

QCPU

**mitsubishi**

Structured Programming Manual

(Fundamentals)

A large graphic featuring the text 'Q series series'. The first 'Q' is a large, stylized, 3D-rendered letter. The word 'series' is written in a serif font, appearing twice: once in a light, semi-transparent style overlapping the first 'Q', and once in a dark, solid style below it. The background consists of overlapping gray rectangles, one of which has a textured pattern.

Mitsubishi  
Programmable Controller

**MELSEC-Q**



# ● SAFETY PRECAUTIONS ●

(Always read these instructions before using this product.)

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

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



# INTRODUCTION

---

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

## CONTENTS

---

|   |       |
|---|-------|
| SAFETY PRECAUTIONS .....                            | A - 1 |
| REVISIONS.....                                      | A - 2 |
| INTRODUCTION.....                                   | A - 3 |
| CONTENTS .....                                      | A - 3 |
| MANUALS.....  | A - 5 |
| PURPOSE OF THIS MANUAL .....                        | A - 6 |
| GENERIC TERMS AND ABBREVIATIONS IN THIS MANUAL..... | A - 8 |

|                                     |                       |
|-------------------------------------|-----------------------|
| <b>1. OVERVIEW</b>                  | <b>1 - 1 to 1 - 4</b> |
| 1.1 Overview                        | 1 - 2                 |
| 1.2 Features of Structured Programs | 1 - 2                 |
| 1.3 Applicable CPU Modules          | 1 - 3                 |
| 1.4 Compatible Software Package     | 1 - 3                 |

|  |                       |
|--|-----------------------|
| <b>2. STRUCTURED DESIGN OF SEQUENCE PROGRAMS</b> | <b>2 - 1 to 2 - 4</b> |
| 2.1 What is a Hierarchical Sequence Program?     | 2 - 2                 |
| 2.2 What is a Structured Sequence Program?       | 2 - 3                 |

|  |                       |
|--|-----------------------|
| <b>3. PROCEDURE FOR CREATING PROGRAMS</b>                          | <b>3 - 1 to 3 - 2</b> |
| 3.1 Procedure for Creating Sequence Programs in Structured Project | 3 - 2                 |

|  |                        |
|--|------------------------|
| <b>4. PROGRAM CONFIGURATION</b>            | <b>4 - 1 to 4 - 28</b> |
| 4.1 Overview of Program Configuration      | 4 - 2                  |
| 4.1.1 Project .....                        | 4 - 3                  |
| 4.1.2 Program files.....                   | 4 - 3                  |
| 4.1.3 Tasks .....                          | 4 - 4                  |
| 4.2 POUs                                   | 4 - 5                  |
| 4.2.1 Types of POU .....                   | 4 - 5                  |
| 4.2.2 Program blocks.....                  | 4 - 6                  |
| 4.2.3 Functions .....                      | 4 - 6                  |
| 4.2.4 Function blocks.....                 | 4 - 7                  |
| 4.2.5 Networks.....                        | 4 - 8                  |
| 4.2.6 Programming languages for POUs ..... | 4 - 9                  |
| 4.2.7 Functions and function blocks .....  | 4 - 10                 |
| 4.2.8 EN and ENO .....                     | 4 - 13                 |
| 4.3 Labels                                 | 4 - 14                 |
| 4.3.1 Global labels.....                   | 4 - 14                 |

|       |  |        |
|-------|--|--------|
| 4.3.2 | Local labels .....                                 | 4 - 14 |
| 4.3.3 | Label classes .....                                | 4 - 15 |
| 4.3.4 | Data types .....                                   | 4 - 16 |
| 4.4   | Device and Address .....                           | 4 - 18 |
| 4.4.1 | Device .....                                       | 4 - 18 |
| 4.4.2 | Address .....                                      | 4 - 19 |
| 4.4.3 | Correspondence between devices and addresses ..... | 4 - 20 |
| 4.5   | Arrays .....                                       | 4 - 23 |
| 4.6   | Structures .....                                   | 4 - 25 |
| 4.7   | Libraries .....                                    | 4 - 26 |
| 4.7.1 | User libraries .....                               | 4 - 27 |

|                            |                        |
|----------------------------|------------------------|
| <b>5. WRITING PROGRAMS</b> | <b>5 - 1 to 5 - 14</b> |
|----------------------------|------------------------|

|       |  |        |
|-------|--|--------|
| 5.1   | ST Language .....  | 5 - 2  |
| 5.1.1 | Standard format .....                                    | 5 - 2  |
| 5.1.2 | Operators in the ST language .....                       | 5 - 3  |
| 5.1.3 | Syntaxes in the ST language .....                        | 5 - 4  |
| 5.1.4 | Calling functions in the ST language .....               | 5 - 9  |
| 5.1.5 | Calling function blocks in the ST language .....         | 5 - 10 |
| 5.2   | Structured Ladder Language .....                         | 5 - 11 |
| 5.2.1 | Standard format .....                                    | 5 - 11 |
| 5.2.2 | Network elements in the structured ladder language ..... | 5 - 12 |

|                 |                           |
|-----------------|---------------------------|
| <b>APPENDIX</b> | <b>App - 1 to App - 8</b> |
|-----------------|---------------------------|

|              |   |         |
|--------------|---|---------|
| Appendix 1   | Character Strings that cannot be Used in Label Names and Data Names ..... | App - 2 |
| Appendix 2   | Recreating Ladder Programs .....  | App - 4 |
| Appendix 2.1 | Procedure for creating a structured program .....                         | App - 4 |
| Appendix 2.2 | Example of creating a structured program .....                            | App - 5 |

|              |                               |
|--------------|-------------------------------|
| <b>INDEX</b> | <b>index - 1 to index - 2</b> |
|--------------|-------------------------------|

## MANUALS

### Related manuals

The manuals related to this product are shown below.

Refer to the following tables when ordering required manuals.

#### (1) Structured programming

| Manual name   | Manual number<br>(Model code) |
|---|-------------------------------|
| QCPU Structured Programming Manual (Common Instructions)<br>Explains the specifications and functions of sequence instructions, basic instructions, and application instructions that can be used in structured programs.<br>(Sold separately)                    | SH-080783ENG<br>(13JW07)      |
| QCPU Structured Programming Manual (Application Functions)<br>Explains the specifications and functions of application functions that can be used in structured programs.<br>(Sold separately)  | SH-080784ENG<br>(13JW08)      |
| QCPU Structured Programming Manual (Special Instructions)<br>Explains the specifications and functions of instructions for network modules, intelligent function modules, and PID control functions that can be used in structured programs.<br>(Sold separately) | SH-080785ENG<br>(13JW09)      |

#### (2) Operation of GX Works2

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

### POINT

The operating manual is included in the CD-ROM with the software package. Manuals in printed form are sold separately. Order a manual by quoting the manual number (model code) listed in the table above.

## PURPOSE OF THIS MANUAL

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

Manuals for reference are listed in the following table according to their purpose.

For information such as the contents and number of each manual, refer to the list of 'Related manuals'.

### (1) Operation of GX Works2

| Purpose                         |   | GX Works2 Installation Instructions   | GX Works2 Beginner's Manual   |   | GX Works2 Version1 Operating Manual   |   |   |
|---------------------------------|---|---|---|---|---|---|---|
|                                 |   |  |  |    |    |    |    |
|                                 |   | -   | Simple Project  | Structured Project  | Common  | Simple Project  | Structured Project  |
| Installation                    | Learning the operating environment and installation method          |  |   |   |   |   |   |
| Operation of Simple project     | Learning the basic operations and operating procedures              |   |  |   |    |    |   |
|                                 | Learning the functions and operation methods for programming        |   |   |   |  |   |   |
|                                 | Learning all functions and operation methods except for programming |   |   |   |  |   |   |
| Operation of Structured project | Learning the basic operations and operating procedures              |   |   |  |  |   |  |
|                                 | Learning the functions and operation methods for programming        |   |   |   |  |  |  |
|                                 | Learning all functions and operation methods except for programming |   |   |   |  |   |   |

(2) Programming

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

## GENERIC TERMS AND ABBREVIATIONS IN THIS MANUAL

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

| Generic term and abbreviation | Description  |
|-------------------------------|--|
| GX works2                     | Generic product name for the SWnDNC-GXW2-E<br>(n: version)   |
| GX Developer                  | Generic product name for the SWnD5C-GPPW-E, SWnD5C-GPPW-EA, SWnD5C-GPPW-EV, and SWnD5C-GPPW-EVA<br>(n: version)  |
| GX IEC Developer              | Generic product name for the SWnD5C-MEDOC3-E<br>(n: version)   |
| CPU module                    | Generic term for the High Performance model QCPU and Universal model QCPU  |
| High Performance model QCPU   | Generic term for the Q02, Q02H, Q06H, Q12H, and Q25H   |
| Universal model QCPU          | Generic term for the Q02U, Q03UD, Q03UDE, Q04UDH, Q04UDEH, Q06UDH, Q06UDEH, Q13UDH, Q13UDEH, Q26UDH, and Q26UDEH |
| Personal Computer             | Generic term for personal computer on which Windows <sup>®</sup> operates  |
| IEC61131-3                    | Abbreviation for the IEC 61131-3 international standard  |
| Common instruction            | Generic term for the sequence instructions, basic instructions, and application instructions                     |
| Special instruction           | Generic term for the PID control instructions and module dedicated instructions                                  |

# 1

# OVERVIEW

---

|     |   |     |
|-----|---|-----|
| 1.1 | Overview . . . . .                        | 1-2 |
| 1.2 | Features of Structured Programs . . . . . | 1-2 |
| 1.3 | Applicable CPU Modules . . . . .          | 1-3 |
| 1.4 | Compatible Software Package . . . . .     | 1-3 |

# 1.1 Overview

---

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

## 1.2 Features of Structured Programs

---

This section explains the features of structured programs.

### (1) Structured design

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

The followings 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 followings are the advantages of creating structured programs.

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

### (2) Multiple programming languages

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

- Different programming language can be used for each program component.

Table 1.2-1 Programming languages that can be used for structured programs

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

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

 Section 4.2.6. Programming languages for POU's

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

 Chapter 5. WRITING PROGRAMS

The ladder languages used in the existing GX Developer and Simple project can be used.

For details on writing programs, refer to the following manuals.

 Programming manuals for each CPU

### (3) Improved program reusability

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

## 1.3 Applicable CPU Modules

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

Table 1.3-1 Applicable CPU modules

| Programmable controller CPU type |   |
|----------------------------------|---|
| High Performance model QCPU      | Q02, Q02H, Q06H, Q12H, Q25H   |
| Universal model QCPU             | Q02U, Q03UD, Q03UDE, Q04UDH, Q04UDEH, Q06UDH, Q06UDEH, Q13UDH, Q13UDEH, Q26UDH, Q26UDEH |

## 1.4 Compatible Software Package

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

Table 1.4-1 Compatible software package

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

### (1) What is GX Works2?

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

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

### POINT

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



# 2

## STRUCTURED DESIGN OF SEQUENCE PROGRAMS

---

|     |  |     |
|-----|--|-----|
| 2.1 | What is a Hierarchical Sequence Program? . . . . . | 2-2 |
| 2.2 | What is a Structured Sequence Program? . . . . .   | 2-3 |

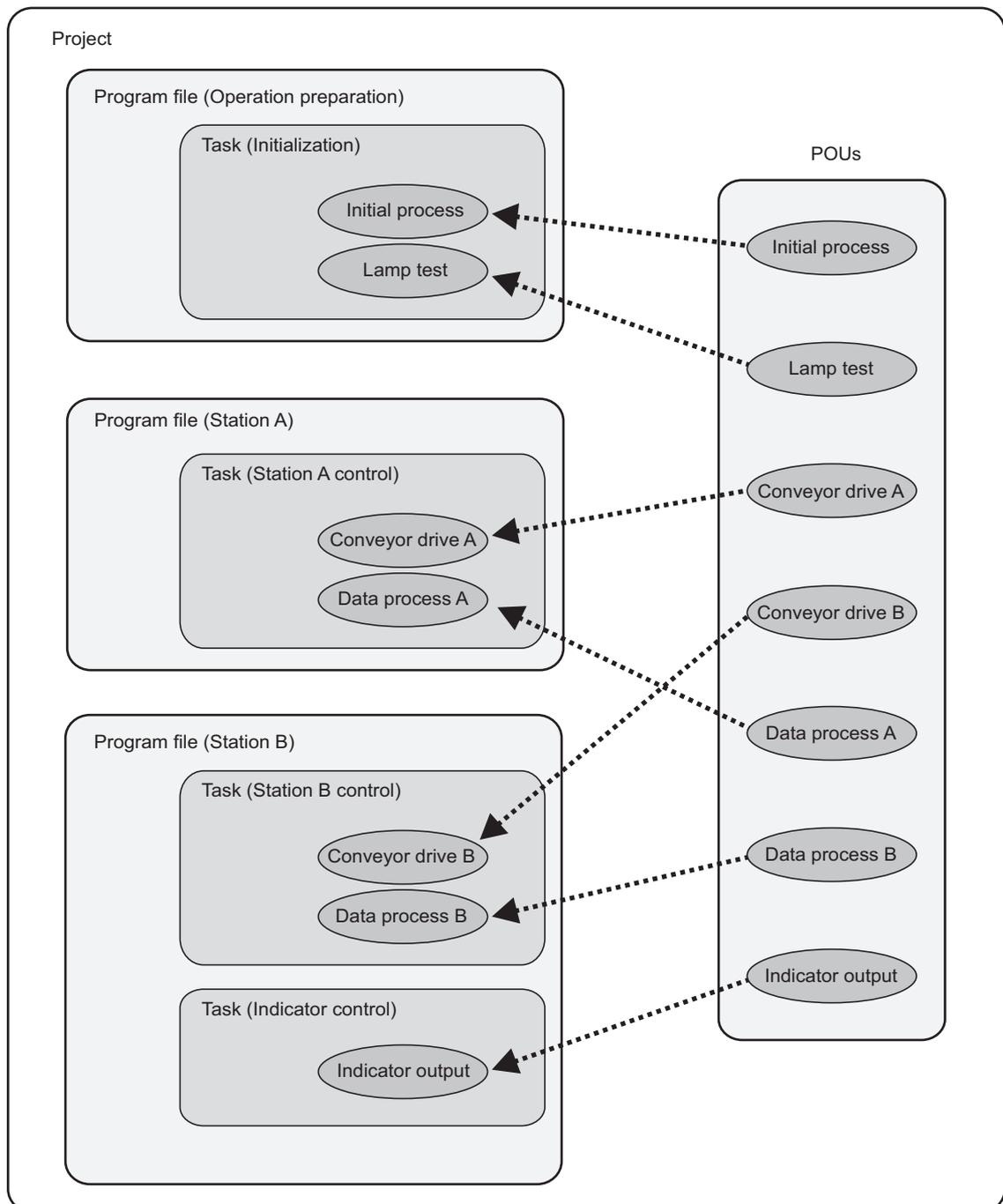
## 2.1 What is a Hierarchical Sequence Program?

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

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

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

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



## 2.2 What is a Structured Sequence Program?

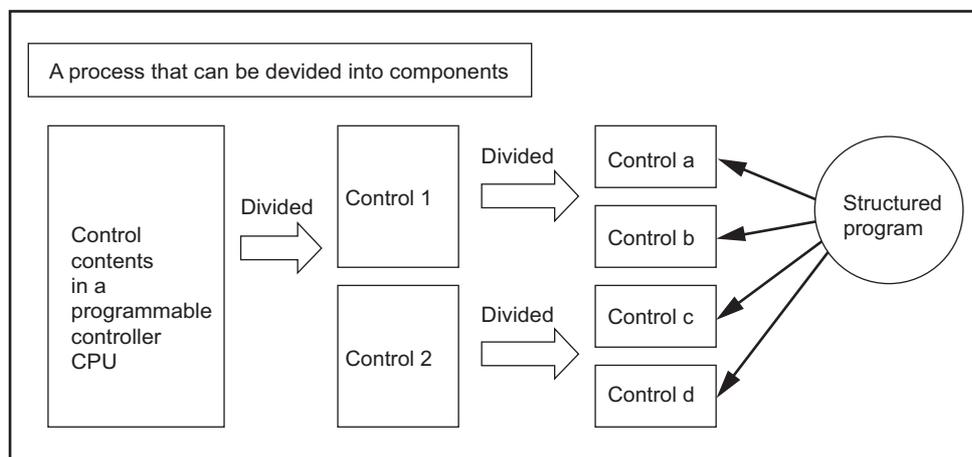
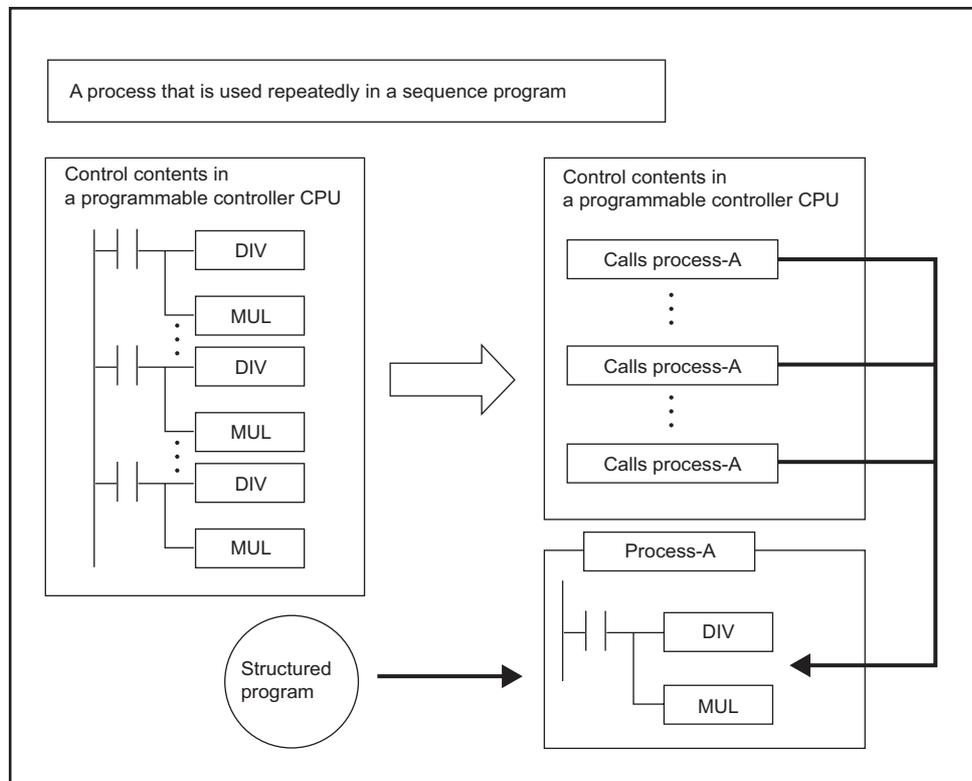
A structured program is a program created by components. Processes in lower levels of hierarchical sequence program are divided to several components according to their processing informations and functions.

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

---

3.1 Procedure for Creating Sequence Programs in Structured Project . . . . . 3-2

1 OVERVIEW

2 STRUCTURED DESIGN OF SEQUENCE PROGRAMS

**3 PROCEDURE FOR CREATING PROGRAMS**

4 PROGRAM CONFIGURATION

5 WRITING PROGRAMS

A APPENDIX

INDEX

# 3.1 Procedure for Creating Sequence Programs in Structured Project

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

## (1) Creating a program structure

| Procedure             |
|-----------------------|
| Create program files. |
| Create tasks.         |



## (2) Creating POUs

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



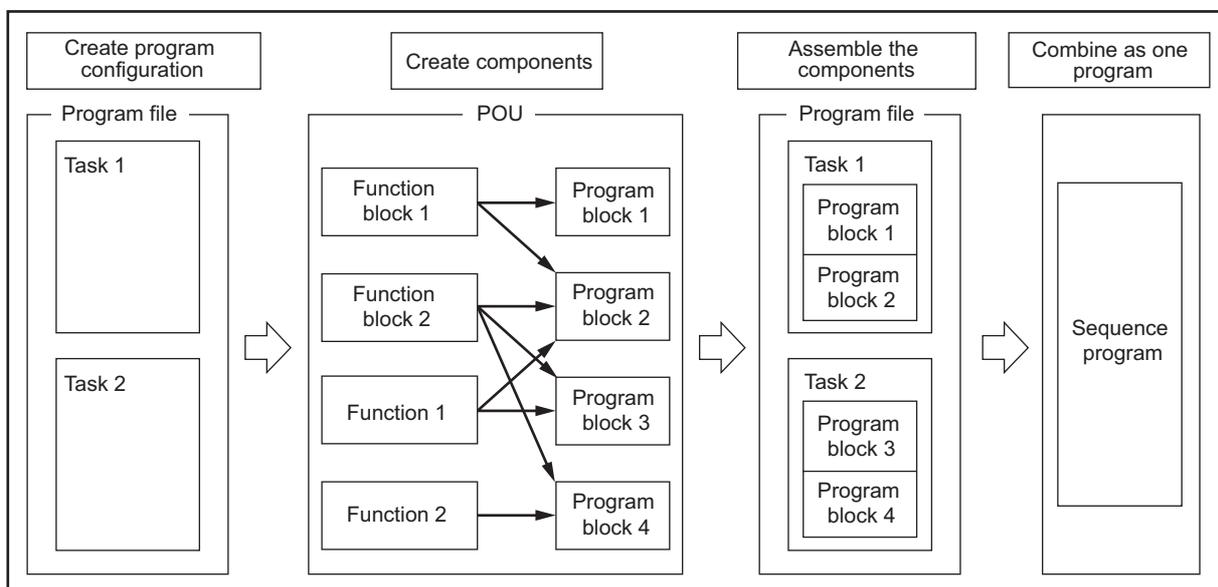
## (3) Editing the programs

| Procedure                      |
|--------------------------------|
| Edit the programs of each POU. |



## (4) Compiling the programs

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



# 4

## PROGRAM CONFIGURATION

---

|     |   |      |
|-----|---|------|
| 4.1 | Overview of Program Configuration . . . . . | 4-2  |
| 4.2 | POUs . . . . .                              | 4-5  |
| 4.3 | Labels . . . . .                            | 4-14 |
| 4.4 | Device and Address . . . . .                | 4-18 |
| 4.5 | Arrays . . . . .                            | 4-23 |
| 4.6 | Structures . . . . .                        | 4-25 |
| 4.7 | Libraries . . . . .                         | 4-26 |

# 4.1 Overview of Program Configuration

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

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

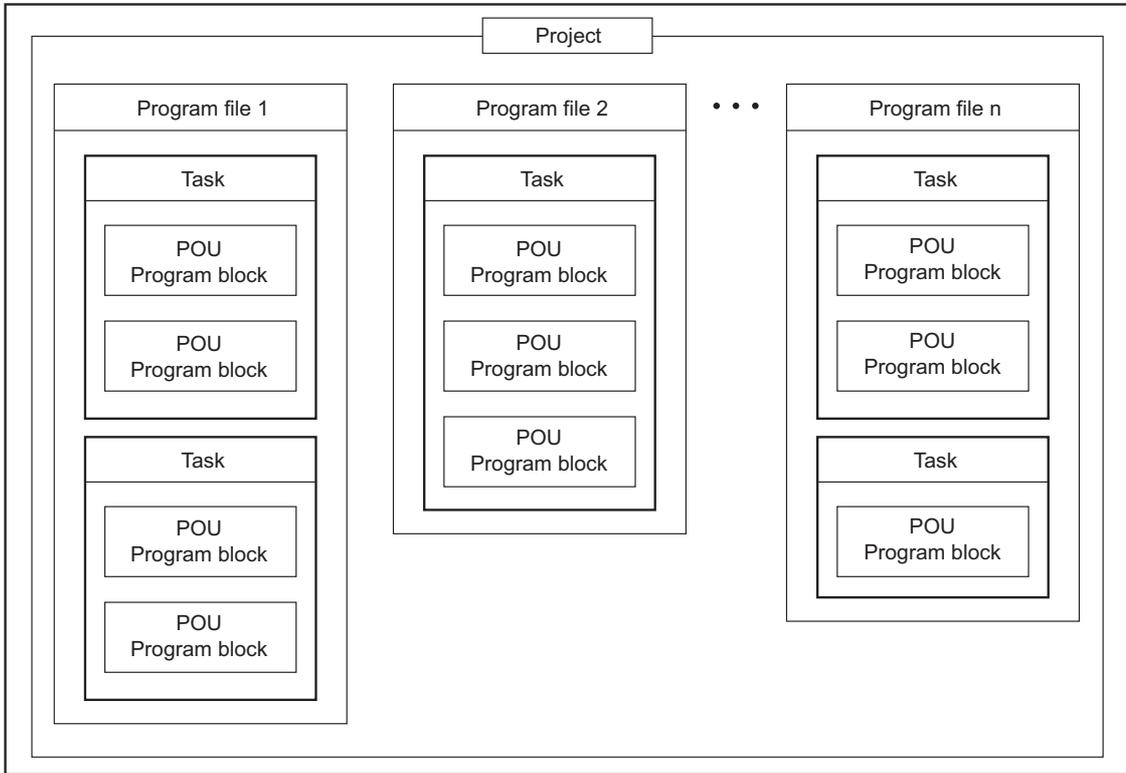
For projects:  Section 4.1.1 Project

For program files:  Section 4.1.2 Program files

For tasks:  Section 4.1.3 Tasks

For POUs:  Section 4.2 POUs

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



## 4.1.1 Project

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

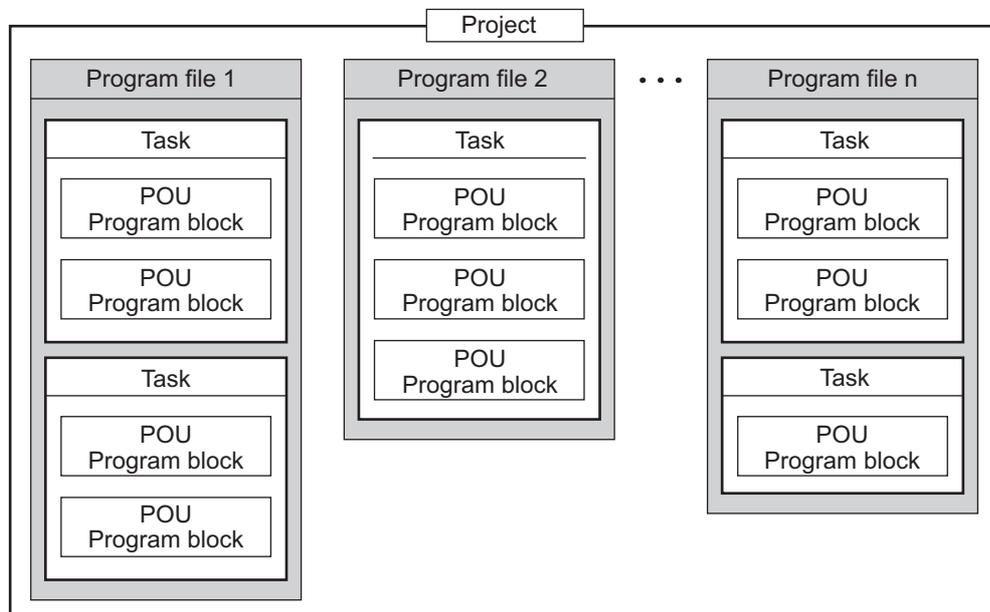
One or more program files need to be created in a project.

## 4.1.2 Program files

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

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

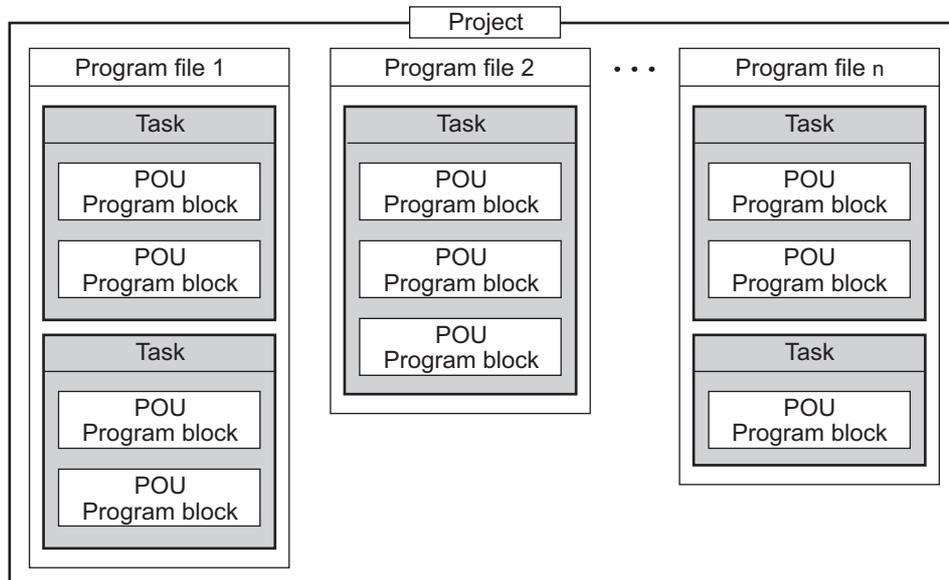
For details on the execution types set in the parameter, refer to the user's manual of each CPU module.



## 4.1.3 Tasks

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

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



### (1) Task executing condition

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

The followings are the types of task executing condition.

- (a) Scan execution (Default executing condition)  
Executes registered program blocks for each scan.
- (b) Event execution  
Executes tasks when values are set to the corresponding devices or labels.
- (c) Fixed scan execution  
Executes tasks in a specified cycle.

A priority can be set for each task execution.

- Priority

When executing conditions of multiple tasks are met simultaneously, the tasks are executed according to the set priority.

Tasks are executed in the order from the smallest priority level number.

Tasks set with a same priority level number are executed in the order of task data name.

## 4.2 POU

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

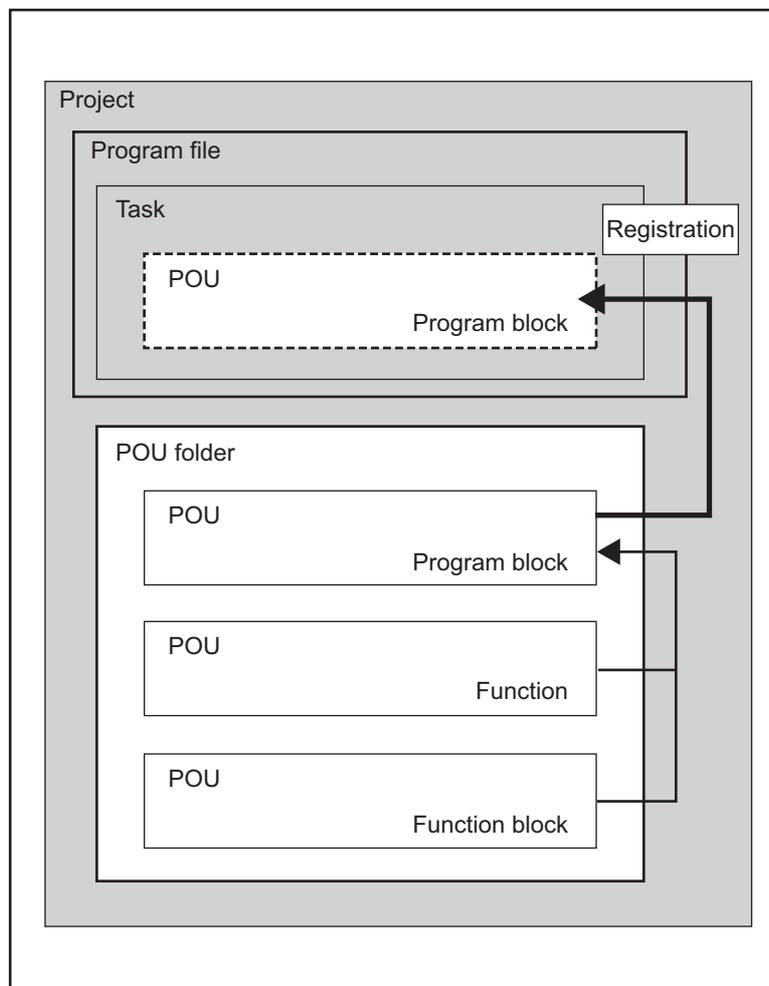
### 4.2.1 Types of POU

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

- Program block
- Function
- Function block

Each POU consists of local labels<sup>\*1</sup> and a program.

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

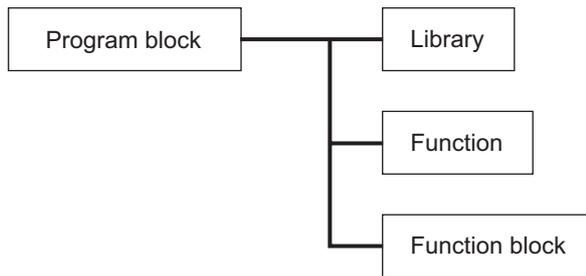


\*1 Local labels are labels that can be used only in programs of declared POU. For details of local labels, refer to the following section.

☞ Section 4.3.2 Local labels

## 4.2.2 Program blocks

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



Sequence programs executed in a programmable controller CPU are created by program blocks of POU.

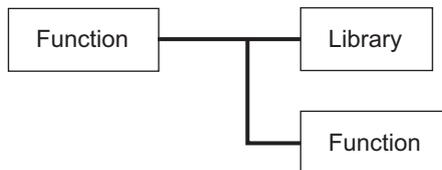
For a simplest sequence program, only one program block needs to be created and registered to a task in order to be executed in a programmable controller CPU.

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

## 4.2.3 Functions

Libraries and functions are used to edit functions.

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



Functions always output same processing results for same input values.

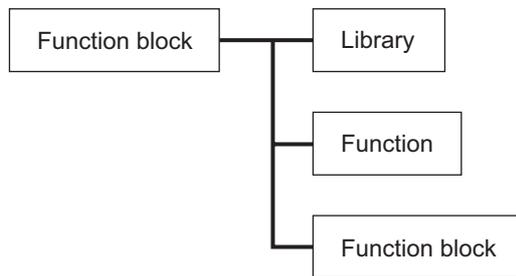
By defining simple and independent algorithms that are frequently used, functions can be reused efficiently.

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

## 4.2.4 Function blocks

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

Function blocks can be used by calling them from program blocks or function blocks. Note that they cannot be called from functions.



Function blocks can retain the input status since they can store values in internal and output variables. Since they use retained values for the next processing, they do not always output the same results even with the same input values.

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

- **Instantiation**

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

☞ Section 4.2.7 Functions and function blocks

### ☒ POINT

Instances are variables representing devices assigned to labels of function blocks.

Devices are automatically assigned when instances are created with local labels.

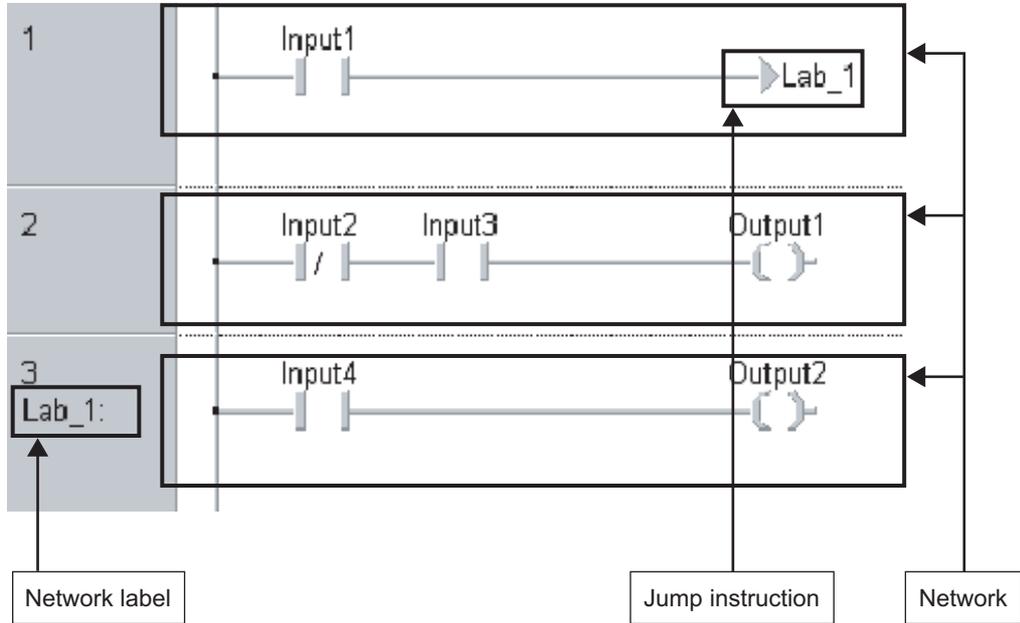
## 4.2.5 Networks

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

In the ST language, networks are not used.

- Network labels

A network label can be set to a network. A network label is used to indicate a jump target for the Jump instruction.



## 4.2.6 Programming languages for POU

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

The following explains the features of each programming language.

### (1) ST: Structured text

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

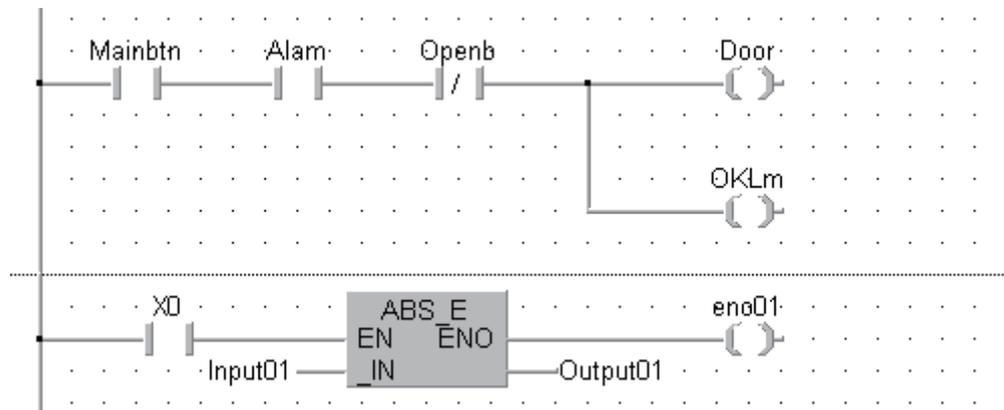
```
intV2 := ABS( intV1);  
  
IF M1 THEN  
    btn01 := TRUE;  
ELSE  
    btn01 := FALSE;  
END_IF;  
  
Output_ENO := ENEG(btn01, Input1);
```

### (2) Structured ladder: (ladder diagram)

The structured ladder language is a graphic language developed based on the relay ladder programming technique. Since it can be understood intuitively, it is commonly used for the sequence programming.

Ladders always start from the base line on the left.

A program written in the structured ladder language is composed of contacts, coils, function blocks, and functions. These elements are connected by vertical and horizontal lines.



## 4.2.7 Functions and function blocks

The following table shows differences between functions and function blocks.

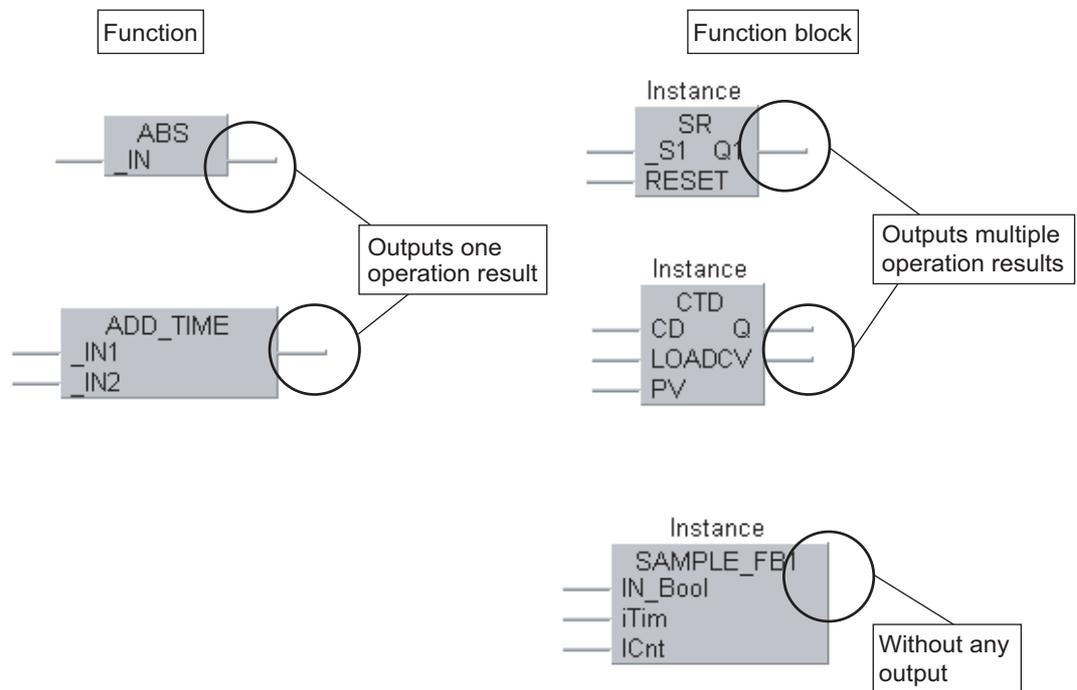
Table 4.21 Differences between functions and function blocks

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

### (1) Output variable assignment

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

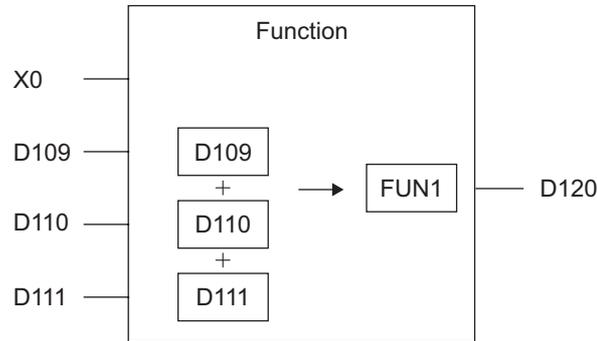
A function block can output multiple operation results. It also can be created without any output.



(2) Internal variables

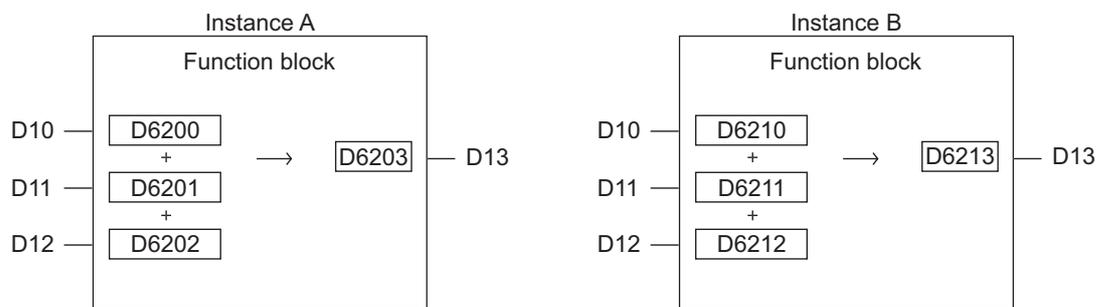
A function does not use internal variables. It uses devices assigned directly to each input variable and repeats operations.

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



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

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

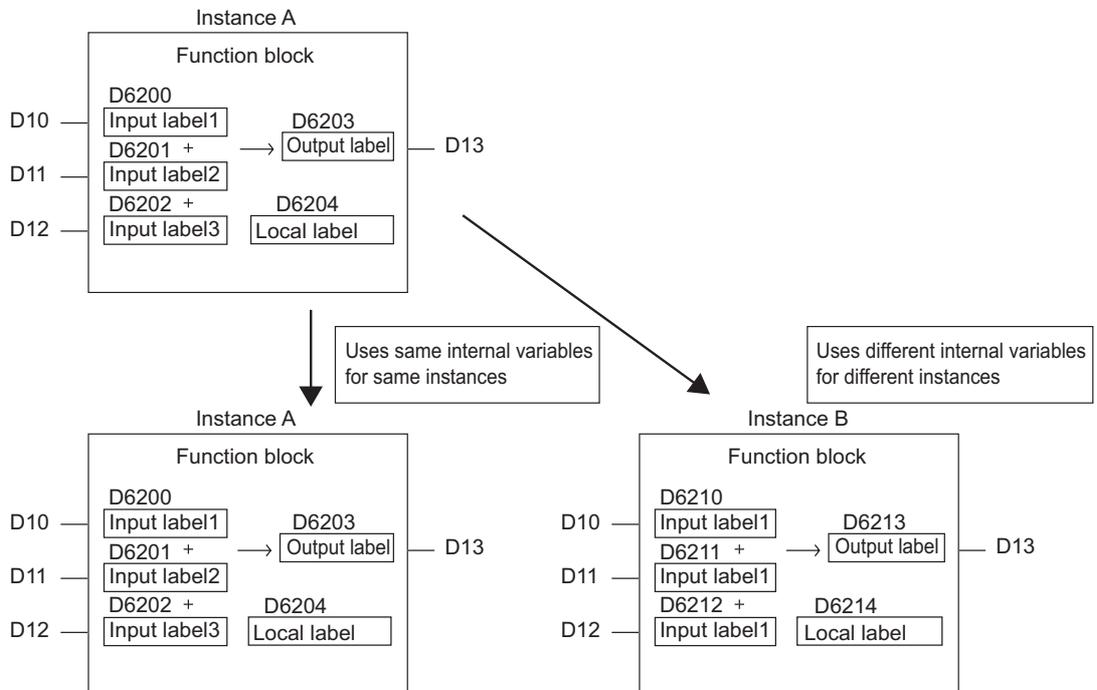


### (3) Creating instances

When using function blocks, create instances to reserve internal variables.

Variables can be called from program blocks and other function blocks by creating instances for function blocks.

To create an instance, declare as a label in a global label or local label of POU that uses function blocks. Same function blocks can be instantiated with different names in a single POU.



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

## 4.2.8 EN and ENO

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

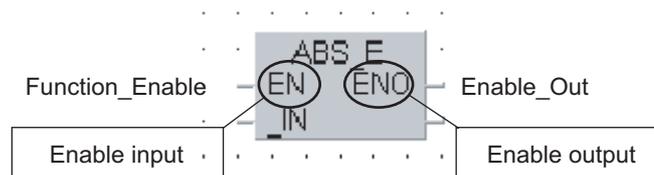
A Boolean variable used as an executing condition of a function is set to an EN.

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

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

An ENO outputs TRUE when the execution of the function is normally completed. It outputs FALSE when the process of the function is abnormally ended.

- Example of a function with EN



In the example above, the ABS\_E function is executed only when the Boolean type label 'Function\_Enable' is TRUE.

If the function is executed normally, the Boolean type label 'Enable\_Out' outputs TRUE.

### ❏ POINT

A setting of an output label to an ENO is not essential.

## 4.3 Labels

---

Labels include global labels and local labels.

### 4.3.1 Global labels

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

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

### 4.3.2 Local labels

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

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

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

### 4.3.3 Label classes

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

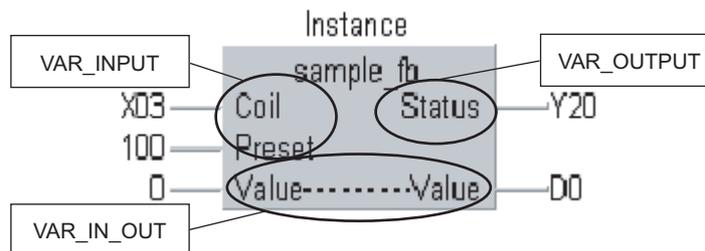
The following table shows label classes.

Table 4.3.3-1 Label classes

| Class               | Description   | Applicable POU |          |                |
|---------------------|---|----------------|----------|----------------|
|                     |   | Program block  | Function | Function block |
| VAR_GLOBAL          | Common label that can be used in program blocks and function blocks   | ○              | ×        | ○              |
| VAR_GLOBAL_CONSTANT | Common constant that can be used in program blocks and function blocks  | ○              | ×        | ○              |
| VAR                 | Label that can be used within the range of declared POU<br>This label cannot be used in other POU.            | ○              | ○        | ○              |
| VAR_CONSTANT        | Constant that can be used within the range of declared POU<br>This constant cannot be used in other POU.      | ○              | ○        | ○              |
| VAR_RETAIN          | Latch type label that can be used within the range of declared POU<br>This label cannot be used in other POU. | ○              | ×        | ○              |
| VAR_INPUT           | Label that receives a value<br>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<br>This label can be changed in a POU.     | ×              | ×        | ○              |

#### POINT

- Input variables, output variables, and input/output variables  
 VAR\_INPUT is an input variable for functions and function blocks, and  
 VAR\_OUTPUT is an output variable for function blocks.  
 VAR\_IN\_OUT can be used for both input and output variables.



## 4.3.4 Data types

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

### (1) Elementary data types

The following data types are available as the elementary data type.\*1

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

Table 4.3.4-1 Elementary data types

| Elementary data type                 | Description              | Value range  | Bit length |
|--------------------------------------|--------------------------|--|------------|
| Bit                                  | Bool                     | 0 (FALSE), 1 (TRUE)  | 1 bit      |
| Word (signed)                        | Integer                  | -32768 to 32767  | 16 bits    |
| Double word (signed)                 | Double-precision integer | -2147483648 to 2147483647                                  | 32 bits    |
| Word (unsigned)/16-bit string        | 16-bit string            | 0 to 65535   | 16 bits    |
| Double word (unsigned)/32-bit string | 32-bit string            | 0 to 4294967295  | 32 bits    |
| Single-precision real                | Real                     | $-2^{128}$ to $-2^{-126}$ , 0, $2^{-126}$ to $2^{128}$     | 32 bits    |
| Double-precision real*2              | Double-precision real    | $-2^{1024}$ to $-2^{-1022}$ , 0, $2^{-1022}$ to $2^{1024}$ | 64 bits    |
| String                               | Character string         | Maximum 255 characters                                     | Variable   |
| Time*3                               | Time value               | T#-24d-0h31m23s648ms to T#24d20h31m23s647ms                | 32 bits    |

\*1: The following data types cannot be used for the structured ladder and ST languages. They can be only used for the ladder language.

- Timer data type: Handles programmable controller CPU timer devices (T).
- Retentive timer data type: Handles programmable controller CPU retentive timer devices (ST).
- Counter data type: Handles programmable controller CPU counter devices (C).
- Pointer data type: Handles programmable controller CPU pointer devices (P).

\*2 Can be used for the Universal model QCPU only.

\*3 The time type is used in time type operation instructions of application function. For details of the application functions, refer to the following manual.

 QCPU Structured Programming Manual (Application Functions)

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

Table 4.3.4-2 Constant expressing method

| Constant type    | Expressing method  | Example            |
|------------------|--|--------------------|
| Bool             | Input FALSE or TRUE, or input 0 or 1.  | TRUE, FALSE        |
| Binary           | Append '2#' in front of a binary number.   | 2#0010, 2#01101010 |
| Octal            | Append '8#' in front of an octal number.   | 8#0, 8#337         |
| Decimal          | Directly input a decimal number, or append 'K' in front of a decimal number.   | 123, K123          |
| Hexadecimal      | Append '16#' or 'H' in front of a hexadecimal number. When a lowercase letter 'h' is appended, it is converted to uppercase automatically. | 16#FF, HFF         |
| Real number      | Directly input a real number or append 'E' in front of a real number.  | 2.34, E2.34        |
| Character string | Enclose a character string with single quotations (') or double quotations (").  | 'ABC', "ABC"       |

(2) Generic data types

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

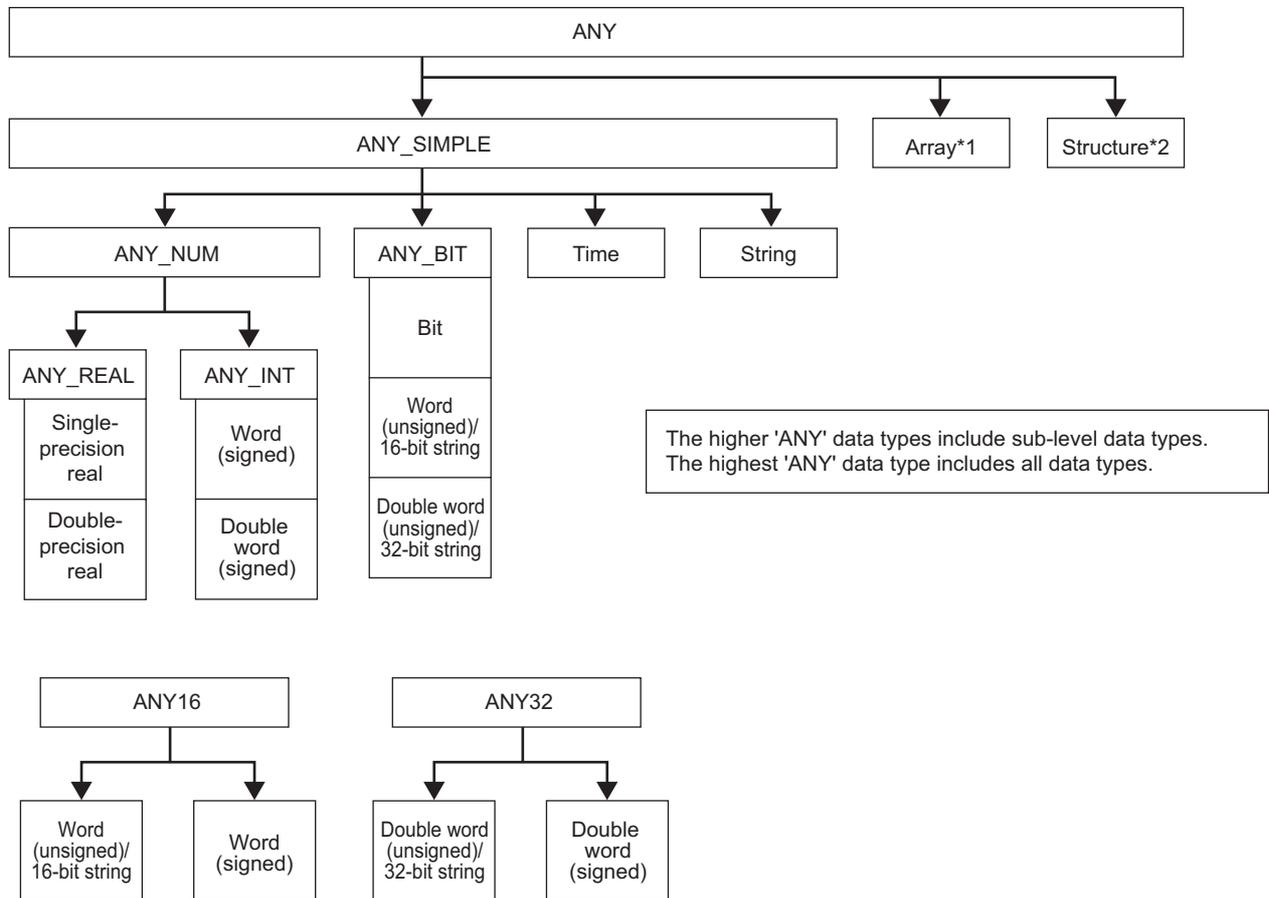
ANY data types are used when multiple data types are allowed for function arguments and return values.

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

For example, if the argument of a function is ANY\_NUM data type, desired data type for an argument can be specified from word (signed) type, double word (signed) type, single-precision real type, and double-precision real type.

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

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



\*1 For arrays, refer to the following section. Section 4.5 Arrays

\*2 For structures, refer to the following section. Section 4.6 Structures

## 4.4 Device and Address

---

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

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

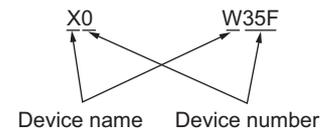
### 4.4.1 Device

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

For details of devices used in the QCPU, refer to the following manual.

 QCPU User's Manual (Function Explanation, Program Fundamentals)

Example)



## 4.4.2 Address

Address is a format defined in IEC61131-3.

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

Table 4.4.2-1 Address definition specifications

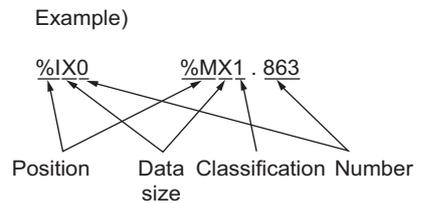
| Start | 1st character: position |          | 2nd character: data size |                       | 3rd character and later: classification   | Number   |
|-------|-------------------------|----------|--------------------------|-----------------------|---|--|
|       |                         |       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

### ● 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.

- X, JX (X device) : I (input)
- Y, JY (Y device) : Q (output)
- Other devices : M (internal)



### ● Data size

Data size is a class indicating the size of data.

The following shows the format rules corresponding to the device format.

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

### ● Classification

Classification is a minor class indicating the type of a device that cannot be identified only by its position and size.

Devices X and Y do not support classification.

For the format corresponding to the device format, refer to the following section.

☞ Section 4.4.3 Correspondence between devices and addresses

## ☒ POINT

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

## 4.4.3 Correspondence between devices and addresses

This section explains the correspondence between devices and addresses.

### (1) Correspondence between devices and addresses

The following table shows the correspondence between devices and addresses.

Table 4.4.3-1 Correspondence between devices and addresses

| 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               |                        |

Table 4.4.3-2 Correspondence between devices and addresses

| Device                |    | Expressing method |                              | Example of correspondence between device and address |                                    |
|-----------------------|----|-------------------|------------------------------|--|------------------------------------|
|                       |    | Device            | Address                      | Device   | Address                            |
| Step relay            | S  | Sn                | %MX2.n                       | S127   | %MX2.127                           |
| SFC transition device | TR | TRn               | %MX18.n                      | TR3  | %MX18.3                            |
| SFC block device      | BL | BLn               | %MX17.n                      | BL3  | %MX17.3                            |
| Link input            | J  | Jx\Xn             | %IX16.x.n                    | J1\X1FFF   | %IX16.1.8191                       |
| Link output           |    | Jx\Yn             | %QX16.x.n                    | J1\Y1FFF   | %QX16.1.8191                       |
| Link relay            |    | Jx\Bn             | %MX16.x.1.n                  | J2\B3FFF   | %MX16.2.1.16383                    |
| Link register         |    | Jx\Wn             | %MW16.x.1.n<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         |

(2) Digit specification for bit devices

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

Table 4.4.3-3 Correspondence of formats with digit specification

| Device  | Address  |
|---|--|
| K[Number of digits][Device name][Device number]<br>(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  |

(3) Bit specification for word devices

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

Table 4.4.3-4 Correspondence of formats with bit specification

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

• Correspondence examples

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

**POINT**

- Index setting, digit specification, and bit specification

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

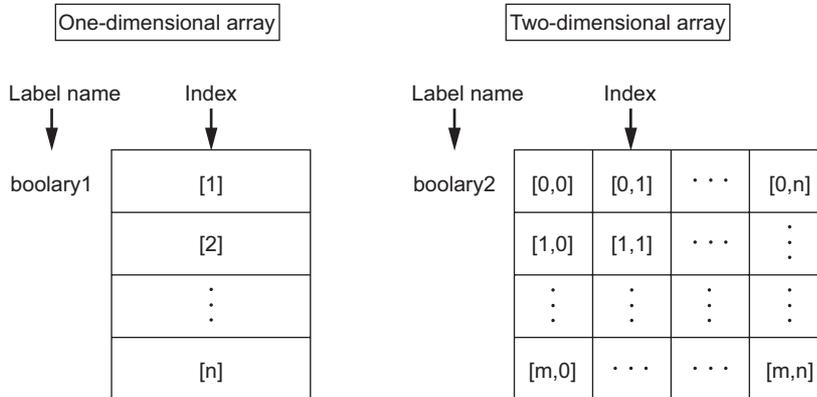
# 4.5 Arrays

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

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

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

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



## (1) Definition of arrays

The following table shows the format of definition.

Table 4.5-1 Form used to define array

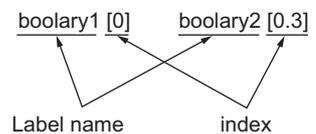
| Number of array dimensions | Format  | Remarks  |
|----------------------------|---|--|
| One dimension              | Array of elementary data type/structure name (array start value .. array end value)   | For elementary data types<br>☞ 4.3.4<br>For structured data types<br>☞ 4.6 |
|                            | (Definition example) Bit (0..2)   |  |
| Two dimensions             | Array of elementary data type/structure name (array start value .. array end value, array start value .. array end value)                                       |  |
|                            | (Definition example) Bit (0..2, 0..1)   |  |
| Three dimensions           | Array of elementary data type/structure name (array start value .. array end value, array start value .. array end value, array start value .. array end value) |  |
|                            | (Definition example) Bit (0..2, 0..1, 0..3)   |  |

## (2) Expression of arrays

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

Values that can be specified for indexes are within the range from -32768 to 32767.

Example)

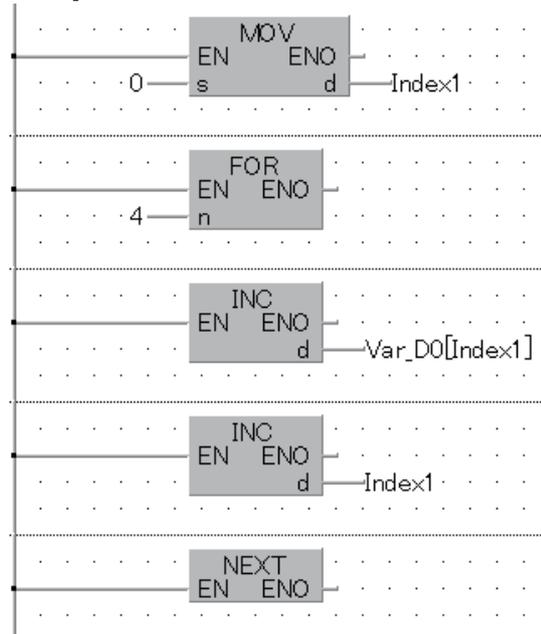


For an array with two or more dimensions, delimit indexes in '[' ]' by ','.

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

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

[Structured ladder]



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

(3) Maximum number of array elements

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

Table 4.5-2 Maximum number of array

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

# 4.6 Structures

A structure is an aggregation of different data type labels.

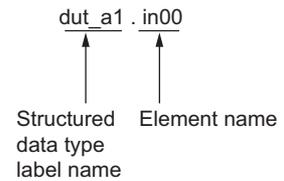
Structures can be used in all POU's.

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

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

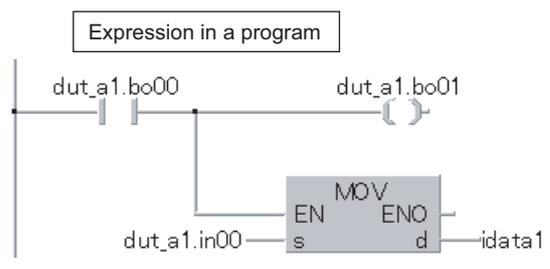
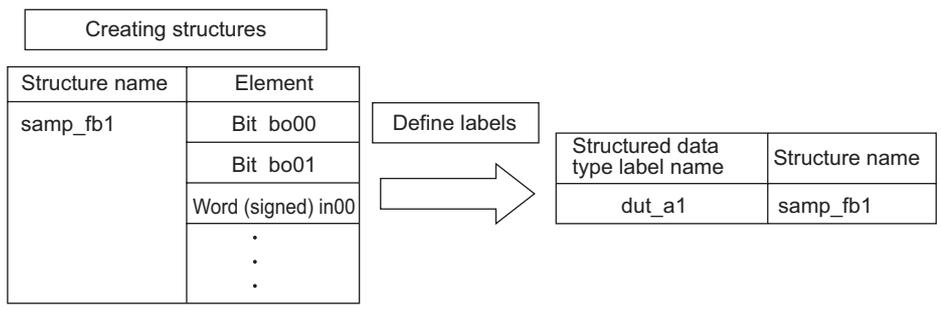
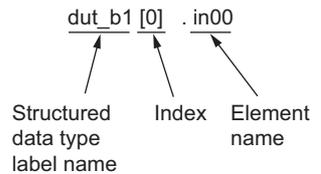
To use each element of structure, append an element name after the structured data type label name with '.' as a delimiter in between.

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 structured data type label name.

Example) When using the element of the arranged structured data



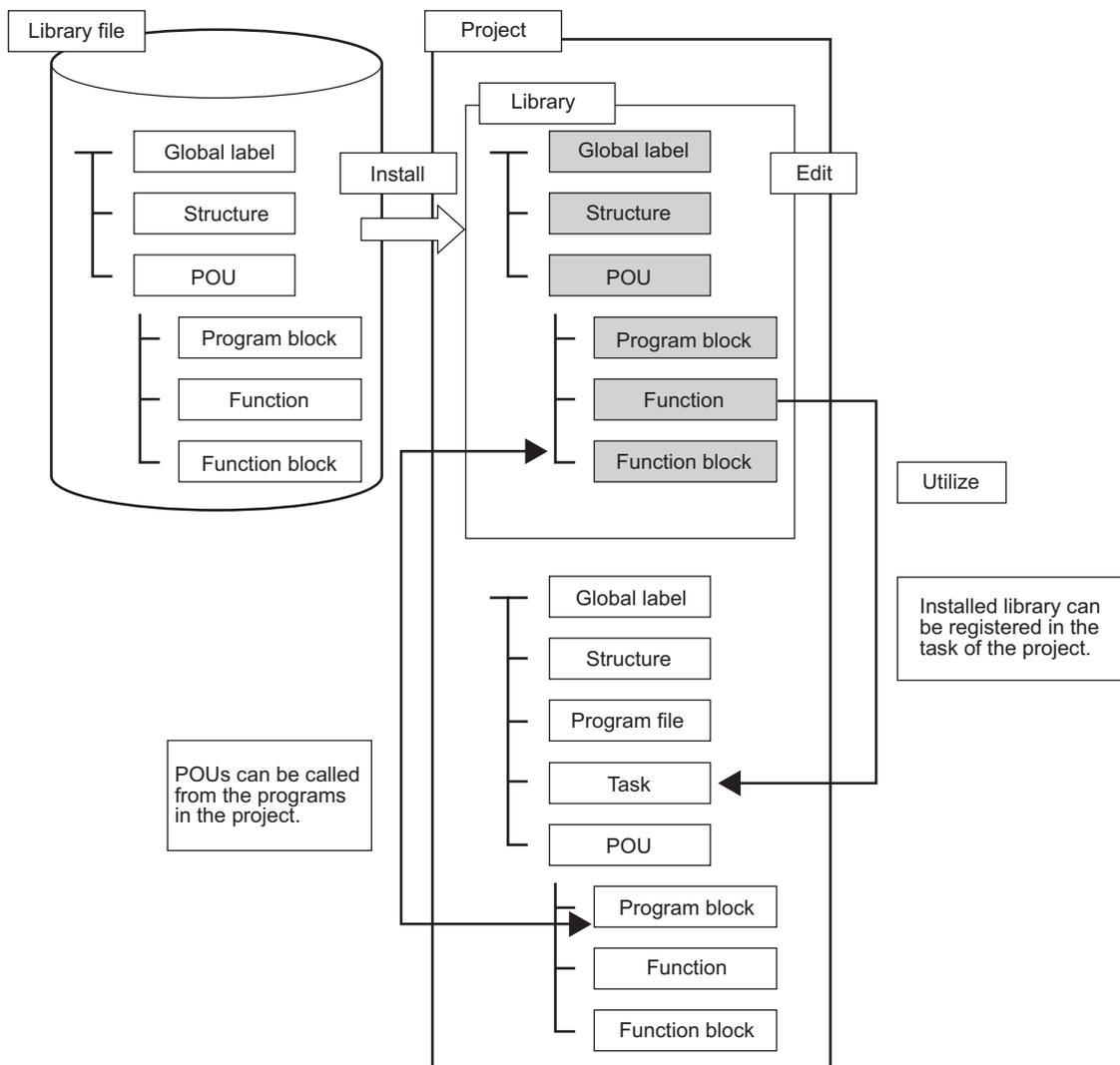
## 4.7 Libraries

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

The followings are the advantages of using libraries.

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

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



## 4.7.1 User libraries

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

(1) Composition of a user library

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

Table 4.7.1-1 Composition of a user library

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



# 5

## WRITING PROGRAMS

---

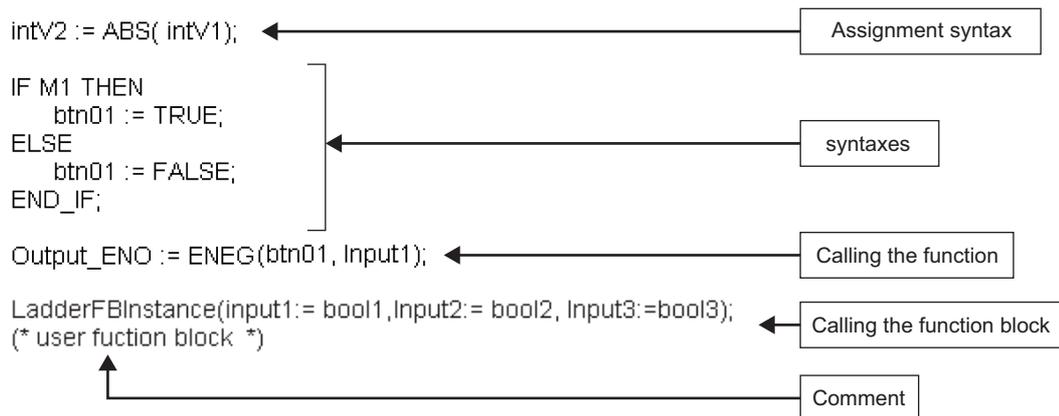
|     |                                  |      |
|-----|----------------------------------|------|
| 5.1 | ST Language .....                | 5-2  |
| 5.2 | Structured Ladder Language ..... | 5-11 |

## 5.1 ST Language

The ST language is a text language with a similar grammatical structure to the C language. Controls such as conditional judgement and repetition process written in syntaxes can be described.

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

### 5.1.1 Standard format

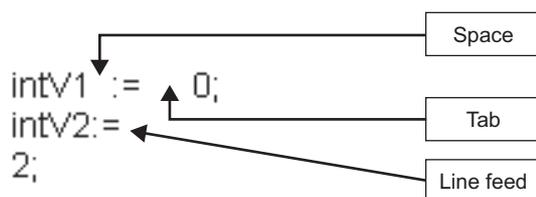


Operators and syntaxes are used for programming in the ST language.

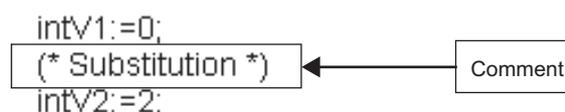
Syntaxes must end with ';'.  
Enter ';' at the end.



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.





## 5.1.3 Syntaxes in the ST language

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

Table 5.1.3-1 Syntaxes in the ST language

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

### (1) Assignment syntax

#### (b) Format

<Left side> := <Right side>;

#### (c) Description

The assignment syntax assigns the result of the right side expression to the label or device of the left side.

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

#### (d) Example

```
intv1:=0;  
intv2:=2;
```

(2) IF THEN conditional syntax

(a) Format

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

(b) Description

The syntax is executed when the value of Boolean expression (conditional formula) is TRUE. The syntax is not executed if the value of Boolean expression is FALSE. Any expression that returns TRUE or FALSE as the result of the Boolean operation with a single bit type variable status, or a complicated expression that includes many variables can be used for the Boolean expression.

(c) Example

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

(3) IF ...ELSE conditional syntax

(a) Format

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

(b) Description

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

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

(c) Example

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

#### (4) IF ...ELSIF conditional syntax

##### (a) Format

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

##### (b) Description

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

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

##### (c) Example

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

(5) CASE conditional syntax

(a) Format

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

(b) Description

The result of the CASE conditional expression is returned as an integer value. The CASE conditional syntax is used to execute a selection syntax by a single integer value or an integer value as the result of a complicated expression.

When the syntax that has the integer selection value that matches with the value of integer expression is executed, and if no integer selection value is matched with the expression value, the syntax that follows the ELSE syntax is executed.

(c) Example

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

(6) FOR...DO syntax

(a) Format

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

(b) Description

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

(c) Example

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

## (7) WHILE...DO syntax

### (a) Format

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

### (b) Description

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

The Boolean expression is evaluated before the execution of the syntax. If the value of Boolean expression is FALSE, the syntax in the WHILE...DO syntax is not executed. Since a return result of the Boolean expression in the WHILE syntax requires only TRUE or FALSE, any Boolean expression that can be specified in the IF conditional syntax can be used.

### (c) Example

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

## (8) REPEAT...UNTIL syntax

### (a) Format

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

### (b) Description

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

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

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

### (c) Example

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

(9) RETURN syntax

(a) Format

```
RETURN;
```

(b) Description

The RETURN syntax is used to end a program in a middle of the process.

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

(c) Example

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

(10) EXIT syntax

(a) Format

```
EXIT;
```

(b) Description

The EXIT syntax is used only in iteration syntaxes to end the iteration syntax in a middle of the process.

When the EXIT syntax is reached during the execution of the iteration loop, the iteration loop process after the EXIT syntax is not executed. The process continues from the line after the one where the iteration syntax is ended.

(c) Example

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

## 5.1.4 Calling functions in the ST language

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

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

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

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

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

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

```
Output1 := ABS(Input1);
```

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

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

## 5.1.5 Calling function blocks in the ST language

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

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

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

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

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

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

FB definition

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

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

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

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

FB definition

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

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

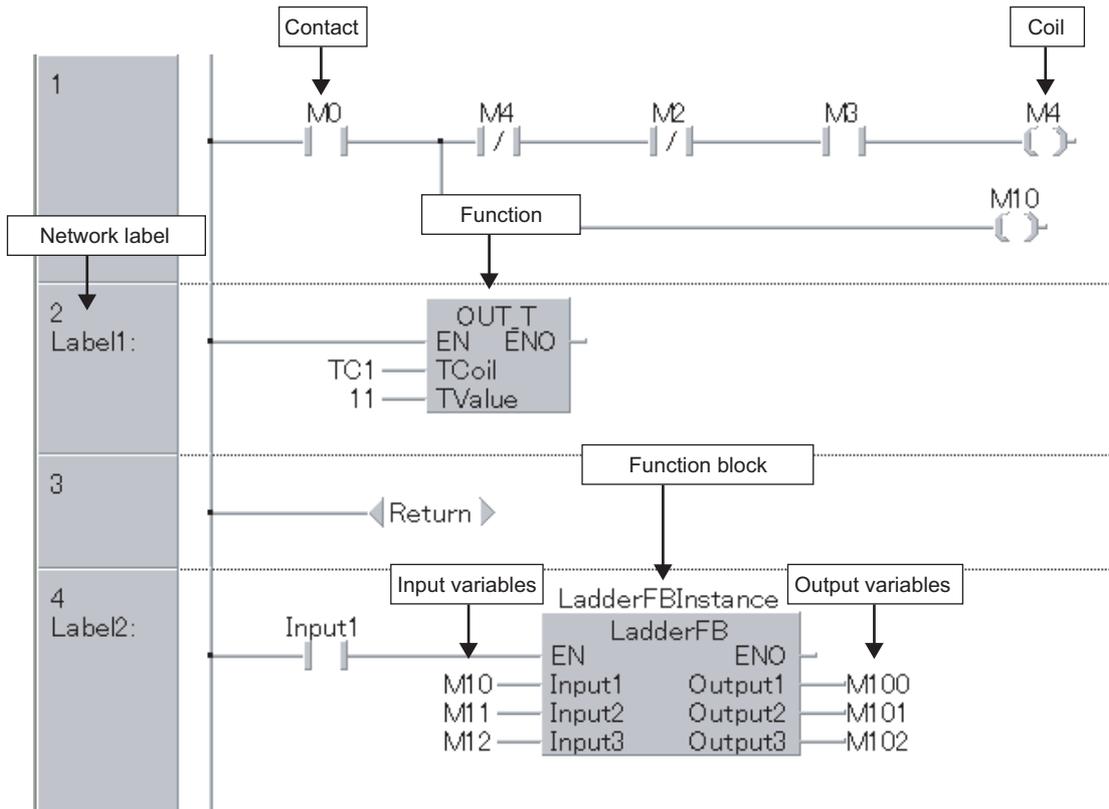
```
FBADD1(IN1:=Input1, IN2:=Input2, IN3:= Input3);  
Output1:=FBADD1.OUT1;  
Output2:=FBADD1.OUT2;
```

## 5.2 Structured Ladder Language

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

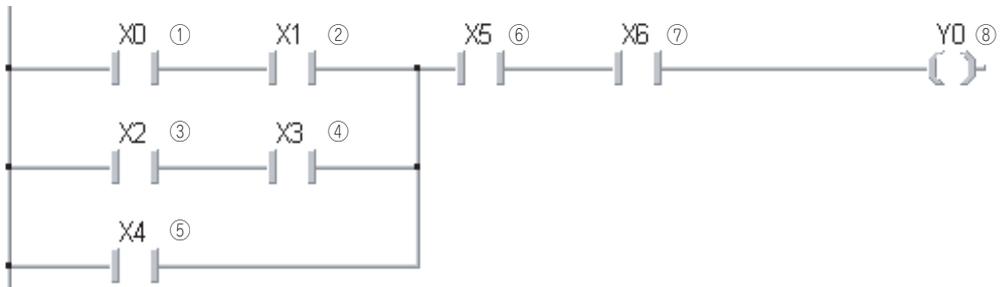
Write programs on the assumption that the power is supplied from the baseline at the left edge of the editor to the connected ladders.

### 5.2.1 Standard format



In the structured ladder language, units of network are used for programming.

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



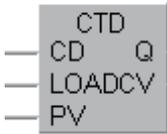
## 5.2.2 Network elements in the structured ladder language

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

Table 5.2.2-1 Network elements in the structured ladder language

| Element          | Network element | Description  |
|------------------|-----------------|--|
| Contact          |                 | Logical operation start (Contact operation start instruction)<br>Reads ON/OFF data of the specified device or label.   |
| Contact          |                 | Logical AND operation (Contact series connection)<br>Reads ON/OFF data of the specified bit device or label, and performs an AND operation between the read data and the previous operation result. The evaluated value is the operation result. |
| Contact          |                 | Logical OR operation (Contact parallel connection)<br>Reads ON/OFF data of the specified device or label, and performs an OR operation between the read data and the previous operation result. The evaluated value is the operation result.     |
| Contact negation |                 | Logical operation start (Contact negation operation start instruction)<br>Reads ON/OFF data of the specified device or label.  |
| Contact negation |                 | Logical AND operation (Contact negation series connection)<br>Reads ON/OFF data of the specified bit device or label, and performs an AND operation between the read data and the previous operation result.                                     |
| Contact negation |                 | Logical OR operation (Contact negation parallel connection)<br>Reads ON/OFF data of the specified device or label, and performs an OR operation between the read data and the previous operation result.   |
| Coil             |                 | Bit device output<br>Outputs the operation result performed up to the OUT instruction to the specified device or label.  |
| Inverted coil    |                 | Bit device output inverse<br>Inverts the status of a specified device or label when the inverse command turns from OFF to ON.  |
| Set              |                 | Device set<br>Turns ON the specified device or label when the SET input is turned ON.<br>The device or label that has been turned ON remains ON when the SET input is turned OFF.  |
| Reset            |                 | Device reset<br>Turns OFF the specified device or label when the RST input is turned ON.<br>If the RST input is OFF, the status of the device does not change.   |
| Jump             |                 | Pointer branch instruction<br>Unconditionally executes the program at the specified pointer number in the same program file.   |
| Return           |                 | Return from subroutine program<br>Indicates the end of a subroutine program. Returns the step to the next step after the instruction which called the subroutine program.  |
| Function         |                 | Executes a function.   |

Table 5.2.2-2 Network elements in the structured ladder language

| Element                               | Network element   | Description  |
|---------------------------------------|---|--|
| Function block                        | <p>Instance</p>  | Executes a function block.   |
| Function argument input               | ? —   | Inputs an argument to a function or function block.                        |
| Function return value output          | — ?   | Outputs the return value from a function or function block.                |
| Function inverted argument input      | ? — ◊   | Inverts and inputs an argument to a function or function block.            |
| Function inverted return value output | ◊ — ?   | Inverts the return value from a function or function block and outputs it. |





# APPENDIX

---

Appendix 1 Character Strings that cannot be Used in Label Names and Data Names . . . . . App-2  
Appendix 2 Recreating Ladder Programs . . . . . App-4

# Appendix 1 Character Strings that cannot be Used in Label Names and Data Names

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

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

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

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

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

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

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

(1) Precautions on using labels

- For label names and instance names, the same name as the one used for data names of task, structured data, POUs and the like cannot be used.
- A space cannot be used.
- A numeral cannot be used in the first character of a label name.
- A label name is case-sensitive during compilation.

# Appendix 2 Recreating Ladder Programs

---

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

## Appendix 2.1 Procedure for creating a structured program

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

### (1) Replacing devices with labels

| Procedure  |
|--|
| Labels include global labels and local labels.<br>Determine the type of labels (global label or local label) to replace devices. |



### (2) Setting labels

| Procedure   |
|---|
| Global labels and local labels to be used in the program must be defined.<br>Define all labels to be used in the program. |



### (3) Creating a program

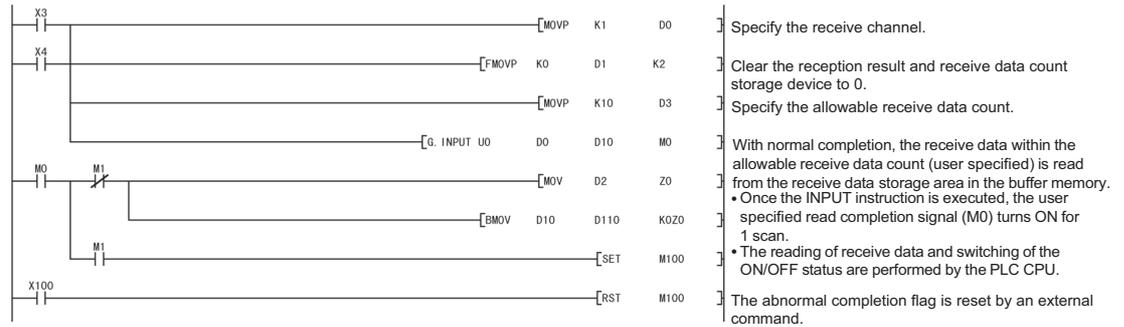
| Procedure   |
|---|
| Create a structured program in the programming language to be used. |

## Appendix 2.2 Example of creating a structured program

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

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

The following shows the original program.



### (1) Replacing devices with labels

Replace devices of the original program with labels.

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

Table App. 2.2-1 Examples of replacement from devices to labels

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

## (2) Setting labels

Set global labels and local labels.

- Setting examples of global labels

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

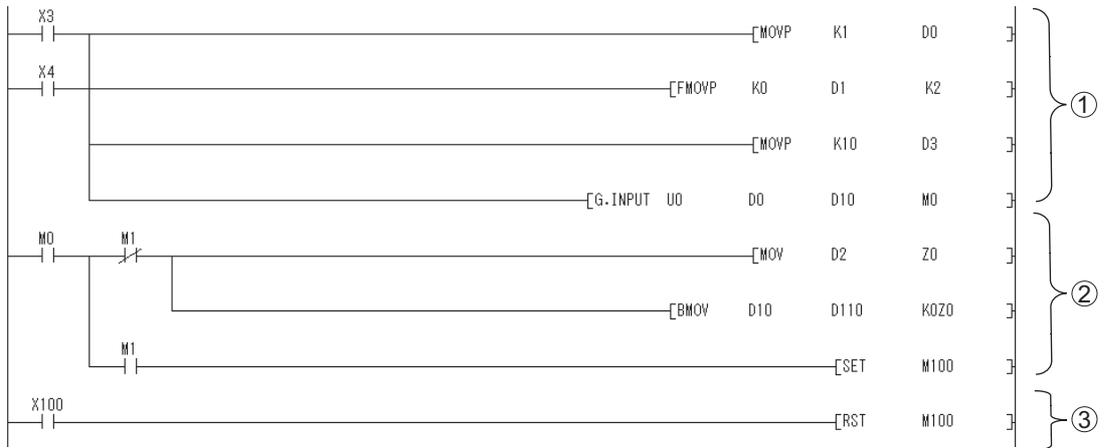
- Setting examples of local labels

|   | Class | Label Name         | Data Type                 | Constant |
|---|-------|--------------------|---------------------------|----------|
| 1 | VAR   | ControlData        | Word[Unsigned]/Bit[16Bit] | ...      |
| 2 | VAR   | ReceiveData        | Word[Unsigned]/Bit[16Bit] | ...      |
| 3 | VAR   | Completion         | Bit[0..1]                 | ...      |
| 4 | VAR   | Data               | Word[Unsigned]/Bit[16Bit] | ...      |
| 5 | VAR   | AbnormalCompletion | Bit                       | ...      |

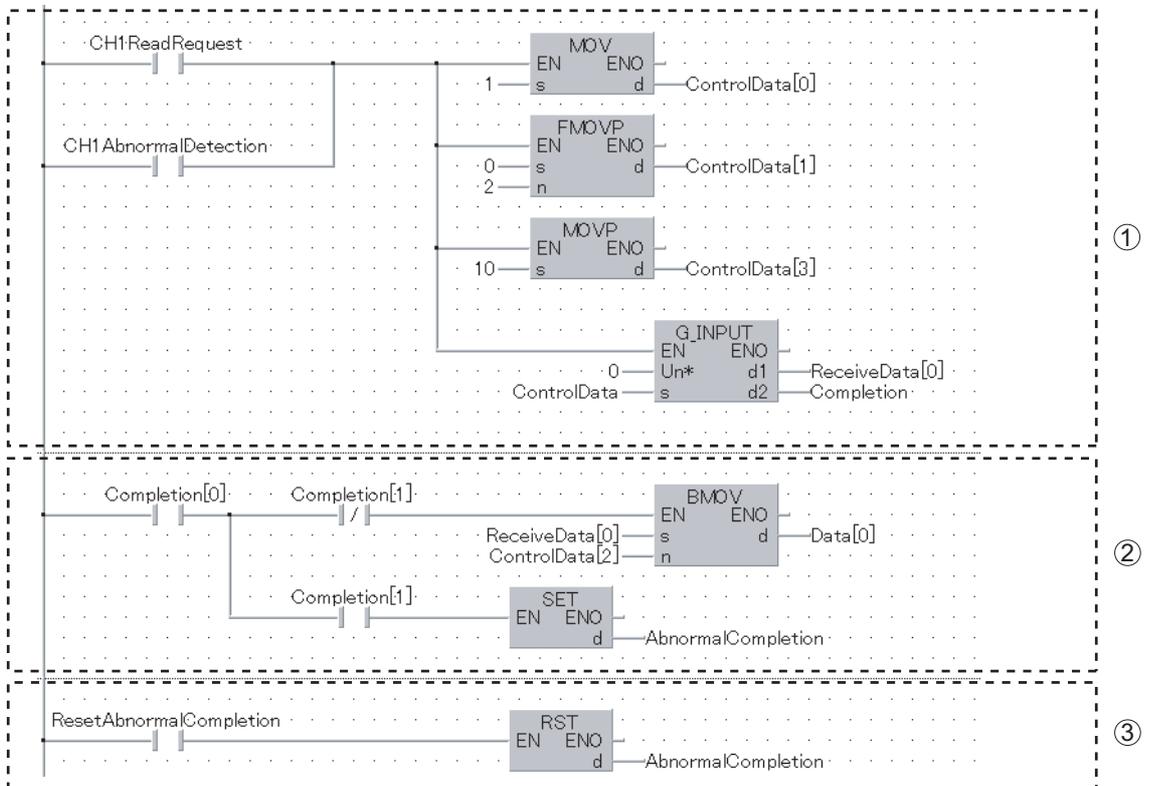
### (3) Creating a structured program

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

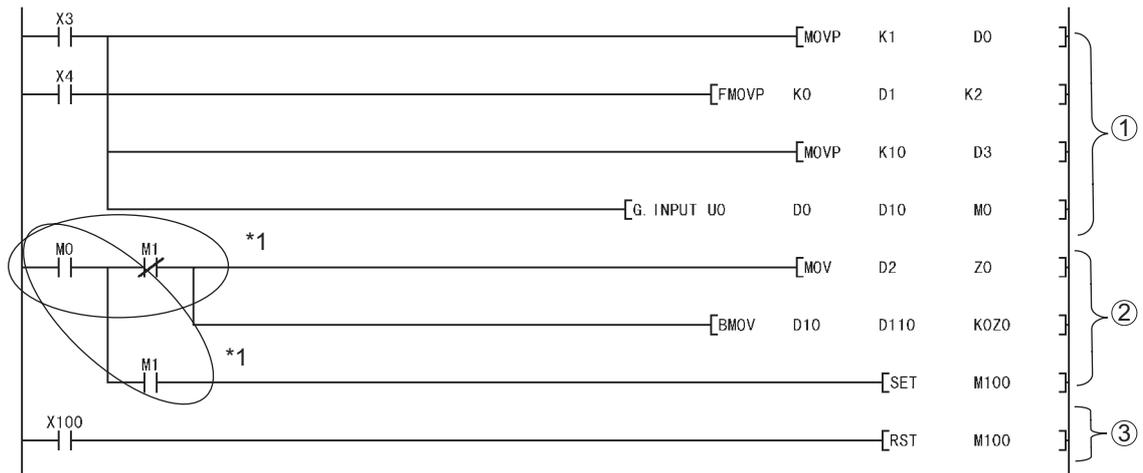
- Original program (Programming language: ladder)



- Structured program (Programming language: structured ladder)



• Original program (Programming language: ladder)



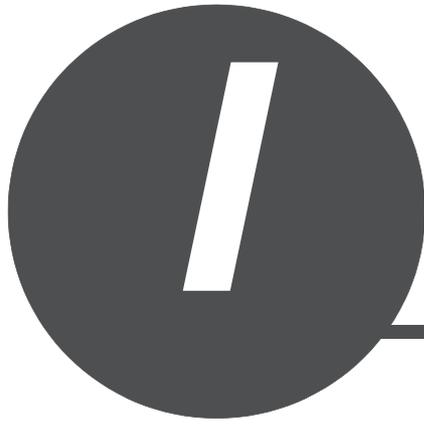
• Structured program (Programming language: ST)

```

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

```

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



# INDEX

---

|              |  |
|--------------|--|
| 1            | OVERVIEW                               |
| 2            | STRUCTURED DESIGN OF SEQUENCE PROGRAMS |
| 3            | PROCEDURE FOR CREATING PROGRAMS        |
| 4            | PROGRAM CONFIGURATION                  |
| 5            | WRITING PROGRAMS                       |
| A            | APPENDIX                               |
| <b>INDEX</b> |  |

|                               |           |
|-------------------------------|-----------|
| <b>[A]</b>                    |           |
| address .....                 | 4-19,4-20 |
| array .....                   | 4-23      |
| <b>[C]</b>                    |           |
| calling function blocks ..... | 5-10      |
| calling functions.....        | 5-9       |
| class .....                   | 4-15      |
| constant.....                 | 4-16      |
| <b>[D]</b>                    |           |
| data types.....               | 4-16      |
| device.....                   | 4-20      |
| <b>[E]</b>                    |           |
| elementary data types.....    | 4-16      |
| EN .....                      | 4-13      |
| ENO .....                     | 4-13      |
| executing condition .....     | 4-4       |
| <b>[F]</b>                    |           |
| function blocks .....         | 4-7       |
| functions.....                | 4-6       |
| <b>[G]</b>                    |           |
| generic data type.....        | 4-17      |
| global labels .....           | 4-14      |
| <b>[H]</b>                    |           |
| hierarchy .....               | 1-2,2-2   |
| <b>[I]</b>                    |           |
| input variables .....         | 4-15      |
| input/output variables .....  | 4-15      |
| instances .....               | 4-7,4-12  |
| <b>[L]</b>                    |           |
| libraries.....                | 4-26      |
| local labels .....            | 4-14      |
| <b>[N]</b>                    |           |
| network elements .....        | 5-12      |
| network label .....           | 4-8       |
| networks.....                 | 4-8       |
| <b>[O]</b>                    |           |
| operators .....               | 5-3       |
| output variables .....        | 4-10,4-15 |
| <b>[P]</b>                    |           |
| POU .....                     | 4-5       |
| priority .....                | 4-4       |
| program.....                  | 4-5       |
| program blocks.....           | 4-6       |
| program files .....           | 4-3       |
| project .....                 | 2-2,4-3   |

|                           |          |
|---------------------------|----------|
| <b>[S]</b>                |          |
| ST.....                   | 4-9      |
| standard format .....     | 5-2,5-11 |
| structured design.....    | 1-2      |
| structured ladder .....   | 4-9      |
| structured programs ..... | 1-2      |
| structures.....           | 4-25     |
| structuring.....          | 2-3      |
| syntax .....              | 5-4      |
| <b>[T]</b>                |          |
| tasks .....               | 4-4      |
| <b>[U]</b>                |          |
| user libraries.....       | 4-27     |

# **WARRANTY**

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

## **1. Gratis Warranty Term and Gratis Warranty Range**

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

### **[Gratis Warranty Term]**

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

### **[Gratis Warranty Range]**

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

## **2. Onerous repair term after discontinuation of production**

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

## **3. Overseas service**

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

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

Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation to damages caused by any cause found not to be the responsibility of Mitsubishi, loss in opportunity, lost profits incurred to the user by Failures of Mitsubishi products, special damages and secondary damages whether foreseeable or not, compensation for accidents, and compensation for damages to products other than Mitsubishi products, replacement by the user, maintenance of on-site equipment, start-up test run and other tasks.

## **5. Changes in product specifications**

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

## **6. Product application**

- (1) In using the Mitsubishi MELSEC programmable controller, the usage conditions shall be that the application will not lead to a major accident even if any problem or fault should occur in the programmable controller device, and that backup and fail-safe functions are systematically provided outside of the device for any problem or fault.
- (2) The Mitsubishi MELSEC programmable controller has been designed and manufactured for applications in general industries, etc. Thus, applications in which the public could be affected such as in nuclear power plants and other power plants operated by respective power companies, and applications in which a special quality assurance system is required, such as for Railway companies or Public service purposes shall be excluded from the programmable controller applications. In addition, applications in which human life or property that could be greatly affected, such as in aircraft, medical applications, incineration and fuel devices, manned transportation, equipment for recreation and amusement, and safety devices, shall also be excluded from the programmable controller range of applications.

However, in certain cases, some applications may be possible, providing the user consults their local Mitsubishi representative outlining the special requirements of the project, and providing that all parties concerned agree to the special circumstances, solely at the users discretion.

Microsoft, Windows are registered trademarks of Microsoft Corporation in the United States and other countries.  
Other company names and product names used in this document are trademarks or registered trademarks of respective companies.



# QCPU

## Structured Programming Manual (Fundamentals)

|                              |           |
|------------------------------|-----------|
| MODEL                        | Q-KP-KI-E |
| MODEL CODE                   | 13JW06    |
| SH(NA)-080782ENG-A(0807)KWIX |           |

 **MITSUBISHI ELECTRIC CORPORATION**

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

When exported from Japan, this manual does not require application to the Ministry of Economy, Trade and Industry for service transaction permission.

Specifications subject to change without notice.