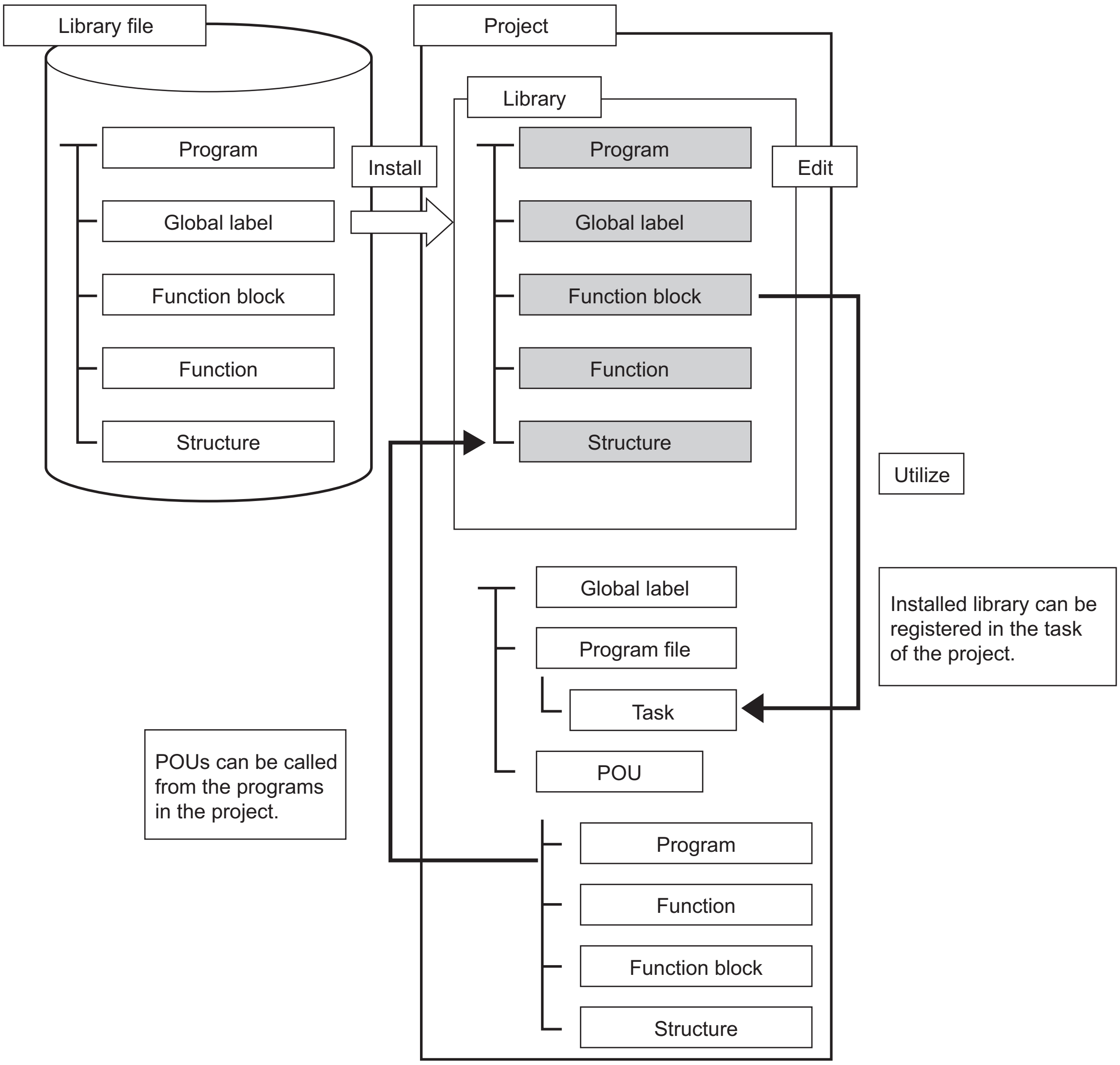
# 4.7 Libraries

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 following 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.



## 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.

### Composition of a user library

The following table shows data that can be registered in 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 programs, functions, and function blocks that can be used as libraries. |

# 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.

- Specify a name within 32 characters.
- Do not use reserved words. For reserved words, refer to the following section.

☞ Page 100 Character Strings That Cannot Be Used in Label Names and Data Names

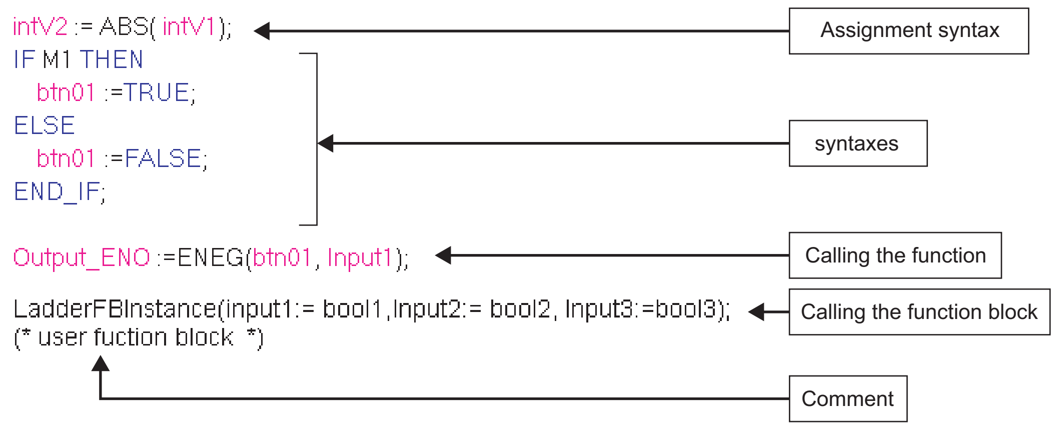
- Use alphanumeric and underscores (_).
- Do not use an underscore at the end of the name. Do not use two or more underscores in succession.
- Do not use spaces.
- Do not use a number for the initial character.
- Constants cannot be used. (An identifier that begins with 'H' or 'h' and an expression where 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'))
- Elementary data type names cannot be used.
- Function/FB names cannot be used.

# 5 WRITING PROGRAMS

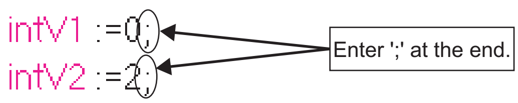
## 5.1 ST

The ST language is a text language with a similar grammatical structure to the C language. Controls such as conditional judgment and repetition process written in syntax can be described.
This language is suitable for programming complicated processes that cannot be easily described by a graphic language (structured ladder/FBD language).

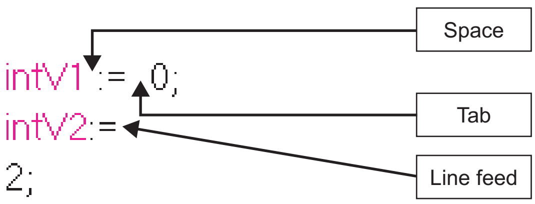
### Standard format



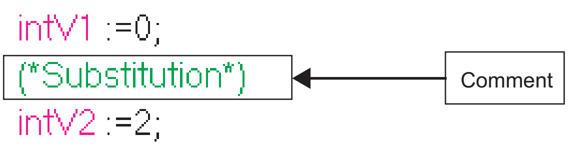

Operators and syntax are used for programming in the ST language.
Syntax must end with ';'.




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




Comments can be inserted in a program. Describe '(*' in front of a comment and '*)' in back of a comment.




Entering a comment in a comment causes the following compile error.
Compile error content: "Parser error" Error code : C1200

```
(* Flag_A = TRUE Control start *) Flag_A = FALSE Stop control *)

(* START (* Stop processing *) Restart End *)
```

# Operators in ST language

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

| Operator | Description | Example | Priority |
|---|---|---|---|
| ( ) | Parenthesized expression | (1+2)*(3+4) | 1 |
| Function ( ) | Function (Parameter list) | ADD_E(bo01, in01, in02, in03) | 2 |
| ** | Exponentiation | re01:= 2.0 ** 4.4 | 3 |
| NOT | Logical negation | NOT bo01 | 4 |
| *<br>/<br>MOD | Multiplication<br>Division<br>Modulus operation | 3 * 4<br>12 / 3<br>13 MOD 3 | 5 |
| +<br>- | Addition<br>Subtraction | in01 + in02<br>in01 - in02 | 6 |
| <, >, <=, >= | Comparison | in01 < in02 | 7 |
| =<br><> | Equality<br>Inequality | in01 = in02<br>in01 <> in02 | 8 |
| AND, & | Logical AND | bo01 & bo02 | 9 |
| XOR | Exclusive OR | bo01 XOR bo02 | 10 |
| OR | Logical OR | bo01 OR bo02 | 11 |

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.

| 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)<br>ANY_NUM (Exponent) | ANY_REAL |

# Syntax in ST language

The following table shows the syntax that can be used in the ST program.

| Type of syntax | Description | | Assignment syntax |
|---|---|---|---|
| Assignment syntax | Assignment syntax | | Page 75 Assignment syntax |
| Conditional syntax | IF conditional syntax | IF THEN conditional syntax | Page 76 IF THEN conditional syntax |
| | | IF ELSE conditional syntax | Page 76 IF ...ELSE conditional syntax |
| | | IF ELSIF conditional syntax | Page 77 IF ...ELSIF conditional syntax |
| | CASE conditional syntax | | Page 77 CASE conditional syntax |
| Iteration syntax | FOR DO syntax | | Page 78 FOR...DO syntax |
| | WHILE DO syntax | | Page 78 WHILE...DO syntax |
| | REPEAT UNTIL syntax | | Page 79 REPEAT...UNTIL syntax |
| Other control syntax | RETURN syntax | | Page 79 RETURN syntax |
| | EXIT syntax | | Page 80 EXIT syntax |



## Assignment syntax

### ■Format

<Left side> := <Right side>;

### ■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.

### ■Example

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

**Point**

Array type labels and structure labels can be used for the assignment syntax.
Note the data types of left side and right side.

- Array type labels

  The data type and the number of elements need to be the same for left side and right side.
  When using array type labels, do not specify elements.

  < Example >

  intAry1 := intAry2;

- Structure labels

  The data type (structured data type) needs to be the same for left side and right side.

  < Example >

  dutVar1 := dutVar2;

## IF THEN conditional syntax

### ■Format

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

### ■Description

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

### ■Example

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

## IF ...ELSE conditional syntax

### ■Format

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

### ■Description

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

Syntax 2 is executed when the value of Boolean expression is FALSE.

### ■Example

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

## IF ...ELSIF conditional syntax

### ■Format

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

### ■Description

Syntax 1 is executed when the value of Boolean expression (conditional expression) 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.



### ■Example

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

## CASE conditional syntax

### ■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;
```

### ■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.

### ■Example

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

## FOR...DO syntax

### ■Format

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

### ■Description

First, initialize the data to be used as an iteration variable.

One or more statements between the DO statement and the END_FOR statement are executed repeatedly, adding or subtracting the initialized iteration variable according to the increase expression until the final value is exceeded.

The iteration variable after the FOR...DO statement is completed retains the value at the end of the processing.

### ■Example

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

## WHILE...DO syntax

### ■Format

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

### ■Description

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

The Boolean expression is evaluated before the execution of the syntax. If the value of Boolean expression is FALSE, the syntax within DO...END_WHILE 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.

### ■Example

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

## REPEAT...UNTIL syntax

### ■Format

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

### ■Description

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

The Boolean expression is evaluated after the execution of the syntax. If the value of Boolean expression is TRUE, the syntax 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.

### ■Example

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



## RETURN syntax

### ■Format

```
RETURN;
```

### ■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.

### ■Example

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

## EXIT syntax

### ■Format

```
EXIT;
```

### ■Description

The EXIT syntax is used only in iteration syntax 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.

### ■Example

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

# Calling functions in ST language

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

| Description of calling functions |
|---|
| 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.

| Function | Example |
|---|---|
| Calling a function with one input variable (Example: ABS) | Output1 := ABS(Input1); |
| Calling a function with three input variables (Example: MAX) | Output1 := MAX(Input1, Input2, Input3); |
| Calling a function with EN/ENO (Example: MOV) | boolENO := MOV(boolEN, Input1, Output1);*1 |

*1 For a function with EN/ENO, the result of the function execution is ENO, and the first argument (Variable 1) is EN.



# Calling function blocks in ST language

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

| Description of calling function blocks in ST language. |
|---|
| Instance name (Input variable1:= Variable1, ... Output variable1: = Variable2, ...); |

Enclose the assignment syntax that assigns variables to the input variable and output variable by '( )' after the instance name.
When using multiple variables, delimit assignment syntax 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.

| Function block | FB definition | | Example |
|---|---|---|---|
| Calling a function block with one input variable and one output variable | FB Name | FBADD | FBADD1(IN1:=Input1);<br>Output1 := FBADD1.OUT1; |
| | FB instance name | FBADD1 | |
| | Input variable1 | IN1 | |
| | Output variable1 | OUT1 | |
| Calling a function block with three input variables and two output variables | FB Name | FBADD | FBADD1(IN1:=Input1, IN2:=Input2, IN3:=Input3);<br>Output1 := FBADD1.OUT1;<br>Output2 := FBADD1.OUT2; |
| | FB instance name | FBADD1 | |
| | Input variable1 | IN1 | |
| | Input variable2 | IN2 | |
| | Input variable3 | IN3 | |
| | Output variable1 | OUT1 | |
| | Output variable2 | OUT2 | |

**Point**

- Arguments using at function block call

VAR_OUTPUT is not appeared on a template if a checkbox in the following option window is not selected; [Tools] → [Options] → "Convert" → "Structured Ladder/FBD/ST" "Compile Condition1" → "Allow VAR_OUTPUT at FB call (ST)".

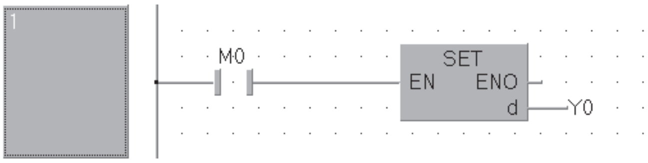
# Precautions when using conditional syntax and iteration syntax

The following explains the precautions when creating ST programs using conditional syntax and iteration syntax.

## Precaution 1

Once the conditions (boolean expression) are met in the conditional syntax or iteration syntax, the bit device which is turned ON in the <syntax> is always set to ON.

### ■A program whose bit device is always set to ON

| ST program | Structured ladder/FBD program equivalent to ST program |
|---|---|
| IF M0 THEN<br>  Y0 := TRUE;<br>END_IF; |  |

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

### ■A program to avoid the bit device to be always set to ON.

| ST program*1 | Structured ladder/FBD program equivalent to ST program |
|---|---|
| IF M0 THEN<br>  Y0 := TRUE;<br>  ELSE<br>  Y0 := FALSE;<br>END_IF; |  |

*1 The above program can also be written as follows.
Y0 := M0;
or
OUT(M0,Y0);
Note that, when the OUT instruction is used in <syntax> of conditional syntax or iteration syntax, the program status becomes the same as the program whose bit device is always set to ON.

## Precaution 2

When Q00UCPU, Q00UJCPU or, Q01UCPU is used, and the string type is applied to Boolean expression (conditional expression) with conditional syntax or iteration syntax, a compilation error may occur.

### ■Program example which causes compilation error

**ST program**

```
IF Var_String = "MOJI" THEN
  Y0 := TRUE;
END_IF;
```

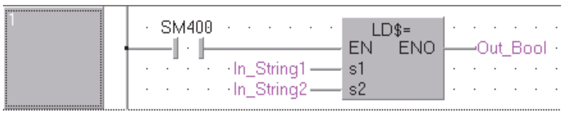
Compilation error occurs
when specifying string type data.

To avoid a compilation error, create the function blocks of the string type comparison with ladder or structured ladder/FBD, and apply the operation result of function blocks to the conditional expression of conditional syntax or iteration syntax.
The following is an example when creating the function blocks with structured ladder/FBD.

### ■Program creation example which avoids compilation error

**1.** Create the function blocks of the string type comparison with structured ladder/FBD program

**Function block (EQFB_01)**




**2.** Apply the operation result of function blocks (EQFB_01) to the conditional expression in ST program.

**Label setting**

| | Class | Label Name | Data Type |
|---|---|---|---|
| 1 | VAR | Var_String | String(32) |
| 2 | VAR | Var_Bool | Bit |
| 3 | VAR | Inst_EQFB | EQFB_01 |

**ST program**

```
Inst_EQFB( In_String1 := Var_String,
           In_String2 := "MOJI",
           Out_Bool := Var_Bool);
IF TRUE = Var_Bool THEN
  Y0 := TRUE;
  END_IF;
```

Apply the operation result of
function blocks (EQFB_01)

## Precaution 3

The following table lists operations when the STMR instruction or instructions that are executed at the rising or falling edge are used in the IF or CASE conditional statement.

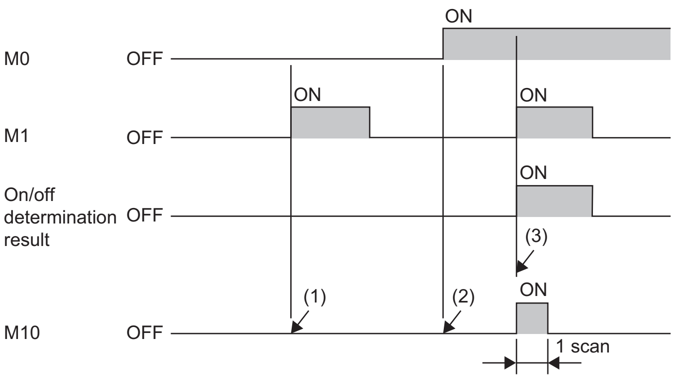
| **Condition** | | | **Operation result** | | | |
|---|---|---|---|---|---|---|
| **Conditional formula of IF or CASE conditional statement** | **Instruction execution condition (EN)** | **On/off determination result of the instruction in the last scan** | **On/off determination result of the instruction** | **Rising edge execution instruction** | **Falling edge execution instruction** | **STMR instruction** |
| TRUE or CASE match | TRUE | ON | ON | Not executed | Not executed | Previous value held |
| | | OFF | ON | Executed | Not executed | Rising edge processing |
| | FALSE | ON | OFF | Not executed | Executed | Falling edge processing |
| | | OFF | OFF | Not executed | Not executed | Previous value held |
| FALSE or CASE mismatch | TRUE | ON | ON*1 | Not executed | Not executed | Previous value held |
| | | OFF | OFF | Not executed | Not executed | Previous value held |
| | FALSE | ON | ON*1 | Not executed | Not executed | Previous value held |
| | | OFF | OFF | Not executed | Not executed | Previous value held |

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

**Ex.**

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

```
IF M0 THEN
    PLS(M1, M10);
END_IF;
```




(1) When M0 is off (the conditional formula of the IF conditional statement is FALSE), the on/off determination result will be off. The PLS instruction is not executed. (The M10 remains off.)
(2) When M0 is on (the conditional formula of the IF conditional statement is TRUE) and M1 is off (the instruction execution condition is off), the on/off determination result will be off. The PLS instruction is not executed. (The M10 remains off.)
(3) When M0 is on (the conditional formula of the IF conditional statement is TRUE) and M1 is also on (the instruction execution condition is on), the on/off determination result will be off to on (rising edge). The PLS instruction is executed. (The M10 is on for one scan.)

- To execute the rising or falling edge execution instruction in the iteration statement, use the edge relay (V) or perform index modification. When the rising or falling edge execution instruction in the iteration statement is used, the instruction may not be executed normally at rising or falling edge.

**Ex.**

When the rising execution instructions is used in the FOR statement

- Example that the edge relay (V) is used*2

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

- Example that the edge relay (V) is not used

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

*2 The edge relay (V) is used 1 bit in the system in addition to the number of bits used in the loop. The edge relay (V) is used up to a total of 11 points (V0 to V10) in the above example.

# Operations when the master control instruction is used

Operations between the MC and MCR instructions when the master control is off will be as follows.

- Off is assigned to the assignment statement (bit).
- The assignment statement (word) performs no processing.
- When the statement is other than assignment statement, the execution processing is not performed.

**Ex.**

For the assignment statement (bit)

```
MC(M0, N1, M1);
M3 := M2;
M20 := MCR(M0, N1);
```

M3 is off because off is assigned when the master control is off.

**Ex.**

For the assignment statement (word)

```
MC(M0, N1, M1);
D3 := D2;
M20 := MCR(M0, N1);
```

D3 retains the previous value because no processing is performed when the master control is off.

**Ex.**

For the statement (OUT instruction) that is other than assignment statement

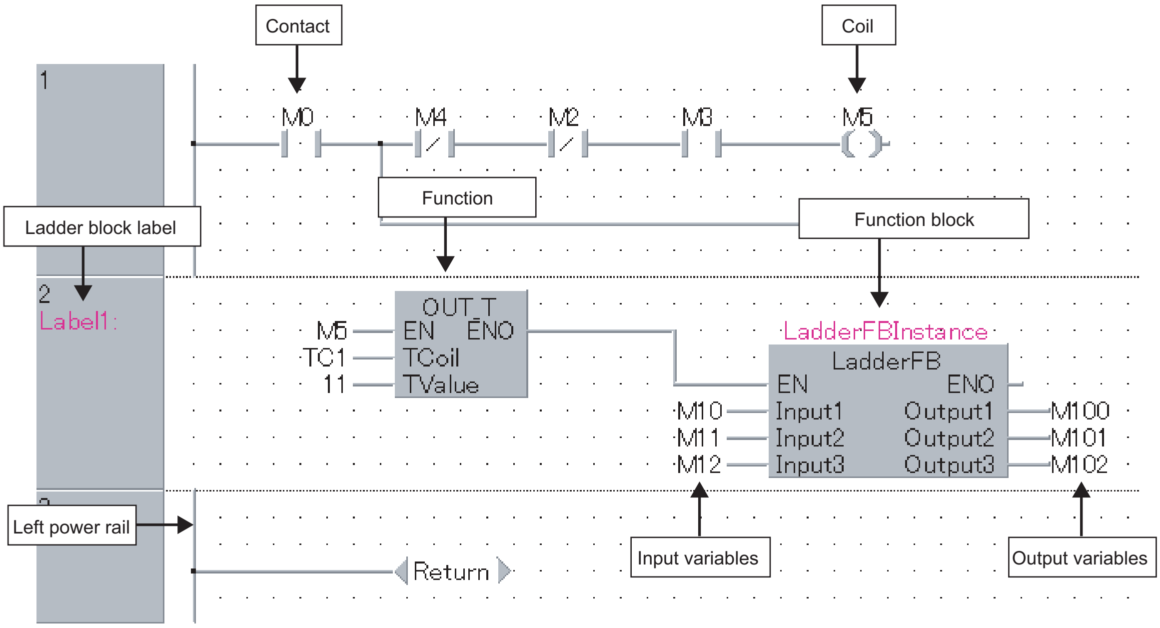
```
MC(M0, N1, M1);
OUT(M2, M3);
M20 := MCR(M0, N1);
```

M3 is off because the instruction is not executed when the master control is off.

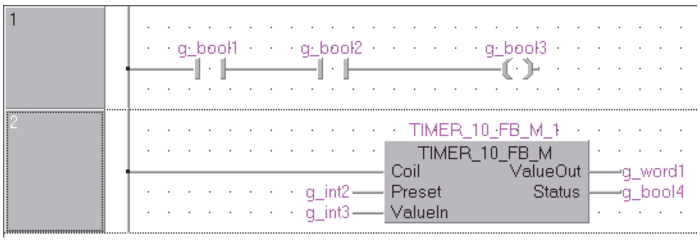
# 5.2 Structured Ladder/FBD

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

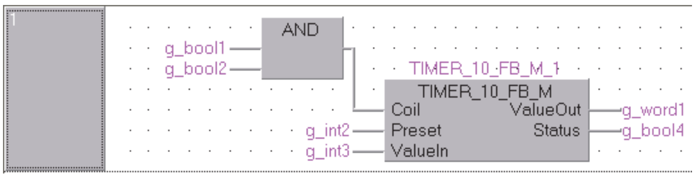
## Standard format




In the structured ladder/FBD language, units of ladder blocks are used for programming.
For structured ladder, connect the left power rail and ladder symbols with lines.



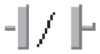
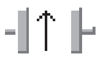
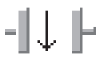
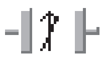
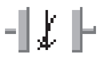
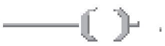
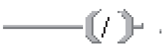
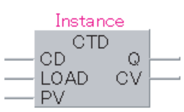
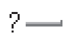

For FBD, connect the ladder symbols with lines according to the flow of data or signals without connecting with the left power rail.



## Ladder symbols in structured ladder/FBD language

The following table shows the ladder symbols that can be used in the structured ladder/FBD language.
For details, refer to the following manual.

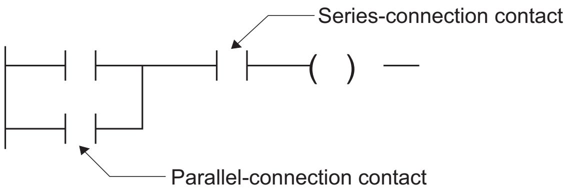
MELSEC-Q/L Structured Programming Manual (Common Instructions)

| Element | Ladder symbol | Description |
|---|---|---|
| Normal*1*2 |  | Turns ON when a specified device or label is ON |
| Negation*1*2 |  | Turns OFF when a specified device or label is OFF. |
| Rising edge*1*2*3 |  | Turns ON at the rising edge (OFF to ON) of a specified device or label. |
| Falling edge*1*2*3 |  | Turns ON at the falling edge (ON to OFF) of a specified device or label. |
| Negated rising edge*1*2*3 |  | 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*3 |  | 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*1 |  | Outputs the operation result to a specified device or label. |
| Negation*1 |  | A specified device or label turns ON when the operation result turns OFF. |
| Set*1 |  | A specified device or label turns ON when the operation result turns ON.<br>Once the device or label turns ON, it remains ON even when the operation result turns OFF. |
| Reset*1 |  | 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. |
| Jump |  | Pointer branch instruction<br>Unconditionally executes the program at the specified pointer number in the same POUs. |
| Return |  | Indicates the end of a subroutine program. |
| Function |  | Executes a function. |
| Function block |  | Executes a function block. |
| Function argument input |  | Inputs an argument to a function or function block. |

| Element | Ladder symbol | Description |
|---|---|---|
| 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 |

*1 Not applicable in FBD.

*2 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.




*3 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 Version 1 Operating Manual (Common)

**Point**

The performance of return differs depending on the programs, functions, and function blocks being used.

- When used in the programs

End the execution of POUs

- When used in the functions

End the functions. Also, return to the next step of the instruction which called the functions.

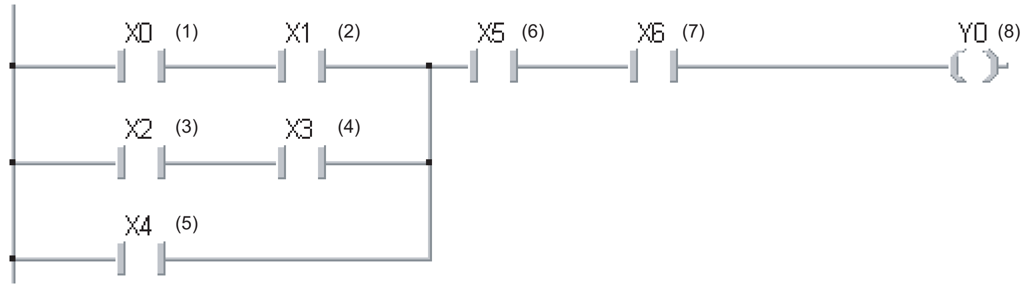
- When used in the function blocks

The performance differs depending on whether "Use Macrocode" is checked or not on the Property screen.
When it is checked, end the execution of POUs.
When it is not checked, end the function blocks. Also, return to the next step of the instruction which called the functions.

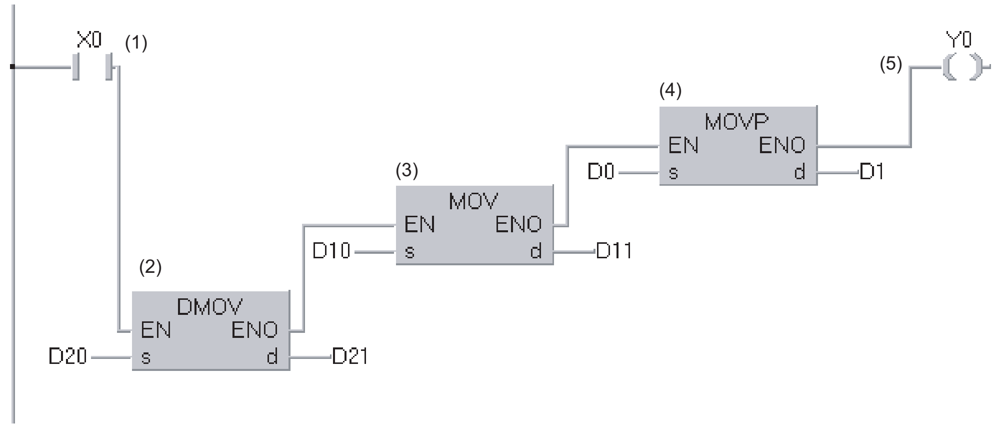
## Executing order

The following figures explain the program executing order.

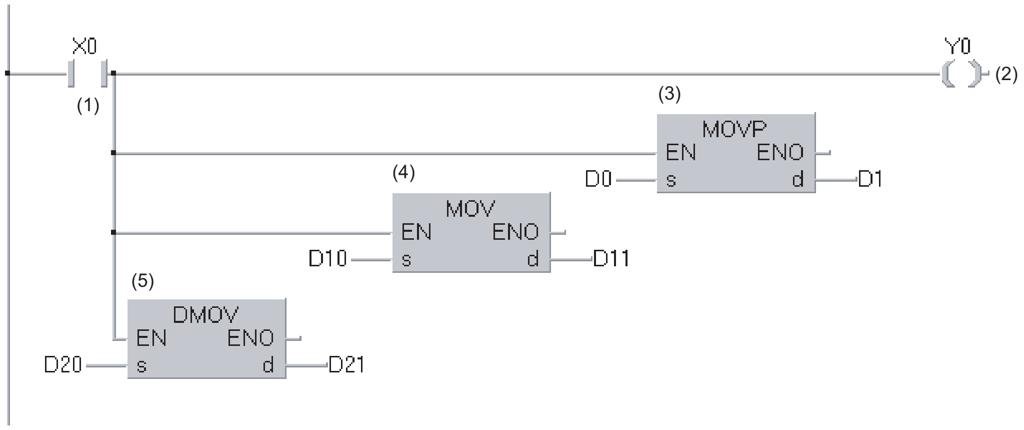
The operation order in a ladder block is from the left power rail to the right and from the top to the bottom.



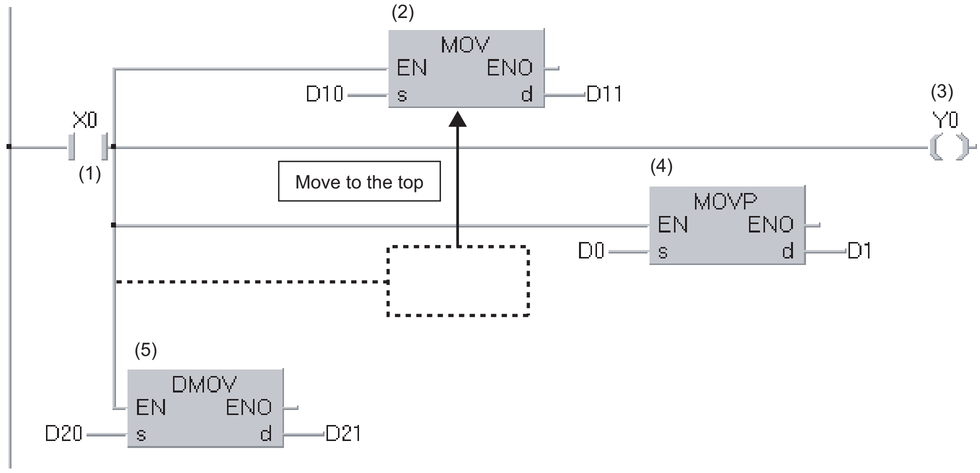

The program is executed from the left power rail to the right when the ladder is not branched and ENs and ENOs are connected in series.



The program is executed from the top to the bottom, when the ladder is branched and ENs and ENOs are connected in parallel.




The program is executed in the order as shown below when the MOV instruction (4) in the above figure is moved to the top.



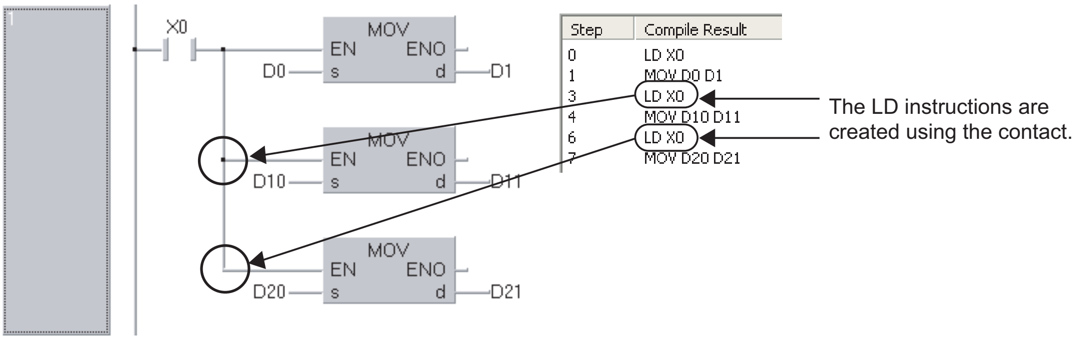
# Ladder branches and compilation results

When the ladder is branched, different compilation results are produced for the program after the branch depending on the program up to the branch. The following explains the precautions on compilation results depending on ladder branches.

## One contact is used up to the branch

The instruction of the contact is used multiple times in the compilation result.

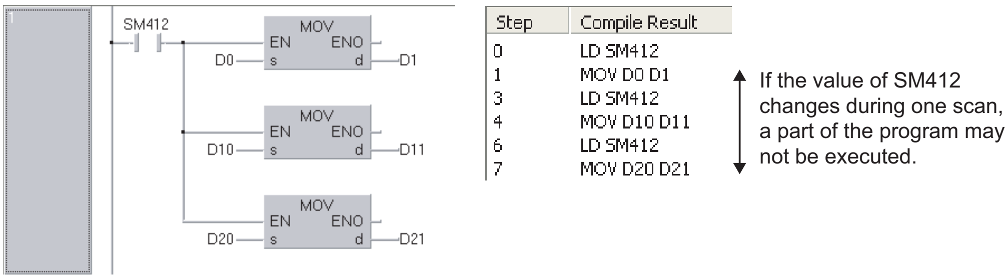
< Example > < Compilation result >




### ■Precautions

When the device in which the value changes during one scan (such as SM412) is used, only a part of the sequence program after the branch is executed, and the rest of the sequence program may not be executed.

< Example > < Compilation result >




When executing multiple instructions against one contact, connect the instructions in series. Since the sequence program uses the LD instruction only once in the compilation result, all sequence programs are executed.

< Example >

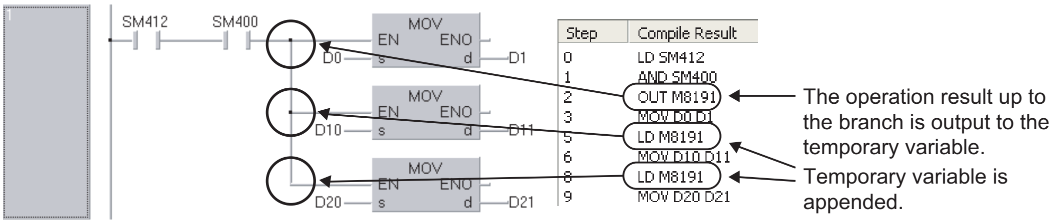

< Compile Result >

| Step | Compile Result |
|---|---|
| 0 | LD SM412 |
| 1 | MOV D0 D1 |
| 3 | MOV D10 D11 |
| 5 | MOV D20 D21 |

## Multiple contacts are used up to the branch

The temporary variable is appended to the branch in the compilation result.

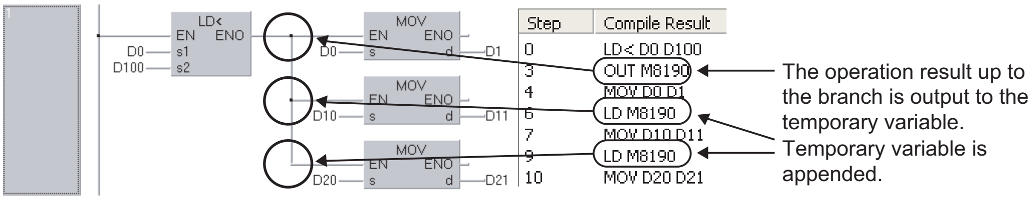
< Example > < Compilation result >




## Output value of function or function block is branched

The temporary variable is appended to the branch in the compilation result.

< Example > < Compilation result >




Connect the instructions in series to avoid using temporary variables in the compilation result.

☞ Page 91 Precautions

For details on temporary variables, refer to the following manual.

📖 GX Works2 Version 1 Operating Manual (Structured Project)

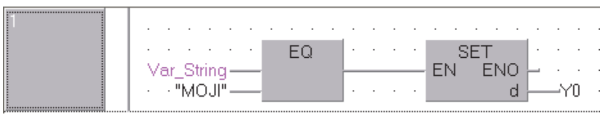
# Precautions on creating programs with structured ladder/FBD

The following explains the Precautions on creating a program with structured ladder/FBD.

When Q00UCPU, Q00UJCPU, Q01UCPU is used, and the string type is applied to enter the standard comparison functions, a compilation error may occur.

**Ex.**

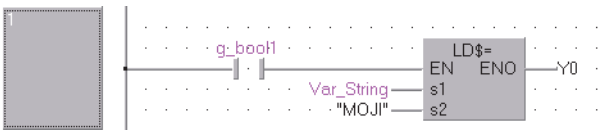
Program example which causes compilation error




To avoid a compilation error, use LD$=, LD$<>, LD$<=, LD$<, LD$>=, or LD$> instructions.

**Ex.**

Program example which avoids compilation error



# APPENDICES

# Appendix 1 Correspondence Between Generic Data Types and Devices

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

## Internal user device

### Bit device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Input | X | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Output | Y | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Internal relay | M | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Latch relay | L | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Annunciator | F | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Edge relay | V | ○ | × | × | × | × | × | × | × | × |
| Step relay | S | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Link special relay | SB | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Link relay | B | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Timer contact*2 | TS | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Timer coil*2 | TC | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Retentive timer contact*2 | STS | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Retentive timer coil*2 | STC | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Counter contact*2 | CS | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |
| Counter coil | CC | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for digit specification.

## Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY[*1] | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Timer current value | T or TN[*2] | × | ○ | × | ○ | × | × | × | × | × |
| Retentive timer current value | ST or STN[*2] | × | ○ | × | ○ | × | × | × | × | × |
| Counter current value | C or CN[*2] | × | ○ | × | ○ | × | × | × | × | × |
| Data register | D | ○[*3] | ○ | × | ○ | × | × | × | × | × |
| Link register | W | ○[*3] | ○ | × | ○ | × | × | × | × | × |
| Link special register | SW | ○[*3] | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for digit specification.

*3 Can be used for bit specification

## Internal system device

### Bit device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Function input | FX | — | — | — | — | — | — | — | — | — |
| Function output | FY | — | — | — | — | — | — | — | — | — |
| Special relay | SM | ○ | ○*2 | ○*2 | ○*2 | ○*2 | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for digit specification.

### Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Function register | FD | — | — | — | — | — | × | × | × | × |
| Special register | SD | ○*2 | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for bit specification

# Link direct device

## Bit device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY[*1] | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Link input | Jn\X | ○ | ○[*2] | ○[*2] | ○[*2] | ○[*2] | × | × | × | × |
| Link output | Jn\Y | ○ | ○[*2] | ○[*2] | ○[*2] | ○[*2] | × | × | × | × |
| Link relay | Jn\B | ○ | ○[*2] | ○[*2] | ○[*2] | ○[*2] | × | × | × | × |
| Link special relay | Jn\SB | ○ | ○[*2] | ○[*2] | ○[*2] | ○[*2] | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for digit specification.

## Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY[*1] | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Link register | Jn\W | ○[*2] | ○ | × | ○ | × | × | × | × | × |
| Link special register | Jn\SW | ○[*2] | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for bit specification

# Intelligent function module device

## Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Intelligent function module device | Un\G | ○*2 | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for bit specification

# Index register

## Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Index register | Z | × | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

# File register

## Word device

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| File register | R or ZR | ○*2 | ○ | × | ○ | × | × | × | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

*2 Can be used for bit specification

# Nesting

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Nesting | N | — | — | — | — | — | — | — | — | — |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

# Pointer

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY*1 | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| Pointer | P | — | — | — | — | — | — | — | — | — |
| Interrupt pointer | I | — | — | — | — | — | — | — | — | — |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

## Constant

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY[*1] | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| — | K, H | ○ | ○ | ○ | ○ | ○ | ○ | × | × | × |
| | E | × | × | × | × | × | ○ | ○ | × | × |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

## String constant

| Device | | Generic data type | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Device name | Device symbol | ANY[*1] | | | | | | | | |
| | | ANY_SIMPLE | | | | | | | | |
| | | ANY_BIT | | | ANY_NUM | | | | Time | String |
| | | | | | 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 | | |
| — | 'Character string' or "Character string" | × | × | × | × | × | × | × | × | ○ |

*1 Devices available for ANY16 are the same as the ones for word (unsigned)/16-bit string and word (signed).
Devices available for ANY32 are the same as the ones for double word (unsigned)/32-bit string and double word (signed).
No devices are available for arrays and structures.

# 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 tables shows character strings that cannot be used for label names or data names.

The numbers from (1) to (9) in the tables indicate the following label names and data names.

| <Label name and data name> | |
|---|---|
| (1) | Project file name |
| (2) | Program file name (Simple (without labels)) |
| (3) | Program file name (Simple (with labels)) |
| (4) | Program file name (structure) |
| (5) | Task name |
| (6) | Global label data name |
| (7) | Structure name |
| (8) | POU name |
| (9) | Label name |

○: Applicable, △: With restrictions, ×: Not applicable

| Category | Character string | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|---|---|
| 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, TIMER, COUNTER, RETENTIVETIMER, POINTER, Bit, Word [Unsigned]/Bit String [16-bit], Double Word [Unsigned]/Bit String [32-bit], Word [Signed], Double Word [Signed], FLOAT (Single Precision), FLOAT (Double Precision), String, Time, Timer, Counter, Retentive Timer, Pointer | ○ | ○ | × | × | × | × | × | × | × |
| Data type hierarchy | ANY, ANY_NUM, ANY_BIT, ANY_REAL, ANY_INT, ANY_DATE | ○ | ○ | × | × | × | × | × | × | × |
| | ANY_SIMPLE, ANY16, ANY32 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| 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*2 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| Character string recognized as device (Device name + Numeral) | Such as X0 | ○ | ○ | × | × | × | × | × | △*3 | × |
| ST operator | NOT, MOD | ○ | ○ | × | × | × | × | × | × | × |
| | (, ), - | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| 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 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |

| Category | Character string | (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) |
|---|---|---|---|---|---|---|---|---|---|---|
| Application instruction in GX Works2 | Application instructions such as DMOD, PCHK, INC(P)<br>For details, refer to the following.<br>• QCPU (Q mode)/LCPU<br>📖 MELSEC-Q/L Programming Manual (Common Instruction)<br>📖 MELSEC-Q/L Structured Programming Manual (Common Instructions)<br>• FXCPU<br>📖 FXCPU Structured Programming Manual [Basic & Applied Instruction]<br>📖 FXCPU Structured Programming Manual [Application Functions] | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| 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 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| 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, FUNCTION_BLOCK, END_FUNCTION_BLOCK, STEP, INITIAL_STEP, END_STEP, TRANSITION, END_TRANSITION, FROM, TO, UNTILWHILE | ○ | ○ | × | × | × | × | × | × | × |
| Function name in application function | Function names in application functions such as AND_E, NOT_E | ○ | ○ | ○ | ○ | ○ | ○ | × | × | × |
| Function block name in application function | Function block names in application functions such as CTD, CTU | ○ | ○ | ○ | ○ | ○ | ○ | × | × | × |
| Symbol | /, \, *, ?, <, >, \|, ", :, [, ] , , , =, +, %, ', ~, @, {, }, &, ^, ., ', tab character | × | × | × | × | × | × | × | × | × |
| | ; | ○ | × | × | × | × | × | × | × | × |
| | !, #, $, ` | ○ | ○ | ○ | ○ | × | ○ | × | × | × |
| 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, RETAIN, INTERVAL | × | ○ | × | × | × | × | × | × | × |
| String that starts with K1 to K8 | Such as K1AAA | ○ | ○ | ○ | ○ | ○ | ○ | ○ | △*1 | × |
| 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 | ○ | ○ | × | ○ | ○ | × | × | △*3 | × |
| 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 Functions cannot be used.

*2 Whether to handle a device name indexed with ZZ device as a reserved word depends on the parameter setting.
When Z device is specified for 32-bit index setting: Not handled as a reserved word
When ZZ device is specified for 32-bit index setting: Handled as a reserved word

*3 Applicable for Simple projects without labels only.

A

## Precautions on using labels

- In a function, the same name as the function cannot be used for a label.
- A space cannot be used.
- A numeral cannot be used at the beginning of label name.
- A label name is not case-sensitive. An error may occur at compilation when the same label names with different cases (example: 'AAA' and 'aaa') are declared.
- An underscore (_) cannot be used at the beginning or end of label name. Consecutive underscores (_) cannot be used for data name and label name.
- For Simple projects, function names and function block names in common instructions and application functions can be used.
- In structured ladder/FBD and ST programs, the same label name can be used for a global label and a local label by setting the following option in GX Works2.

Check the "Use the same label name in global label and local label" item under [Tool] ⇨ [Options] ⇨ "Compile" ⇨ "Basic Setting".

# 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.

## 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.

| Procedure | |
|---|---|
| **1.** Replacing devices with labels | Labels include global labels and local labels.<br>Determine the type of labels (global label or local label) to replace devices. |
| **2.** Setting labels | Global labels and local labels to be used in the program must be defined.<br>Define all labels to be used in the program. |
| **3.** Creating a program | Create a structured program in the programming language to be used. |

A

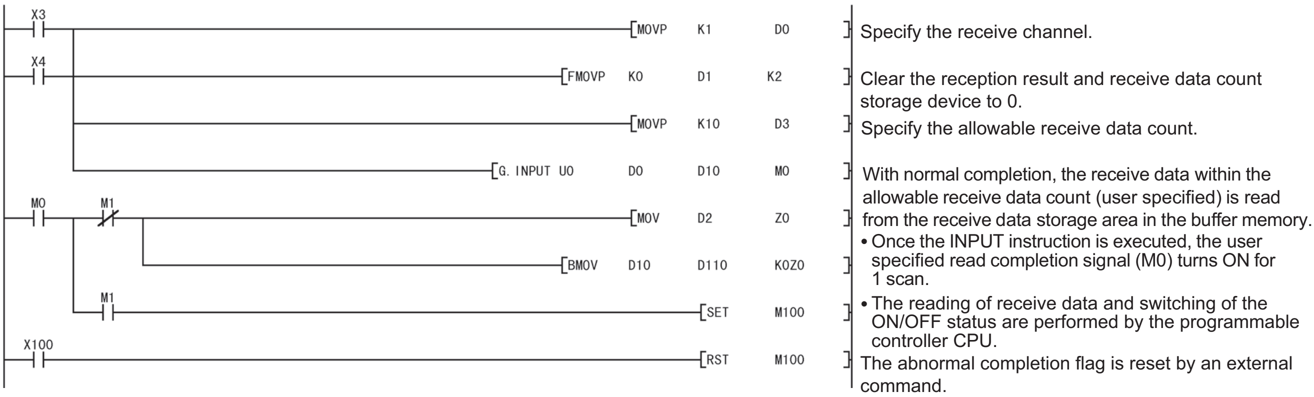
# Example of creating a structured program

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

**Ex.**

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/FBD and ST languages.

The following shows the original program.




## 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.

| 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] | ReceiveData |
| 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 |

## 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 | ... | |

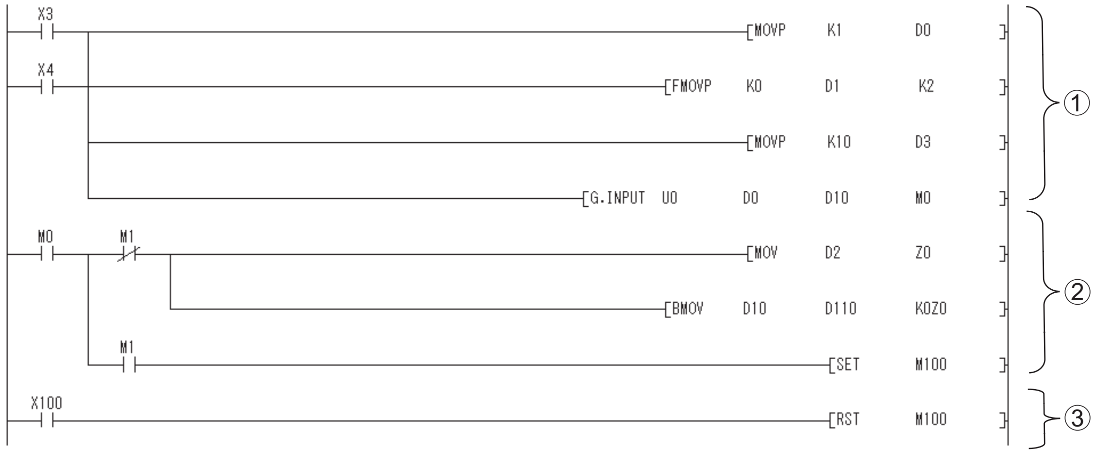
*1 Devices of local labels are automatically assigned within the range specified in the device/label automatic-assign setting in GX Works2. To assign the same devices as those in the original ladder program, set them as global labels.

A

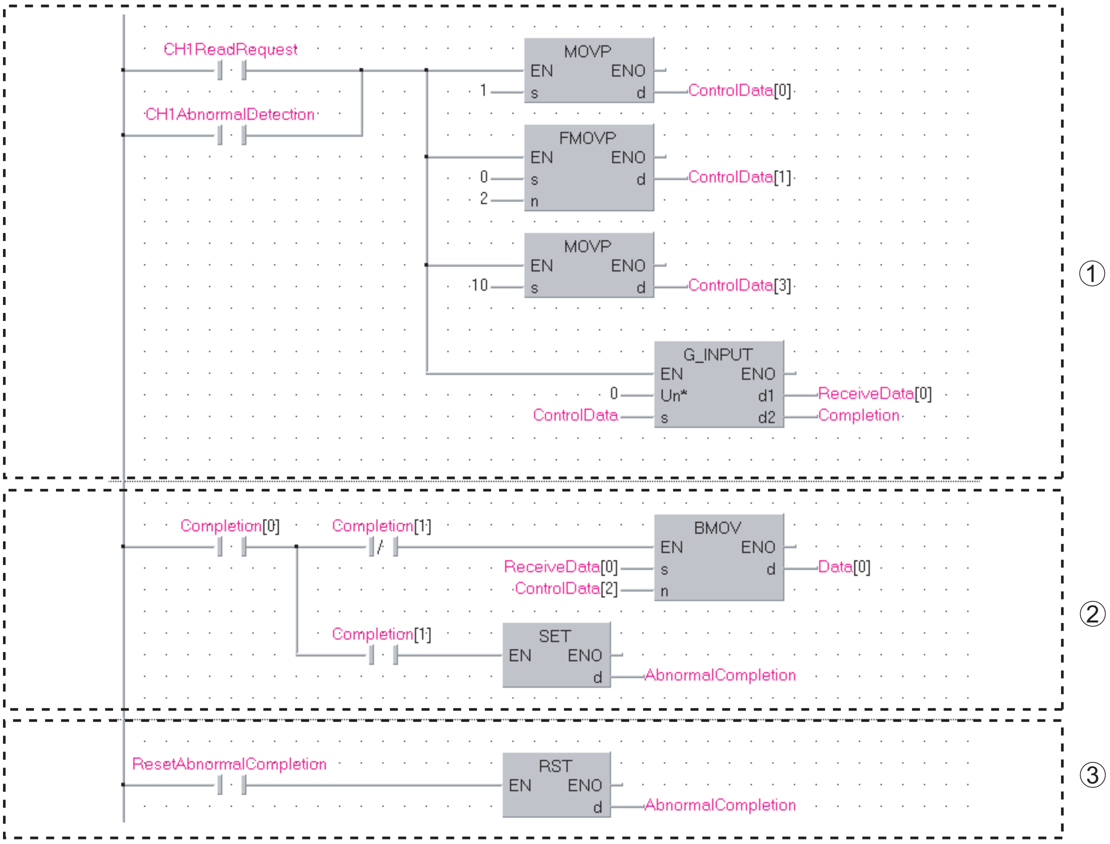
## Creating a structured program

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

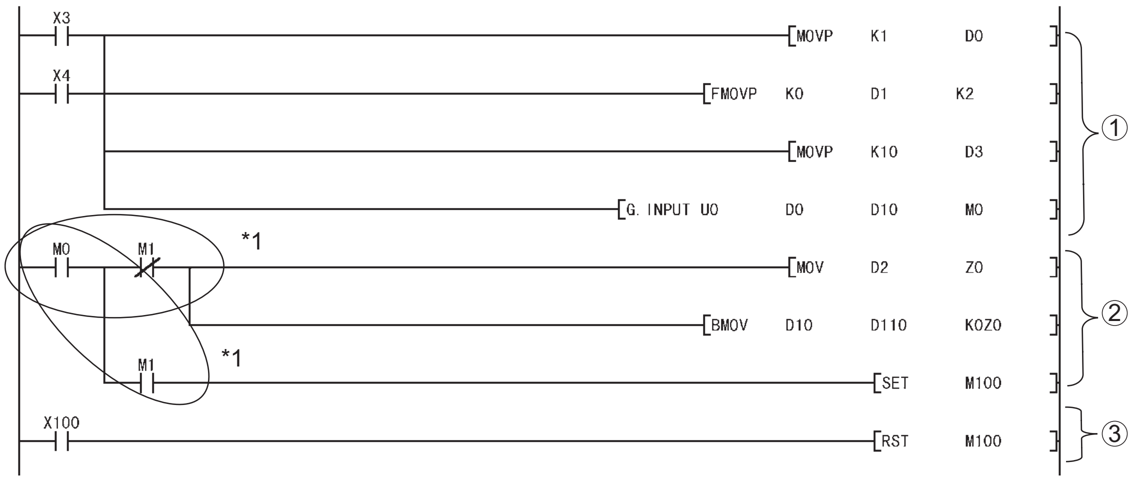
- Original program (Programming language: ladder)




- Structured program (Programming language: structured ladder/FBD)



- 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
SET(Completion[0] AND Completion[1], AbnormalCompletion);
```
②

```
RST(ResetAbnormalCompletion,  AbnormalCompletion);
```
③

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

A

# INDEX

## T



## U



## W



I

# REVISIONS

*The manual number is given on the bottom left of the back cover.

| Print date | *Manual number | Revision |
|---|---|---|
| July 2008<br>⋮<br>July 2013 | SH(NA)-080782ENG-A<br>⋮<br>SH(NA)-080782ENG-M | Due to the transition to the e-Manual, the details of revision have been deleted. |
| October 2015 | SH(NA)-080782ENG-N | Complete revision (layout change) |
| September 2018 | SH(NA)-080782ENG-O | ■Added or modified parts<br>INTRODUCTION, Section 4.4, 4.6 |

Japanese manual number: SH-080735-T

# WARRANTY

Please confirm the following product warranty details before using this product.

## 1. Gratis Warranty Term and Gratis Warranty Range

If any faults or defects (hereinafter "Failure") found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company.
However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing on-site that involves replacement of the failed module.

[Gratis Warranty Term]

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

[Gratis Warranty Range]

(1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.

(2) Even within the gratis warranty term, repairs shall be charged for in the following cases.

1. Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
2. Failure caused by unapproved modifications, etc., to the product by the user.
3. When the Mitsubishi product is assembled into a user's device, Failure that could have been avoided if functions or structures, judged as necessary in the legal safety measures the user's device is subject to or as necessary by industry standards, had been provided.
4. Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
5. Failure caused by external irresistible forces such as fires or abnormal voltages, and Failure caused by force majeure such as earthquakes, lightning, wind and water damage.
6. Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
7. Any other failure found not to be the responsibility of Mitsubishi or that admitted not to be so by the user.

## 2. Onerous repair term after discontinuation of production

(1) Mitsubishi shall accept onerous product repairs for seven (7) years after production of the product is discontinued. Discontinuation of production shall be notified with Mitsubishi Technical Bulletins, etc.

(2) Product supply (including repair parts) is not available after production is discontinued.

## 3. Overseas service

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

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

Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation to:

(1) Damages caused by any cause found not to be the responsibility of Mitsubishi.

(2) Loss in opportunity, lost profits incurred to the user by Failures of Mitsubishi products.

(3) Special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products.

(4) Replacement by the user, maintenance of on-site equipment, start-up test run and other tasks.

## 5. Changes in product specifications

The specifications given in the catalogs, manuals or technical documents are subject to change without prior notice.

# TRADEMARKS

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The company names, system names and product names mentioned in this manual are either registered trademarks or trademarks of their respective companies.

In some cases, trademark symbols such as '™' or '®' are not specified in this manual.



SH(NA)-080782ENG-O

SH(NA)-080782ENG-O(1809)KWIX
MODEL: Q/FX-KP-KI-E
MODEL CODE: 13JW06

**MITSUBISHI ELECTRIC CORPORATION**

HEAD OFFICE : TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS : 1-14 , YADA-MINAMI 5-CHOME , HIGASHI-KU, NAGOYA , JAPAN