Cat. No. W427-E1-01

SYSMAC

CX-Programmer IEC

Ver. 1.0

WS02-CPIC1-E

CS1-H (FB)/CJ1-H (FB) CPU Units

OPERATION MANUAL

OMRON

CX-Programmer IEC

Ver. 1.0
WS02-CPIC1-E

CS1-H (FB)/CJ1-H (FB) CPU Units

Operation Manual

Produced September 2003

Notice:

OMRON products are manufactured for use according to proper procedures by a qualified operator and only for the purposes described in this manual.

The following conventions are used to indicate and classify precautions in this manual. Always heed the information provided with them. Failure to heed precautions can result in injury to people or damage to property.

⚠ DANGER Indicates an imminently hazardous situation which, if not avoided, will result in death or serious injury.

⚠ WARNING Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.

⚠ Caution Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

OMRON Product References

All OMRON products are capitalized in this manual. The word “Unit” is also capitalized when it refers to an OMRON product, regardless of whether or not it appears in the proper name of the product.

The abbreviation “Ch,” which appears in some displays and on some OMRON products, often means “word” and is abbreviated “Wd” in documentation in this sense.

The abbreviation “PLC” means Programmable Controller. “PC” is used, however, in some Programming Device displays to mean Programmable Controller.

Visual Aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

1,2,3... 1. Indicates lists of one sort or another, such as procedures, checklists, etc.

No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

TABLE OF CONTENTS

TABLE OF CONTENTS

About this Manual:

This manual describes the function blocks and related functionality of the CX-Programmer IEC and includes the sections described on the next page. The CX-Programmer IEC can be used only for SYSMAC CS-series and CJ-series CPU Units that support function blocks. These CPU Units are indicated as the CS1-H (FB)/CJ1-H (FB) CPU Units.

This manual describes only CX-Programmer IEC operations that are different from those of the non-IEC CX-Programmer. For operations not related to function blocks, refer to the *CX-Programmer Operation Manual* (enclosed, Cat. No. W414). This manual also provides only specifications and information on the battery replacement procedure for the CS1-H (FB)/CJ1-H (FB) CPU Units. For other information, refer to the CS/CJ-series manuals.

Please read this manual and related manuals carefully and be sure you understand the information provided before attempting to install or operate the CX-Programmer IEC or the CS1-H (FB)/CJ1-H (FB) CPU Units. Be sure to read the precautions provided in the following section.

Manuals Related to the CX-Programmer IEC

Name	Cat. No.	Contents
SYSMAC WS02-CPIC1-E CX-Programmer IEC Operation Manual (CS1G-CPU42H/44H (FB), CS1H-CPU65H/67H (FB), CJ1G-CPU42H/43H/44H (FB) CPU Units)	W427	(This manual) Describes the functionality unique to the CX-Programmer IEC based on function blocks. Functionality that is the same as that of the CX-Programmer is described in W414 (enclosed).
SYSMAC WS02-CXPC1-E-V3□ CX-Programmer Operation Manual	W414	Provides information on how to use the CX-Programmer for all functionality except for function blocks.

Manuals Related to the CS1-H (FB) and CJ1-H (FB) CPU Units

Name	Cat. No.	Contents
SYSMAC CS Series CS1G/H-CPU□□-EV1, CS1G/H-CPU□□H Programmable Controllers Operation Manual	W339	Provides an outline of and describes the design, installation, maintenance, and other basic operations for the CS-series PLCs. The following information is included: An overview and features The system configuration Installation and wiring I/O memory allocation Troubleshooting Use this manual together with the W394.
SYSMAC CJ Series CJ1G/H-CPU□□H, CJ1M-CPU□□, CJ1G-CPU□□ Programmable Controllers Operation Manual	W393	Provides an outline of and describes the design, installation, maintenance, and other basic operations for the CJ-series PLCs. The following information is included: An overview and features The system configuration Installation and wiring I/O memory allocation Troubleshooting Use this manual together with the W394.

Name	Cat. No.	Contents
SYSMAC CS/CJ Series CS1G/H-CPU□□-EV1, CS1G/H-CPU□□H, CJ1G/H-CPU□□H, CJ1M-CPU□□, CJ1G-CPU□□ Programmable Controllers Programming Manual	W394	Describes programming and other methods to use the functions of the CS/CJ-series PLCs. The following information is included: Programming Tasks File memory Other functions Use this manual together with the W339 or W393.
SYSMAC CS/CJ Series CS1G/H-CPU□□-EV1, CS1G/H-CPU□□H, CJ1G/H-CPU□□H, CJ1M-CPU□□, CJ1G-CPU□□ Programmable Controllers Instructions Reference Manual	W340	Describes the ladder diagram programming instructions supported by CS/CJ-series PLCs. When programming, use this manual together with the *Operation Manual* (CS1: W339 or CJ1: W393) and *Programming Manual* (W394).
SYSMAC CS/CJ Series CS1G/H-CPU□□-EV1, CS1G/H-CPU□□H, CS1W-SCB21-V1/41-V1, CS1W-SCU21/41, CJ1G/H-CPU□□H, CJ1M-CPU□□, CJ1G-CPU□□, CJ1W-SCU21/41 Communications Commands Reference Manual	W342	Describes the communications commands that can be addressed to CS/CJ-series CPU Units. The following information is included: C-series (Host Link) commands FINS commands Note: This manual describes commands that can be sent to the CPU Unit without regard for the communications path, which can be through a serial communications port on the CPU Unit, a communications port on a Serial Communications Unit/Board, or a port on any other Communications Unit.

Overview of Contents

Precautions provides general precautions for using the CX-Programmer IEC.

Section 1 provides an overview of CX-Programmer IEC functionality and general information on function blocks.

Section 2 provides information on and procedures for creating function blocks.

Section 3 provides technical specifications and restrictions for function blocks and information on the battery replacement procedure.

The ***Appendices*** provide additional information required for programming, including data types, ST language keywords, a table of external variables, and tables of instructions support and operand restrictions.

⚠ WARNING Failure to read and understand the information provided in this manual may result in personal injury or death, damage to the product, or product failure. Please read each section in its entirety and be sure you understand the information provided in the section and related sections before attempting any of the procedures or operations given.

PRECAUTIONS

This section provides general precautions for using the CX-Programmer IEC.

The information contained in this section is important for the safe and reliable application of the CX-Programmer IEC. You must read this section and understand the information contained before attempting to set up or operate the CX-Programmer IEC.

1 Intended Audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of installing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of managing FA systems and facilities.

2 General Precautions

The user must operate the product according to the performance specifications described in the operation manuals.

Before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems, machines, and equipment that may have a serious influence on lives and property if used improperly, consult your OMRON representative.

Make sure that the ratings and performance characteristics of the product are sufficient for the systems, machines, and equipment, and be sure to provide the systems, machines, and equipment with double safety mechanisms.

This manual provides information for programming and operating the product. Be sure to read this manual before attempting to use the product and keep this manual close at hand for reference during operation.

⚠ **WARNING** It is extremely important that a PLC and all PLC Units be used for the specified purpose and under the specified conditions, especially in applications that can directly or indirectly affect human life. You must consult with your OMRON representative before applying a PLC System to the above-mentioned applications.

3 Safety Precautions

⚠ **WARNING** Confirm safety sufficiently before transferring I/O memory area status from the CX-Programmer IEC to the CPU Unit. The devices connected to Output Units may malfunction, regardless of the operating mode of the CPU Unit. Caution is required in respect to the following functions.

- Transferring from the CX-Programmer IEC to real I/O (CIO Area) in the CPU Unit using the PLC Memory Window.
- Transferring from file memory to real I/O (CIO Area) in the CPU Unit using the Memory Card Window.

⚠ **Caution** Confirm safety at the destination node before transferring a program to another node or changing contents of the I/O memory area. Doing either of these without confirming safety may result in injury.

⚠ **Caution** Execute online editing only after confirming that no adverse effects will be caused by extending the cycle time. Otherwise, the input signals may not be readable.

⚠ **Caution** Confirm safety sufficiently before monitoring power flow and present value status in the Ladder Section Window or when monitoring present values in the Watch Window. If force-set/reset or set/reset operations are inadvertently performed by pressing short-cut keys, the devices connected to Output Units may malfunction, regardless of the operating mode of the CPU Unit.

4 Application Precautions

Observe the following precautions when using the CX-Programmer IEC.

- User programs cannot be uploaded to the CX-Programmer IEC.
- Observe the following precautions before starting the CX-Programmer IEC.
  - Exit all applications not directly related to the CX-Programmer IEC. Particularly exit any software such as screen savers, virus checkers, email or other communications software, and schedulers or other applications that start up periodically or automatically.
  - Disable sharing hard disks, printers, or other devices with other computers on any network.
  - With some notebook computers, the RS-232C port is allocated to a modem or an infrared line by default. Following the instructions in documentation for your computer and enable using the RS-232C port as a normal serial port.
  - With some notebook computers, the default settings for saving energy do not supply the rated power to the RS-232C port. There may be both Windows settings for saving energy, as well as setting for specific computer utilities and the BIOS. Following the instructions in documentation for your computer, disable all energy saving settings.
- Do not turn OFF the power supply to the PLC or disconnect the connecting cable while the CX-Programmer IEC is online with the PLC. The computer may malfunction.
- Confirm that no adverse effects will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.
  - Changing the operating mode of the PLC.
  - Force-setting/force-resetting any bit in memory.
  - Changing the present value of any word or any set value in memory.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- When online editing is performed, the user program and parameter area data in CS1-H (FB)/CJ1-H (FB) CPU Units is backed up in the built-in flash memory. The BKUP indicator will light on the front of the CPU Unit when the backup operation is in progress. Do not turn OFF the power supply to the CPU Unit when the BKUP indicator is lit. The data will not be backed up if power is turned OFF. To display the status of writing to flash memory on the CX-Programmer, select *Display dialog to show PLC Memory Backup Status* in the PLC properties and then select ***Windows*** – ***PLC Memory Backup Status*** from the View Menu.

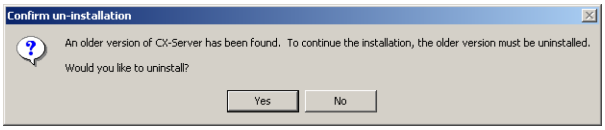
- If a project file created with the non-IEC CX-Programmer is read and the *Device Type* is changed to one that supports function blocks, the default function block memory allocations (function block instance area, refer to *2-2-6 Setting the FB Instance Areas*) will overlap with any of the following addresses used in the user program, causing errors when compiling:
  W000 to W511, EM 20480 to EM 32767 in the last EM bank, T1024 to T4095, and C1024 to C4095.
  If addresses are duplicated and an error occurs, either change the function block memory allocations or the addresses used in the user program.
- If a user program containing function blocks created on the CX-Programmer IEC is downloaded to a CPU Unit that does not support function blocks (e.g., the CS1-H or CJ1-H), all instances will be treated as illegal commands and it will not be possible to edit or execute the user program.
- The CX-Programmer IEC cannot be connected online to any CS-series or CJ-series CPU Unit not supported by it.
- CXP files from the non-IEC version of CX-Programmer for CPU Unit models not supported by the CX-Programmer IEC cannot be read by the CX-Programmer IEC.
- When specifying the first or last word of multiple words for an instruction operand, I/O parameters cannot be used to pass data to or from I/O variables. Internal array variables must be used. This applies, for example, to the first source word for SEND(090) or the starting word or end word for BSET(071).

  For multiword operands, an array variable must be prepared in advance with the required number of elements and the data must be set for the array in the function block definition. The first or last element in the array variable is then specified for the operand to set the first or last word. Refer to *3-4 Function Block Applications Guidelines* for details.
- Input values are passed from parameters to input variables before the algorithm is processed. Consequently, values cannot be read from parameters to input variables within the algorithm. If it is necessary to read a value within the execution cycle of the algorithm, do not pass the value from a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).

  In a similar fashion, output variables are passed to the corresponding parameters after the algorithm is processed. Consequently, values cannot be written from output variables to parameters within the algorithm. If it is necessary to write a value within the execution cycle of the algorithm, do not write the value to a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).
- Always use variables with AT settings in the following cases.
  - The first destination word at the remote node for SEND(090) and the first source word at the remote node for RECV(098)
  - Auxiliary Area flags and bits that are not registered for external variables and that need to be read or written within the execution cycle of an algorithm

5 Installation Precaution

If the non-IEC version of CX-Programmer is already installed when installing the CX-Programmer IEC, the following overwrite confirmation dialog box will be displayed.



Confirm un-installation

An older version of CX-Server has been found. To continue the installation, the older version must be uninstalled.

Would you like to uninstall?

Always click the **Yes** Button and install CX-Server version 2.00.

If the **No** Button is clicked, it will not be possible to use the CX-Programmer IEC (i.e., it will not be possible to select a *Device Type* that supports function blocks (FB)).

Even if the **Yes** Button is clicked, the non-IEC version of CX-Programmer will not be uninstalled and can be used as normal.

SECTION 1
Introduction

This section introduces the CX-Programmer IEC and explains the features that are not contained in the non-IEC version of CX-Programmer.

1-1 Introducing the CX-Programmer IEC

1-1-1 Functions and Features

The CX-Programmer IEC is a Programming Device that can use standard IEC 61131-3 function blocks. The CX-Programmer IEC is the same as non-IEC CX-Programmer version 3.0 except that function block functionality has been added. The CX-Programmer IEC is compatible with the CS/CJ-series PLCs and has the following features.

- Project files (.cxp) created with non-IEC CX-Programmer can be imported and reused. Function blocks can be created in ladder language by cutting and pasting program rungs.
- User-defined processes can be converted to block format by using function blocks.
- Function block algorithms can be written in the ladder programming language or in the structured text (ST) language. (See note.)
    - When ladder programming is used, ladder programs created with non-IEC CX-Programmer can be reused by copying and pasting.
    - When ST language is used, it is easy to program mathematical processes that would be difficult to enter with ladder programming.

  Note The ST language is an advanced language for industrial control (primarily PLCs) that is described in IEC 61131-3. The ST language supported by CX-Programmer IEC conforms to the IEC 61131-3 standard.
- Function blocks can be created easily because variables do not have to be declared in text. They are registered in variable tables.
  A variable can be registered automatically when it is entered in a ladder or ST program. Registered variables can also be entered in ladder programs after they have been registered in the variable table.
- A single function block can be converted to a library function as a single file, making it easy to reuse function blocks for standard processing.
- A program check can be performed on a single function block to easily confirm the function block’s reliability as a library function.
- One-dimensional variable arrays are supported, so data handling is easier for many applications.

Note The IEC 61131 standard was defined by the International Electrotechnical Commission (IEC) as an international programmable controller (PLC) standard. The standard is divided into 7 parts. Specifications related to PLC programming are defined in *Part 3 Textual Languages (IEC 61131-3).*

1-1-2 Specifications

Specifications that are not listed in the following table are identical to the specifications for CX-Programmer Version 3.0.

Item		Specifications
Model number		WS02-CPIC1-E
Setup disk		CD-ROM
Compatible CPU Units		Only the following CS1-H and CJ1-H CPU Units are compatible. No other CPU Units can be used. (See note.) • CS1G-CPU42H/44H (FB) • CS1H-CPU65H/67H (FB) • CJ1G-CPU42H/43H/44H (FB) **Note** Non-IEC CX-Programmer project files (.cxp) created for the following models can be read and reused by changing the *Device Type* to one that supports function blocks. Once the existing project file has been changed, CX-Programmer IEC function blocks can be used. • CS1G-CPU42H/43H/44H/45H • CS1H-CPU63H/64H/65H/66H/67H • CJ1G-CPU42H/43H/44H/45H • CJ1H-CPU65H/66H
		CS/CJ Series Function Restrictions • Program Restrictions Subroutine numbers 128 to 1023 cannot be used in Subroutine Instructions (SBS, GSBS, RET, MCRO, and SBN). Only numbers 0 to 127 can be used. • Instructions Not Supported in Function Block Definitions Block Program Instructions (BPRG and BEND), Subroutine Instructions (SBS, GSBS, RET, MCRO, and SBN), Jump Instructions (JMP, CJP, and CJPN), Step Ladder Instructions (STEP and SNXT), Immediate Refresh Instructions (!), I/O REFRESH (IORF), ONE-MS TIMER (TMHH), and HIGH-SPEED TIMER (TIMH) • Timer/Counter PV refreshing method: Binary only For details, refer to *3-3 Restrictions on Function Blocks*.
Compatible computers	Computer	IBM PC/AT or compatible
	CPU	133 MHz Pentium or faster with Windows 98, SE, or NT 4.0
	OS	Microsoft Windows 98, SE, Me, 2000, XP, or NT 4.0 (with service pack 6 or higher)
	Memory	64 Mbytes min. with Windows 98, SE, or NT 4.0 Refer to *Computer System Requirements* below for details.
	Hard disk space	100 Mbytes min. available disk space
	Monitor	SVGA (800 × 600 pixels) min. **Note** Use “small font” for the font size.
	CD-ROM drive	One CD-ROM drive min.
	COM port	One RS-232C port min.

Item			Specifications	
Functions not supported by non-IEC CX-Programmer	Defining and creating function blocks	Number of function block definitions	896 max. per CPU Unit	
		Function block names	64 characters max.	
		Variables	Variable names	30,000 characters max.
			Variable types	Inputs, Outputs, Internals, and Externals
			Number of I/O variables in function block definitions	64 max. (not including EN and ENO)
			Allocation of addresses used by variables	Automatic allocation (The allocation range can be set by the user.)
			Actual address specification	Supported
			Array specifications	Supported (one-dimensional arrays only)
		Language	Function blocks can be created in ladder programming language or structured text (ST, see note).	
	Creating instances	Number of instances	2,048 max. per CPU Unit	
		Instance names	30,000 characters max.	
	Storing function blocks as library files		Each function block definition can be stored as one file for reuse in other projects.	

Note The ST language conforms to the IEC 61131-3 standard, but CX-Programmer IEC supports only assignment statements, selection statements (CASE and IF statements), iteration statements (FOR, WHILE, and REPEAT statements), arithmetic operators, logical operators, comparison operators, and comments. Other statements and operators are not supported. For details, refer to *Appendix B Structured Text Keywords.*

Restrictions on Particular CPU Units

- If a user program created with CX-Programmer IEC contains function blocks, it cannot be downloaded to a CPU Unit that does not support function blocks. If the program is downloaded to a CPU Unit that does not support function blocks, all function block instances will be treated as illegal instructions and it will not be possible to edit or execute the user program.
- The CX-Programmer IEC cannot be placed online with a CPU Unit that does not support function blocks.
- The CX-Programmer IEC cannot read non-IEC CX-Programmer CXP files for CPU Units it does not support.

Computer System Requirements

Item		OS		
		Windows 95 (See note 2.), 98, or NT 4.0 Service Pack 6	**Windows 2000 or Me**	**Windows XP**
Computer		IBM PC/AT or compatible	IBM PC/AT or compatible	IBM PC/AT or compatible
CPU		Pentium class 133 MHz or faster	Pentium class 150 MHz or faster	Pentium class 300 MHz or faster
Memory (RAM) capacity	Programs up to 30 Ksteps	64 Mbytes min. (96 Mbytes min. when also using CX-Simulator)	96 Mbytes min. (128 Mbytes min. when also using CX-Simulator)	128 Mbytes min. (192 Mbytes min. when also using CX-Simulator)
	For programs up to 120 Ksteps	128 Mbytes min. (128 Mbytes min. when also using CX-Simulator)	192 Mbytes min. (192 Mbytes min. when also using CX-Simulator)	256 Mbytes min. (256 Mbytes min. when also using CX-Simulator)
	For programs over 120 Ksteps	192 Mbytes min. (192 Mbytes min. when also using CX-Simulator)	256 Mbytes min. (256 Mbytes min. when also using CX-Simulator)	384 Mbytes min. (384 Mbytes min. when also using CX-Simulator)
Hard disk space		100 Mbytes min. available	100 Mbytes min. available	100 Mbytes min. available
Display		800 × 600 SVGA min.	800 × 600 SVGA min.	800 × 600 SVGA min.
CD-ROM drive		One CD-ROM drive min.	One CD-ROM drive min.	One CD-ROM drive min.
COM port		One RS-232C port min.		

Note (1) The required memory (RAM) capacity is the capacity required to create programs. If the computer's memory is less than the required memory capacity, the CX-Programmer may operate slowly.

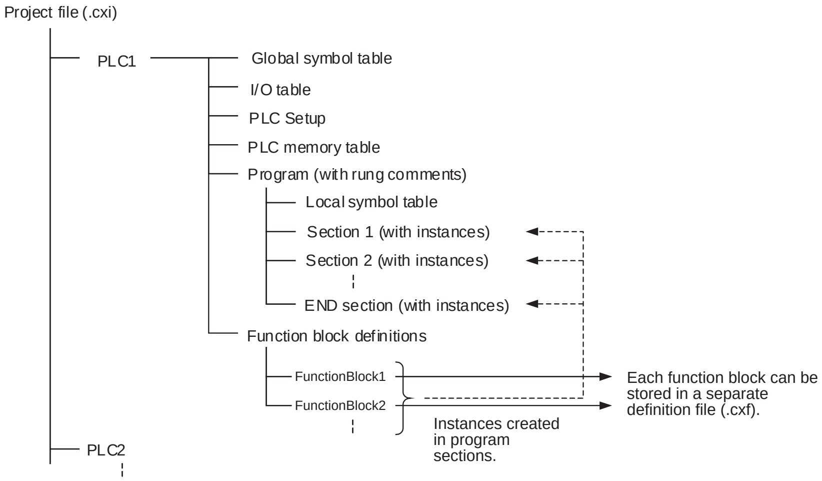
(2) Windows 95 cannot be used when connecting through a Controller Link Support Board (PCI Card) or SYSMAC LINK Support Board (PCI Card).

1-1-3 Files Created with CX-Programmer IEC

Project Files (*.cxi)

Projects created in CX-Programmer IEC contain all of the program-related data, such as function block definitions and programs with instances. The data is stored as a file with a "cxi" filename extension.

The following diagram shows the contents of a project. The function block definitions are created at the same directory level as the program within the relevant PLC directory.


Project file (.cxi)
PLC1
Global symbol table
I/O table
PLC Setup
PLC memory table
Program (with rung comments)
Local symbol table
Section 1 (with instances)
Section 2 (with instances)
END section (with instances)
Function block definitions
FunctionBlock1
FunctionBlock2
Instances created in program sections.
Each function block can be stored in a separate definition file (.cxf).
PLC2

Note Project files created with non-IEC CX-Programmer (*.cxp) can be read (imported) but cannot be saved. After importing a file, the CX-Programmer IEC functions can be used if the *Device Type* is changed to one that supports function blocks. Once the *Device Type* has been changed, existing program rungs can be copied and pasted, function blocks can be created in the ladder programming language, and the data can be saved as a CX-Programmer IEC project file (*.cxi).

Function Block/Library Files (.cxf)

A function block definition created in a project in CX-Programmer IEC can be saved as a file (1 definition = 1 file) so that definitions can be read into other programs and reused.

Project Text Files in CX-Programmer IEC (*.cxt)

The project files created in CX-Programmer IEC (*.cxi) can be saved as CXT text files (*.cxt) just as in the non-IEC CX-Programmer.

1-1-4 CX-Programmer IEC Menus

The following tables list CX-Programmer IEC menus that are different from non-IEC CX-Programmer menus. Menus that are the same are not listed.

Main Menu

Main menu	Submenu		Shortcut	Function
Insert	Function Block Invocation		F	Creates an instance of a function block in the program at the present cursor location.
	Function Block Parameter		P	When the cursor is located to the left of an input variable or the right of an output variable, sets the variable's input or output parameter.
PLC	Memory	Function Block Memory Allocation	---	Sets the range of addresses (function block instance areas) internally allocated to the selected instance's variables.
		Function Block Memory Statistics	---	Checks the status of the addresses internally allocated to the selected instance's variables.
		Function Block Memory Address	---	Checks the addresses internally allocated to each variable in the selected instance.
		Optimize Function Memory	---	Optimizes the allocation of addresses internally allocated to variables.

Main Popup Menus

Popup Menu for Function Block Definitions

Popup menu		Function
Insert Function Block	Ladder	Creates a function block definition with a ladder programming language algorithm.
	Structured Text	Creates a function block definition with an ST language algorithm.
	From file	Reads a function block definition from a function block library file (*.cxf).

Popup Menu for Inserted Function Blocks

Popup menu	Function
Open	Displays the contents of the selected function block definition on the right side of the window.
Save Function Block File	Saves the selected function block definition in a file.
Compile	Compiles the selected function block definition.

Popup Menu for Instances

Popup menu		Function
Edit		Changes the instance name.
Update Invocation		When a function block definition's I/O variables have been changed after the instance was created, an error will be indicated by displaying the instance's left bus bar in red. This command updates the instance with the new information and clears the error.
Go To	Function Block Definition	Displays the selected instance's function block definition on the right side of the window.

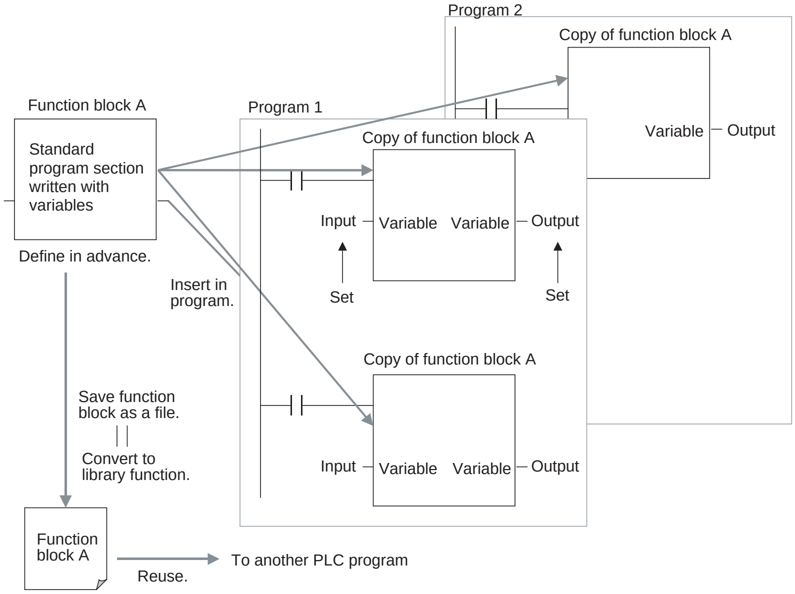
1-2 Function Blocks

1-2-1 Outline

A function block is a basic program element containing a standard processing function that has been defined in advance. Once the function block has been defined, the user just has to insert the function block in the program and set the I/O in order to use the function.

As a standard processing function, a function block does not contain actual addresses, but variables. The user sets addresses or constants in those variables. These address or constants are called parameters. The addresses used by the variables themselves are allocated automatically by the CX-Programmer IEC for each program.

With the CX-Programmer IEC, a single function block can be saved as a single file and reused in other PLC programs, so standard processing functions can be made into libraries.



1-2-2 Advantages of Function Blocks

Function blocks allow complex programming units to be reused easily. Once standard programming is created in a function block and saved in a file, it can be reused just by placing the function block in a program and setting the parameters for the function block's I/O. The ability to reuse existing function blocks will save significant time when creating/debugging programs, reduce coding errors, and make the program easier to understand.

Structured Programming

Structured programs created with function blocks have better design quality and require less development time.

Easy-to-read "Black Box" Design

The I/O operands are displayed as variable names in the program, so the program is like a "black box" when entering or reading the program and no extra time is wasted trying to understand the internal algorithm.

Use One Function Block for Multiple Processes

Many different processes can be created easily from a single function block by using the parameters in the standard process as input variables (such as timer SVs, control constants, speed settings, and travel distances).

Reduce Coding Errors

Coding mistakes can be reduced because blocks that have already been debugged can be reused.

Data Protection

The variables in the function block cannot be accessed directly from the outside, so the data can be protected. (Data cannot be changed unintentionally.)

Improved Reusability with Variable Programming

The function block's I/O is entered as variables, so it isn't necessary to change data addresses in a block when reusing it.

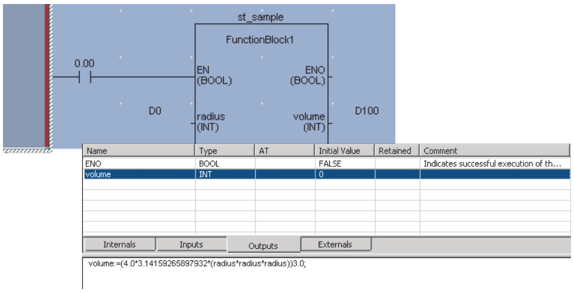
Creating Libraries

Processes that are independent and reusable (such as processes for individual steps, machinery, equipment, or control systems) can be saved as function block definitions and converted to library functions.

The function blocks are created with variable names that are not tied to actual addresses, so new programs can be developed easily just by reading the definitions from the file and placing them in a new program.

Compatible with Multiple Languages

Mathematical expressions can be entered in structured text (ST) language.



Name	Type	AT	Initial Value	Retained	Comment
ENO	BOOL		FALSE		Indicates successful execution of th...
volume	INT		0		

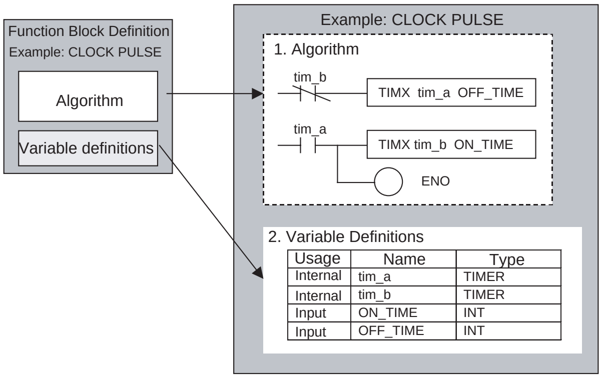
```
volume:=(4.0*3.14159265897932*(radius*radius*radius))3.0;
```

1-2-3 Function Block Structure

A function block consists of the function block definition that is created in advance and the function block instances that are inserted in the program.

Function Block Definition

The function block definition is the basic element that makes the function block reusable. Each function block definition contains the algorithm and variable definitions, as shown in the following diagram.



Usage	Name	Type
Internal	tim_a	TIMER
Internal	tim_b	TIMER
Input	ON_TIME	INT
Input	OFF_TIME	INT

1. Algorithm

Standardized programming is written with variable names rather than actual I/O memory addresses. In the CX-Programmer IEC, algorithms can be written in either ladder programming or structured text.

2. Variable Definitions

The variable table lists each variable's usage (input, output, or internal) and properties (data type, etc.). For details, refer to *1-3 Variables*.

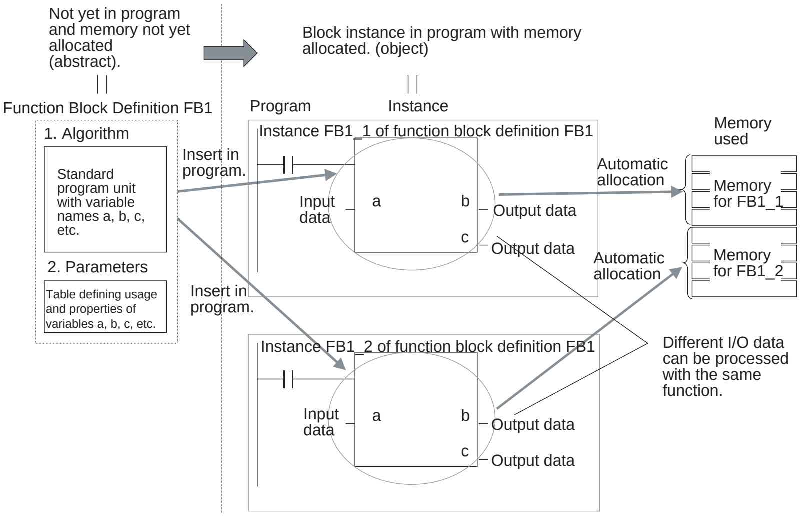
Number of Function Block Definitions

Up to 896 function block definitions can be created for one CPU Unit.

Instances

When a function block definition is inserted in a program, the function block uses a particular memory area for its variables. Each function block definition that is inserted in the program is called an “instance” or “function block instance.” Each instance is assigned an identifier called an “instance name.”

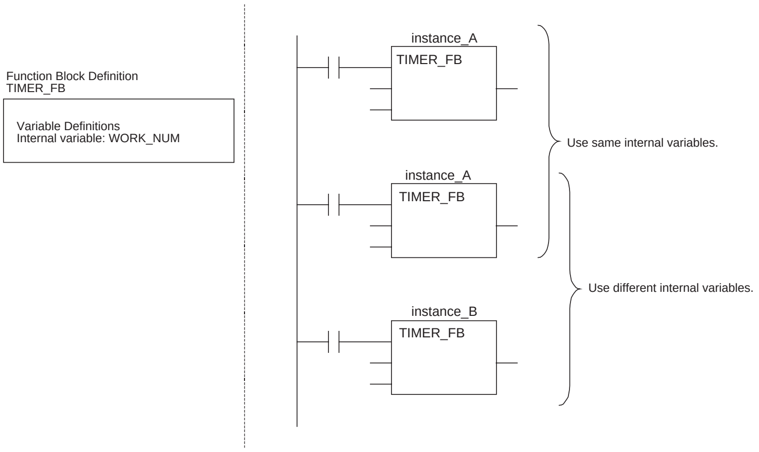
By generating instances, a single function block definition can be used to process different I/O data with the same function.


Not yet in program and memory not yet allocated (abstract).
Block instance in program with memory allocated. (object)
Function Block Definition FB1
Program
Instance
1. Algorithm
Standard program unit with variable names a, b, c, etc.
2. Parameters
Table defining usage and properties of variables a, b, c, etc.
Insert in program.
Insert in program.
Instance FB1_1 of function block definition FB1
Instance FB1_2 of function block definition FB1
Input data
a
b
c
Output data
Output data
Memory used
Automatic allocation
Memory for FB1_1
Automatic allocation
Memory for FB1_2
Different I/O data can be processed with the same function.

Note Instances are managed by names. More than one instance with the same name can also be inserted in the program. If two or more instances have the same name, they will use the same internal variables. Instances with different names will have different internal variables.

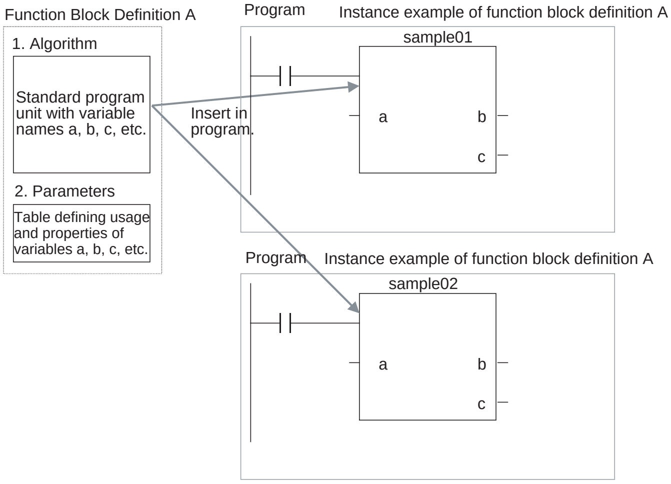
For example, consider three function blocks that use a timer as an internal variable. In this case all instances will have to be given different names. If more than one instance uses the same name, the use of the timer would be duplicated, which is not allowed.

If, however, internal variables are not used or they are used only temporarily and initialized the next time an instance is executed, the same instance name can be used to save memory.


Function Block Definition TIMER_FB
Variable Definitions
Internal variable: WORK_NUM
instance_A
TIMER_FB
instance_A
TIMER_FB
instance_B
TIMER_FB
Use same internal variables.
Use different internal variables.

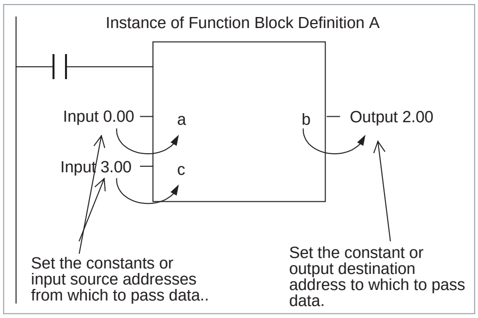
Number of Instances

Multiple instances can be created from a single function block definition. Up to 2,048 instances can be created for a single CPU Unit. The allowed number of instances is not related to the number of function block definitions or the number of tasks in which the instances are inserted.


Function Block Definition A
1. Algorithm
Standard program unit with variable names a, b, c, etc.
2. Parameters
Table defining usage and properties of variables a, b, c, etc.
Insert in program.
Program
Instance example of function block definition A
sample01
a
b
c
Program
Instance example of function block definition A
sample02
a
b
c

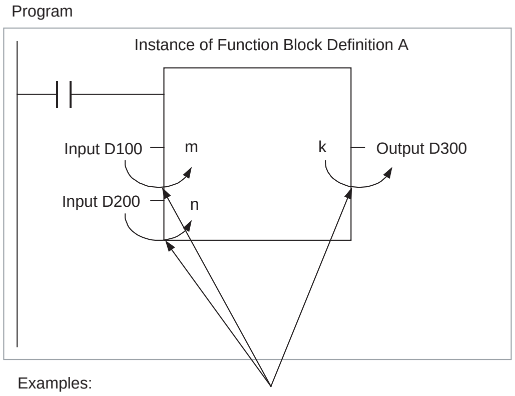
Parameters

Each time an instance is created, the actual I/O memory addresses or constants used to pass data to and from the I/O variables are set. These addresses or constants are called parameters.


Instance of Function Block Definition A
Input 0.00
a
b
Output 2.00
Input 3.00
c
Set the constants or input source addresses from which to pass data..
Set the constant or output destination address to which to pass data.

Here, it is not the input source address itself, but the contents at the input address in the form and size specified by the variable data type that is passed to the function block. In a similar fashion, it is not the output destination address itself, but the contents for the output address in the form and size specified by the variable data type that is passed from the function block.

Even if an input source address (i.e., an input parameter) or an output destination address (i.e., an output parameter) is a word address, the data that is passed will be the data in the form and size specified by the variable data type starting from the specified word address.



Examples:

If m is type WORD, one word of data from D100 will be passed to the variable.

If n is type DWORD, two words of data from D200 and D201 will be passed to the variable.

If k is type LWORD, four words of data from the variable will be passed to the D300 to D303.

Note

(1) Only addresses in the following areas can be used as parameters: CIO Area, Auxiliary Area, DM Area, EM Area (banks 0 to C), Holding Area, and Work Area.
The following cannot be used: Index and data registers (both direct and indirect specifications) and indirect addresses to the DM Area and EM Area (both in binary and BCD mode).

(2) Local and global symbols in the user program can also be specified as parameters. To do so, however, the data size of the local or global symbol must be the same as the data size of the function block variable.

(3) When an instance is executed, input values are passed from parameters to input variables before the algorithm is processed. Output values are passed from output variables to parameters just after processing the algorithm. If it is necessary to read or write a value within the execution cycle of the algorithm, do not pass the value to or from a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).

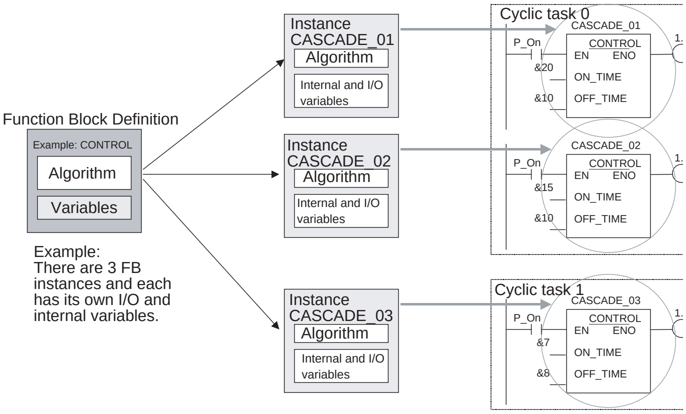
⚠ **Caution** When specifying the first or last word of multiple words for an instruction operand, I/O parameters cannot be used to pass data to or from I/O variables. Internal array variables must be used. This applies, for example, to the first source word for SEND(090) or the starting word and end word for BSET(071). For multiword operands, an array variable must be prepared in advance with the required number of elements and the data must be set for the array in the function block definition. The first or last element in the array variable is then specified for the operand to set the first or last word. Refer to *3-4 Function Block Applications Guidelines* for details.

⚠ **Caution** Input values are passed from parameters to input variables before the algorithm is processed. Consequently, values cannot be read from parameters to input variables within the algorithm. If it is necessary to read a value within the execution cycle of the algorithm, do not pass the value from a parameter. Assign the value to an internal variable and use an AT setting (specified addresses). In a similar fashion, output variables are passed to the corresponding parameters after the algorithm is processed. Consequently, values cannot be written from output variables to parameters within the algorithm. If it is necessary to write a value within the execution cycle of the algorithm, do not write the value to a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).

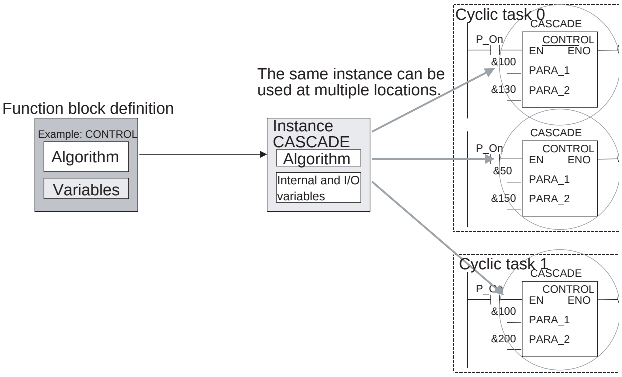
■ Reference Information

A variety of processes can be created easily from a single function block by using parameter-like elements (such as fixed values) as input variables and changing the values passed to the input variables for each instance.

Example: Creating 3 Instances from 1 Function Block Definition



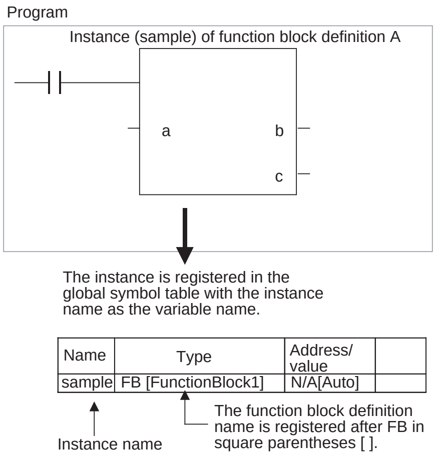
If internal variables are not used, if processing will not be affected, or if the internal variables are used in other locations, the same instance name can be used at multiple locations in the program.



Some precautions are required when using the same memory area. For example, if an instance containing a timer instruction is used in more than one program location, the same timer number will be used causing coil duplication, and the timer will not function properly if both instructions are executed.

Registration of Instances

Each instance name is registered in the global symbol table as a file name.


Program
Instance (sample) of function block definition A
a
b
c
The instance is registered in the global symbol table with the instance name as the variable name.
Instance name
The function block definition name is registered after FB in square parentheses [ ].

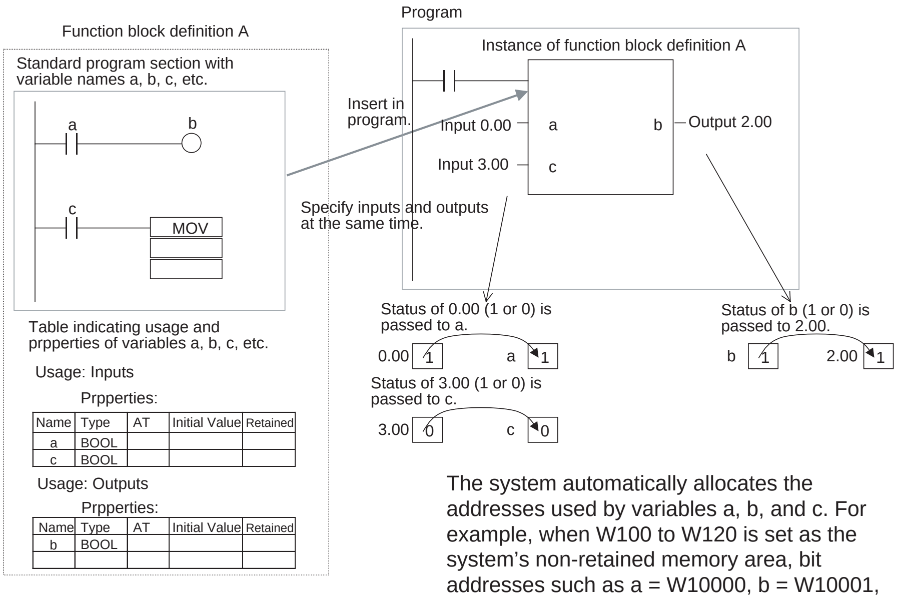
Name	Type	Address/ value	
sample	FB [FunctionBlock1]	N/A[Auto]	

1-3 Variables

1-3-1 Introduction

In a function block, the addresses are not entered as actual I/O memory addresses, they are all entered as variable names. Each time an instance is created, the actual addresses used by the variable are allocated automatically in the specified I/O memory areas by the CX-Programmer IEC. Consequently, it isn't necessary for the user to know the actual I/O memory addresses used in the function block, just as it isn't necessary to know the actual memory allocations in a computer. A function block differs from a subroutine in this respect, i.e., the function block uses variables and the addresses are like "black boxes."

Example:


Program
Function block definition A
Instance of function block definition A
Standard program section with variable names a, b, c, etc.
a
b
c
MOV
Insert in program.
Input 0.00
a
b
Output 2.00
Input 3.00
c
Specify inputs and outputs at the same time.
Status of 0.00 (1 or 0) is passed to a.
0.00
1
a
1
Status of 3.00 (1 or 0) is passed to c.
3.00
0
c
0
Status of b (1 or 0) is passed to 2.00.
b
1
2.00
1
Table indicating usage and prpperties of variables a, b, c, etc.
Usage: Inputs
Prpperties:
Usage: Outputs
Prpperties:

Name	Type	AT	Initial Value	Retained
a	BOOL			
c	BOOL			

Name	Type	AT	Initial Value	Retained
b	BOOL			

The system automatically allocates the addresses used by variables a, b, and c. For example, when W100 to W120 is set as the system's non-retained memory area, bit addresses such as a = W10000, b = W10001, and c = W10002 will be allocated.

1-3-2 Variable Usage and Properties

Variable Usage

The following variable types (usages) are supported.

Internals: Internal variables are used only within an instance. They cannot be used pass data directly to or from I/O parameters.

Inputs: Input variables can input data from input parameters outside of the instance. The default input variable is an EN (Enable) variable, which passes input condition data.

Outputs: Output variables can output data to output parameters outside of the instance. The default output variable is an ENO (Enable Out) variable, which passes the instance's execution status.

Externals: External variables are global symbols registered in advance as variables in the CX-Programmer IEC, such as Condition Flags and some Auxiliary Area bits.

The following table shows the number of variables that can be used and the kind of variable that is created by default for each of the variable usages.

Variable usage	Allowed number	Variable created by default
Inputs	Up to 64 per function block (not including EN)	EN (Enable): Receives an input condition. The instance is executed when the variable is ON. The instance is not executed when the variable is OFF.
Outputs	Up to 64 per function block (not including ENO)	EN (Enable Output): Outputs the function block's execution status. The variable is turned ON when the instance starts being executed. It can be turned OFF by the algorithm. The variable remains OFF when the instance is not executed.
Internals	Unlimited	None
Externals	Reserved variables only (28)	Global symbols registered in advance as variables in the CX-Programmer IEC, such as Conditions Flags or some Auxiliary Area bits. For details, refer to *Appendix C External Variables*.

1-3-3 Variable Properties

Variables have the following properties.

Variable Name

The variable name is used to identify the variable in the function block. It doesn’t matter if the same name is used in other function blocks.

Note The variable name can be up to 30,000 characters long, but must not begin with a number. Also, the name cannot contain two underscore characters in a row. There are no other restrictions. (Consequently, it is acceptable to use addresses such as “A20300” as variable names.)

Data Type

Select one of the following data types for the variable. Any of the following types may be used.

Data type	Content	Size	Inputs	Outputs	Internals
BOOL	Bit data	1 bit	OK	OK	OK
INT	Integer	16 bits	OK	OK	OK
UNIT	Unsigned integer	16 bits	OK	OK	OK
DINT	Double integer	32 bits	OK	OK	OK
UDINT	Unsigned double integer	32 bits	OK	OK	OK
LINT	Long (8-byte) integer	64 bits	OK	OK	OK
ULINT	Unsigned long (8-byte) integer	64 bits	OK	OK	OK
WORD	16-bit data	16 bits	OK	OK	OK
DWORD	32-bit data	32 bits	OK	OK	OK
LWORD	64-bit data	64 bits	OK	OK	OK
REAL	Real number	32 bits	OK	OK	OK
LREAL	Long real number	64 bits	OK	OK	OK
TIMER	Timer (See note 1.)	1 bit or 16 bits	OK	OK	OK
COUNTER	Counter (See note 2.)	1 bit or 16 bits	OK	OK	OK

Note

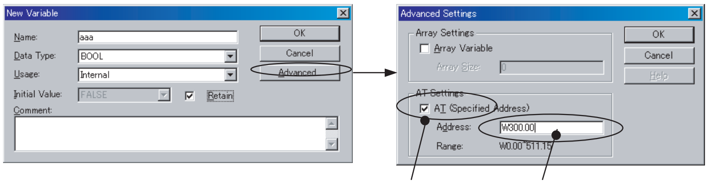
(1) When a variable is entered in the timer number (0 to 4095) operand of a timer instruction, such as TIM or TIMH, the data type will be TIMER. When this variable is used as an operand in another instruction, it will be treated as the timer Completion Flag if the operand takes 1-bit data or as a timer PV if the operand takes 16-bit data. The timer PVs are 16-bit binary data because the CX-Programmer IEC can use only binary format for the PVs. The TIMER data type cannot be used in ST language function blocks.

(2) When a variable is entered in the counter number (0 to 4095) operand of a counter instruction, such as CNT or CNTR, the data type will be COUNTER. When this variable is used as an operand in another instruction, it will be treated as a counter Completion Flag if the operand takes 1-bit data or as a counter PV if the operand takes 16-bit data. The counter PVs are 16-bit binary data because the CX-Programmer IEC can use only binary format for the PVs.
The COUNTER data type cannot be used in ST language function blocks.

AT Settings (Allocation to an Actual Addresses)

It is possible to set a variable to a particular I/O memory address rather than having it allocated automatically by the system. To specify a particular address, the user can input the desired I/O memory address in this property. This property can be set for internal variables only. Even if a specific address is set, the variable name must still be used in the algorithm.

- Setting Procedure
  Click the **Advanced** Button, select the *AT (Specified Address)* option, and input the desired address in the *Address* field.



New Variable

Name: aaa

Data Type: BOOL

Usage: Internal

Initial Value: FALSE ☑ Retain

Comment:

Advanced Settings

Array Settings

☐ Array Variable

Array Size: 0

AT Settings

☑ AT (Specified Address)

Address: W300.00

Range: W0.00~511.15

Select the AT option. Input the address.

- Even though a specified address is being used for the variable, specify the variable name in the algorithm in the function block definition. (Specify a variable name regardless of whether an address is being specified for the variable.)

Note

(1) Only addresses in the following areas can be used for AT settings: CIO Area, Auxiliary Area, DM Area, EM Area (banks 0 to C), Holding Area, and Work Area. The following cannot be used: Index and data registers (both direct and indirect specifications) and indirect addresses to the DM Area and EM Area (both in binary and BCD mode).

(2) Always use variables with AT settings in the following cases.

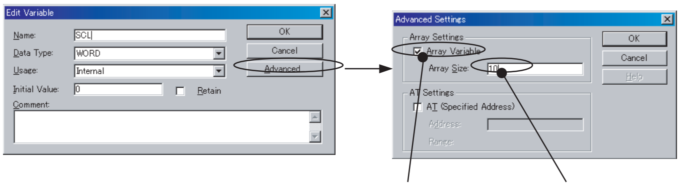
- The first destination word at the remote node for SEND(090) and the first source word at the remote node for RECV(098)
- Auxiliary Area flags and bits that are not registered for external variables and that need to be read or written within the execution cycle of an algorithm (Auxiliary Area flags and bits can be used as parameters to pass data when these conditions do not apply.)

Array Settings

A variable can be treated as a single array of data with the same properties. To convert a variable to an array, specify that it is an array and specify the maximum number of elements.

This property can be set for internal variables only. Only one-dimensional arrays are supported by the CX-Programmer IEC.

- Setting Procedure
  Click the **Advanced** Button, select the *Array Variable* option, and input the maximum number of elements in the *Size* field.



Edit Variable

Name: SCL

Data Type: WORD

Usage: Internal

Initial Value: 0 ☐ Retain

Comment:

Advanced Settings

Array Settings

☑ Array Variable

Array Size: 10

AT Settings

☐ AT (Specified Address)

Address:

Range:

Select the Array Variable option. Input the maximum number of elements.

- When entering an array variable name in the algorithm in a function block definition, enter the array index number in square brackets after the variable number.

For details on array settings, refer to *Variable Definitions* in *3-1-2 Function Block Elements*.

■ Reference Information

When specifying the first or last word of multiple words for an instruction operand, I/O parameters cannot be used to pass data to or from I/O variables. Internal array variables must be used. For multiword operands, an array variable must be prepared in advance with the required number of elements and the data must be set for the array in the function block definition. The first or last element in the array variable is then specified for the operand to set the first or last word. Refer to *3-4 Function Block Applications Guidelines* for details. Refer to *Appendix D Instruction Support and Operand Restrictions* for the instructions and operands that require designation of a first or last word address for a multiword operand.

Initial Value

This is the initial value set in a variable before the instance is executed for the first time. Afterwards, the value may be changed as the instance is executed.

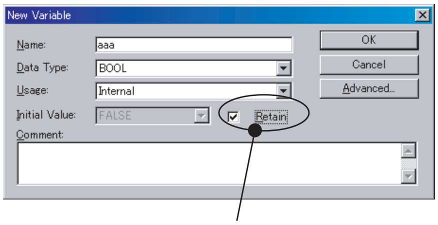
For example, set a boolean variable (bit) to either 1 (TRUE) or 0 (FALSE). Set a WORD variable to a value between 0 and 65,535 (between 0000 and FFFF hex).

If an initial value is not set, the variable will be set to 0. For example, a boolean variable would be 0 (FALSE) and a WORD variable would be 0000 hex.

Retain

Select the *Retain Option* if you want an internal variable's data to be retained when the PLC is turned ON again and when the PLC starts operating.

- Setting Procedure
  Select the *Retain Option*.



New Variable

Name: aaa

Data Type: BOOL

Usage: Internal

Initial Value: FALSE

☑ Retain

Comment:

Select the Retain option.

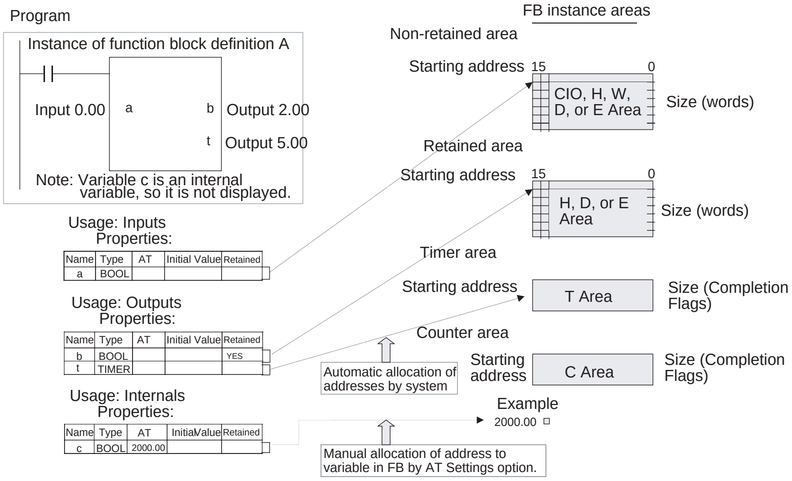
1-3-4 Property Settings and Variable Usage

The following table shows which properties must be set, can be set, and cannot be set, based on the variable usage.

Property	Variable usage		
	Internals	Inputs	Outputs
Name	Must be set.	Must be set.	Must be set.
Type	Must be set.	Must be set.	Must be set.
AT (specified address)	Can be set.	Cannot be set.	Cannot be set.
Initial Value	Can be set.	Can be set.	Can be set.
Retain	Can be set.	Cannot be set.	Cannot be set.

1-3-5 Internal Allocation of Variable Addresses

When an instance is created from a function block definition, the CX-Programmer IEC internally allocates addresses to the variables. Addresses are allocated to all of the variables registered in the function block definition except for variables that have been assigned actual addresses with the *AT Settings* property.


Program
Instance of function block definition A
Input 0.00
a
b
t
Output 2.00
Output 5.00
Note: Variable c is an internal variable, so it is not displayed.
Usage: Inputs Properties:
Name Type AT Initial Value Retained
a BOOL
Usage: Outputs Properties:
Name Type AT Initial Value Retained
b BOOL YES
t TIMER
Usage: Internals Properties:
Name Type AT InitialValue Retained
c BOOL 2000.00
FB instance areas
Non-retained area
Starting address
15
0
CIO, H, W, D, or E Area
Size (words)
Retained area
Starting address
15
0
H, D, or E Area
Size (words)
Timer area
Starting address
T Area
Size (Completion Flags)
Counter area
Starting address
C Area
Size (Completion Flags)
Automatic allocation of addresses by system
Example
2000.00
Manual allocation of address to variable in FB by AT Settings option.

Setting Internal Allocation Areas for Variables

The user sets the function block instance areas in which addresses are allocated internally by the CX-Programmer IEC. The variables are allocated automatically by the system to the appropriate instance area set by the user. The following data areas can be set for the instance areas.

Non-retained Area

- Applicable variables: Internal variables that do not have the *Retain Option* selected to retain the variable’s content when the power is turned ON or program execution starts.

  Note TIMER and COUNTER data types are not allocated to the non-retained area.
- Allowed data areas: I/O (CIO Area), H (Holding Area), W (Work Area), D (DM Area), or E (EM Area)

  Note Bit data can be accessed even if the DM or EM Area is specified.
- Units: Set in word units.
- Default allocation: W000 to W511

Retained Area

- Applicable variables: Internal variables that have the *Retain Option* selected to retain the variable’s content when the power is turned ON or program execution starts.

  Note TIMER and COUNTER data types are not allocated to the retained area.
- Allowed data areas: H (Holding Area), D (DM Area), or E (EM Area)

  Note Bit data can be accessed even if the DM or EM Area is specified.
- Units: Set in word units.
- Default allocation: Words 20480 to 32767 of the last EM bank

  Note The default area is words 20480 to 32767 of the last EM bank. The last EM bank number depends on the CPU Unit being used.

Timer Area

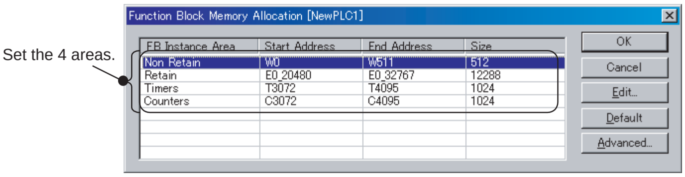
- Applicable variables: Variables that have the data type property set to TIMER.
- Allowed data areas: Timer Completion Flags (1 bit each) or timer PVs (16 bits each)
- Default allocation: T3072 to T4095 timer Completion Flags (1 bit each) or timer PVs (16 bits each)

Counter Area

- Applicable variables: Variables that have the data type property set to COUNTER.
- Allowed data areas: Counter completion flags (1 bit each) or counter PVs (16 bits each)
- Default allocation: C3072 to C4095 counter Completion Flags (1 bit each) or counter PVs (16 bits each)

Setting Procedure

Select ***Memory - Function Block Memory Allocation*** from the *PLC* Menu. Set the areas in the following dialog box.



Set the 4 areas.

FB Instance Area	Start Address	End Address	Size
Non Retain	W0	W511	512
Retain	E0_20480	E0_32767	12288
Timers	T3072	T4095	1024
Counters	C3072	C4095	1024

Setting Example:

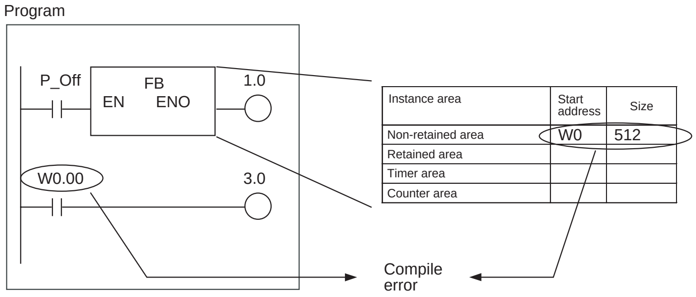
Instance area	Start Address	End Address	Size
Non Retain	W400	W449	50
Retain	E0_20480	E0_32767	12288
Timers	T3072	T4095	1024
Counters	C3072	C4095	1024

Specifying Instance Area Addresses from the User Program

If there are instructions in the user program that access addresses in the instance areas, the CX-Programmer IEC will display an error on the Output Window's *Compile (Program Check)* Tab Page in the following cases:

- When attempting to download the user program to the CPU Unit or attempting to write the program through online editing. (Neither downloading or editing will be possible.)
- When a program check is performed by the user by selecting ***Program - Compile (Program Check)*** or ***Compile All Programs (Check)*** from the *PLC* Menu.

For example, if W000 to W511 is specified as the non-retained instance area and W000 is used in the ladder program, the following error will be displayed when compiling: ERROR: ... (*omitted*) ... Address - W0.00 is reserved for Function Block use.

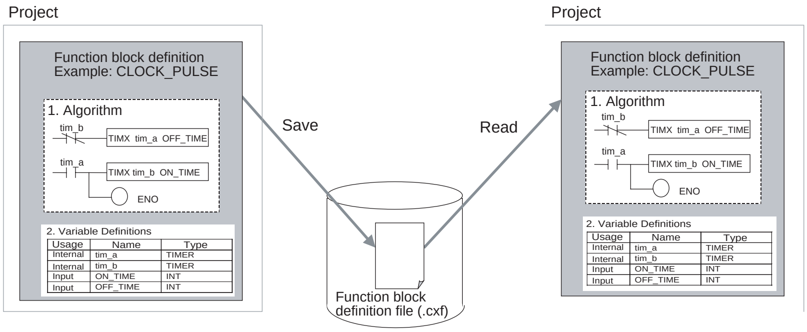


Instance area	Start address	Size
Non-retained area	W0	512
Retained area		
Timer area		
Counter area		

Note When a variable is added or deleted, addresses are automatically re-allocated to the instance areas. Consecutive addresses are required for each instance, so all of the variables will be allocated to a different block of addresses if the original block of addresses cannot accommodate the change in variables. This will result in an unused block of addresses. A memory optimization function can be executed to eliminate the unused area of memory so that the memory is used more efficiently.

1-4 Converting Function Block Definitions to Library Files

A function block definition created in the CX-Programmer IEC can be stored as a single file known as a function block definition file with filename extension.cxf. These files can be reused in other projects (PLCs).



Usage	Name	Type
Internal	tim_a	TIMER
Internal	tim_b	TIMER
Input	ON_TIME	INT
Input	OFF_TIME	INT

Usage	Name	Type
Internal	tim_a	TIMER
Internal	tim_b	TIMER
Input	ON_TIME	INT
Input	OFF_TIME	INT

1-5 Operating Procedures

Once a function block definition has been created and an instance of the algorithm has been created, the instance is used by calling it when it is time to execute it. Also, the function block definition that was created can be saved in a file so that it can be reused in other projects (PLCs).

1-5-1 Creating Function Blocks and Executing Instances

The following procedure outlines the steps required to create and execute a function block.

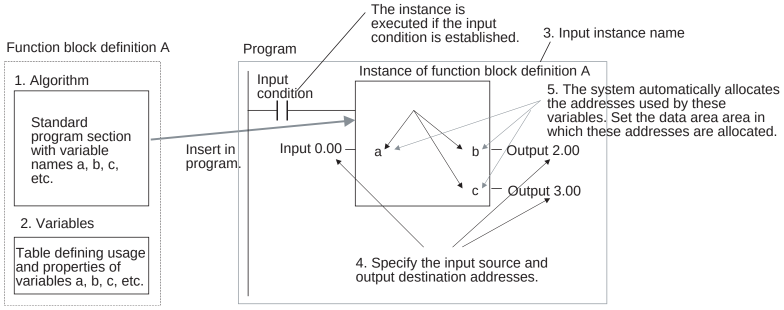
1,2,3...

1. First, create the function block definition including the algorithm and variable definitions in ladder program or ST language.

   Note (a) Create the algorithm entirely with variable names.

   (b) When entering the algorithm in ladder programming language, project files created with Non-IEC CX-Programmer can be reused by reading the project file into the CX-Programmer IEC and copying and pasting useful parts.

2. When creating the program, insert copies of the completed function block definition. This step creates instances of the function block.
3. Enter an instance name for each instance.
4. Set the variables' input source addresses and/or constants and output destination addresses and/or constants as the parameters to pass data for each instance.
5. Select the created instance, select ***Memory - Function Block Memory Allocation*** from the *PLC* Menu, and set the internal data area for each type of variable.
6. Transfer the program to the CPU Unit.
7. Start program execution in the CPU Unit and the instance will be called and executed if their input conditions are ON.



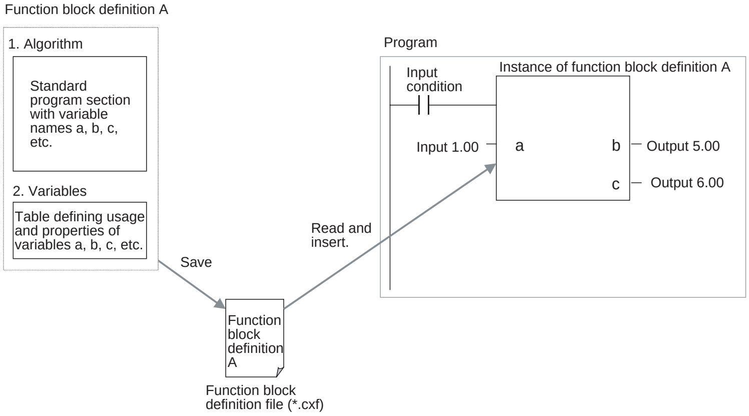
The instance is executed if the input condition is established.
3. Input instance name
Function block definition A
Program
Instance of function block definition A
1. Algorithm
Input condition
5. The system automatically allocates the addresses used by these variables. Set the data area area in which these addresses are allocated.
Standard program section with variable names a, b, c, etc.
Insert in program.
Input 0.00
a
b
Output 2.00
c
Output 3.00
2. Variables
Table defining usage and properties of variables a, b, c, etc.
4. Specify the input source and output destination addresses.

1-5-2 Reusing Function Blocks

Use the following procedure to save a function block definition as a file and use it in a program for another PLCs.

1,2,3...

1. Select the function block that you want to save and save it as a function block definition file (*.cxf).
2. Open the other PLC's project and open/read the function block definition file (*.cxf) that was saved.
3. Insert the function block definition in the program when creating the new program.


Function block definition A
1. Algorithm
Standard program section with variable names a, b, c, etc.
2. Variables
Table defining usage and properties of variables a, b, c, etc.
Save
Function block definition A
Function block definition file (*.cxf)
Read and insert.
Program
Input condition
Instance of function block definition A
Input 1.00
a
b
c
Output 5.00
Output 6.00

Note In the CX-Programmer IEC, each function block definition can be compiled and checked as a program. We recommend compiling to perform a program check on each function block definition file before saving or reusing the file.

SECTION 2
Creating Function Blocks

This section describes the procedures for creating function blocks on the CX-Programmer IEC.

2-1 Procedural Flow

The following procedures are used to create function blocks, save them in files, transfer them to the PLC, monitor them, and debug them.

Creating Function Blocks

Create a Project

Refer to *2-2-1 Creating a Project* for details.

■ Creating a New Project

1,2,3...

1. Start the CX-Programmer IEC and select ***New*** from the File Menu.
2. Select a *Device type* with a name ending in "(FB)."

■ Using a Non-IEC CX-Programmer Project

1,2,3...

1. Start the CX-Programmer IEC and read the project file (.cxp) created with non-IEC CX-Programmer (see note).

   Note The PLC must be the CS1-H, CS1G-H, CJ1H-H, or CJ1G-H.
2. Change the *Device type* to one with a name ending in "(FB)."

Create a Function Block Definition

Refer to *2-2-2 Creating a New Function Block Definition* for details.

1,2,3...

1. Select *Function Blocks* in the project workspace and right-click.
2. Select ***Insert Function Blocks - Ladder*** or ***Insert Function Blocks - Structured Text*** from the popup menu.

Define the Function Block

Refer to *2-2-3 Defining a Function Block* for details.

■ Registering Variables before Inputting the Ladder Program or ST Program

1,2,3...

1. Register variables in the variable table.
2. Create the ladder program or ST program.

■ Registering Variables as Necessary while Inputting the Ladder Program or ST Program

1,2,3...

1. Create the ladder program or ST program.
2. Register a variable in the variable table whenever required.

Create an Instance from the Function Block Definition

Refer to *2-2-4 Creating Instances from Function Block Definitions* for details.

■ Inserting Instances in the Ladder Section Window and then Inputting the Instance Name

1,2,3...

1. Place the cursor at the location at which to create an instance (i.e., a copy) of the function block and press the **F** Key.
2. Input the name of the instance.
3. Select the function block definition to be copied.

■ Registering Instance Names in the Global Symbol Table and then Selecting the Instance Name when Inserting

1,2,3...

1. Select *Function Block* as the data type for the variable in the global symbol table.
2. Press the **F** Key in the Ladder Section Window.
3. Select the name of the instance that was registered from the pull-down menu on the *Function Block Instance* Field.

Allocate External I/O to the Function Block

Refer to *2-2-5 Setting Function Block Parameters* for details.

1,2,3...

1. Place the cursor at the position of the input variable or output variable and press the **P** Key.
2. Input the source address for the input variable or the destination address for the output variable.

Set the Function Block Memory Allocations (Instance Areas)

Refer to *2-2-6 Setting the FB Instance Areas* for details.

1,2,3...

1. Select the instance and select ***Memory - Function Block Memory Allocation*** from the PLC Menu.
2. Set the function block memory allocations.

Saving and Reusing Function Block Files

Compile the Function Block Definition and Save It as a Library File

Refer to *2-2-9 Compiling Function Block Definitions* and *2-2-10 Saving Function Block Definitions to Files* for details.

1,2,3...

1. Compile the function block that has been saved.
2. Save the function block as a function block definition file (.cxf).
3. Read the file into another PLC project.

Transferring the Program to the PLC

Refer to *2-2-11 Downloading Programs to a CPU Unit*.

Monitoring and Debugging the Function Block

Refer to *2-2-12 Monitoring and Debugging Function Blocks*.

2-2 Procedures

2-2-1 Creating a Project

Either new projects can be created in CX-Programmer IEC or programs previously requested on non-IEC CX-Programmer can be read to create projects.

Creating New Projects with CX-Programmer IEC

1,2,3...

1. Start the CX-Programmer IEC and select ***New*** from the File Menu.
2. In the Change PLC Window, select a *Device Type* with a name ending in “(FB).” These are listed in the following table.
3. Press the **Settings** Button and select the *CPU Type.* All other settings are the same as for non-IEC CX-Programmer.

Device	CPU	Program size	Number of EM banks
CS1H-H (FB)	CPU67	250 Ksteps	13 banks
	CPU65	60 Ksteps	3 banks
CS1G-H (FB)	CPU44	30 Ksteps	1 bank
	CPU42	10 Ksteps	1 bank
CJ1G-H (FB)	CPU44	30 Ksteps	1 bank
	CPU43	20 Ksteps	1 bank
	CPU42	10 Ksteps	1 bank

Reusing Projects Created on Non-IEC CX-Programmer

1,2,3...

1. Start the CX-Programmer IEC, select ***Open*** from the File Menu, and read the project file (.cxp) created with non-IEC CX-Programmer (see note).

   Note The PLC must be the CS1-H, CS1G-H, CJ1H-H, or CJ1G-H.
2. Select the PLC name in the project workspace, right-click, and select ***Change*** from the popup menu.
3. In the Change PLC Window, select a *Device Type* with a name ending in “(FB).” These are listed in the following table.
4. Press the **Settings** Button and select the *CPU Type.* All other settings are the same as for non-IEC CX-Programmer.

Device	CPU	Program size	Number of EM banks
CS1H-H (FB)	CPU67	250 Ksteps	13 banks
	CPU65	60 Ksteps	3 banks
CS1G-H (FB)	CPU44	30 Ksteps	1 bank
	CPU42	10 Ksteps	1 bank
CJ1G-H (FB)	CPU44	30 Ksteps	1 bank
	CPU43	20 Ksteps	1 bank
	CPU42	10 Ksteps	1 bank

Note Observe the following precautions when changing the *Device type* of a project created with non-IEC CX-Programmer to one that supports function blocks.

(1) Internal Allocations for Variables
If a project file created with the non-IEC CX-Programmer is read and the *Device Type* is changed to one that supports function blocks, the default function block memory allocations (instance area, refer to *2-2-6 Setting*

the FB Instance Areas) will overlap with any of the following addresses used in the user program and errors will occur when compiling:
W000 to W511, EM 20480 to EM 32767 in the last EM bank, T1024 to T4095, and C1024 to C4095.
If addresses are duplicated and an error occurs, either change the function block memory allocations or the addresses used in the user program.

(2) Specifying the Current EM Bank
The CS1-H (FB)/CJ1-H (FB) CPU Units cannot use the current EM bank function, i.e., the EM bank must always be specified directly. For CPU Units with model numbers of CPU42, CPU43, and CPU44 there is only one EM bank, bank 0, which must be specified as E0_1000. For other CPU Units, which have more than one EM bank, the EMBC(281) instruction must be used as follows to determine the EM bank being used:

```
Example:   EMBC &2
           MOV #1111 E1000
```

Change to the following:

```
           MOV #1111 E2_1000
```

(3) Timer/Counter PV Refresh Method
The CS1-H (FB)/CJ1-H (FB) CPU Units do not support the BCD refresh method for timer/counter refresh values. Only the binary refresh method can be used. If any instructions for the BCD refresh method, such as TIM, are used in existing programs being reused on the CX-Programmer IEC, an error will occur and these instructions must be changed to the binary refresh form. Refer to *6-4 Changing the Timer/Counter PV Refresh Mode* in the *Programming Manual* for details.

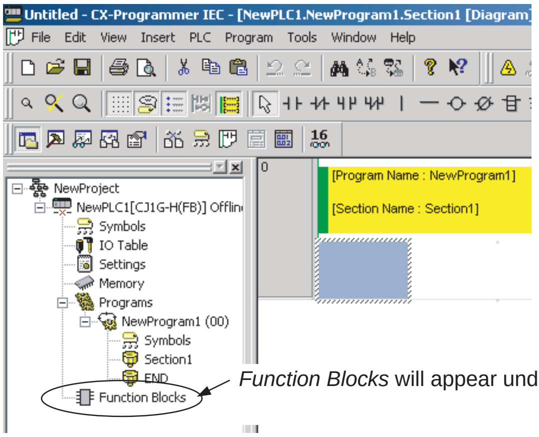
(4) Operation of Timer Instructions with Timer Numbers T2048 to T4095
If the option in the PLC properties to execute T2048 to T4095 timers the same as other timers is selected after reading the project, timers with these timer numbers will operate differently in function blocks from the same timers on the CS1-H or CJ1-H CPU Unit at the following times:

- When the cycle time is over 80 ms
- When one of these timers is in a task placed on standby with the TKON/TKOFF instructions.

To achieve the same operation as on the CS1-H or CJ1-H CPU Unit, clear the selection of the option in the PLC properties to execute T2048 to T4095 timers the same as other timers. Function blocks, however, use timer numbers T3072 to T4095 by default. Timer instructions with timer numbers T0000 to T2047 will thus operate differently in the main programs from those in function blocks. To solve this problem and achieve the same operation, change the timer numbers used by function blocks to T0000 to T2047. Refer to *3-5-3 Operation of Timer Instructions* for details.

2-2-2 Creating a New Function Block Definition

1,2,3... 1. When a project is created, a *Function Blocks* icon will appear in the project workspace as shown below.

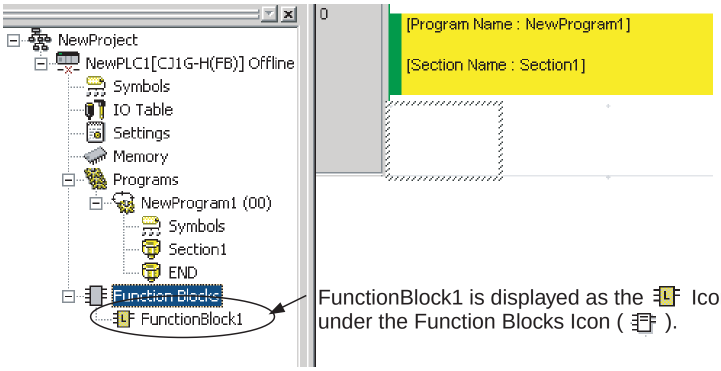

Untitled - CX-Programmer IEC - [NewPLC1.NewProgram1.Section1 [Diagram]
File Edit View Insert PLC Program Tools Window Help
16
0
NewProject
NewPLC1[CJ1G-H(FB)] Offline
Symbols
IO Table
Settings
Memory
Programs
NewProgram1 (00)
Symbols
Section1
END
Function Blocks
[Program Name : NewProgram1]
[Section Name : Section1]

Function Blocks will appear under the PLC.

2. Function blocks are created by inserting function block definitions after the Function Blocks icon. Function block can be defined using either ladder programming or structured text.

   • Defining Function Blocks with Ladders
   Select *Function Blocks* in the project workspace, right-click, and select ***Insert Function Blocks - Ladder*** from the popup menu. (Or select ***Function Block - Ladder*** from the Insert Menu.)

   • Defining Function Blocks with Structured Text
   Select *Function Blocks* in the project workspace, right-click, and select ***Insert Function Blocks - Structured Text*** from the popup menu. (Or select ***Function Block - Structured Text*** from the Insert Menu.)



FunctionBlock1 is displayed as the Icon under the Function Blocks Icon ( ).

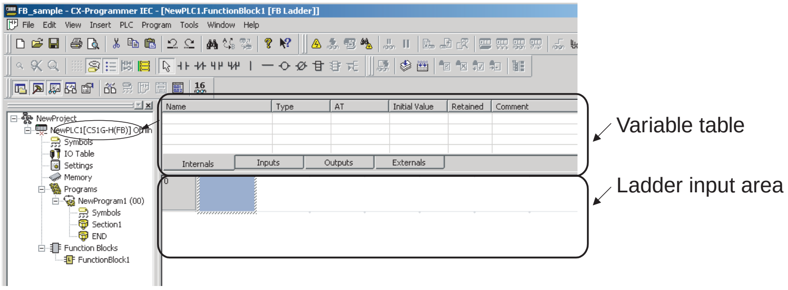
3. By default, a function block called FunctionBlock1 will be automatically inserted after the Function Blocks icon. This icon contains the definitions for the function block.

4. Whenever a function block definition is created, the name FunctionBlock□ will be assigned automatically, where □ is a serial number. These names can be changed. All names must contain no more than 64 characters.

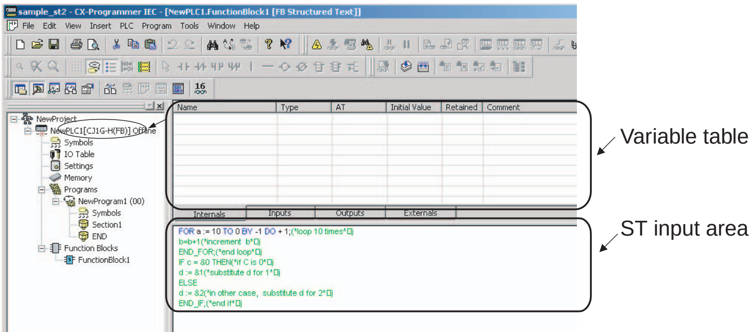
Function Block Definitions

One of the following windows will be displayed when the function block icon is double-clicked (or if it is right-clicked and ***Open*** is selected from the popup menu). A variable table for the variables used in the function block is displayed on top and an input area for the ladder program or structured text is displayed on the bottom.

Ladder Program



Structured Text



```
FOR a := 10 TO 0 BY -1 DO + 1;(*loop 10 times*□j
b=b+1(*increment  b*□j
END_FOR;(*end loop*□j
IF c = &0 THEN(*if C is 0*□j
d := &1(*substitute d for 1*□j
ELSE
d := &2(*in other case,  substitute d for 2*□j
END_IF;(*end if*□j
```

As shown, a function block definition consists of a variable table that serves as an interface and a ladder program or structured text that serves as an algorithm.

Variable Table as an Interface

At this point, the variable table is empty because there are no variables allocated for I/O memory addresses in the PLC.

Ladder Program or Structure Text as an Algorithm

- With some exceptions, the ladder program for the function block can contain any of the instructions used in the normal program. Refer to *3-3 Restrictions on Function Blocks* for restrictions on the instructions that can be used.
- Structured text can be input according to the ST language defined in IEC61131-3.

2-2-3 Defining a Function Block

A function block is defined by registering variables and creating an algorithm. There are two ways to do this.

- Register the variables first and then input the ladder program or structure text.
- Register variables as they are required while inputting input the ladder program or structure text.

Registering Variables First

Registering Variables in the Variable Table

The variables are divided by type into four sheets in the variable table: Internals, Inputs, Outputs, and Externals.

These sheets must be switched while registering or displaying the variables.

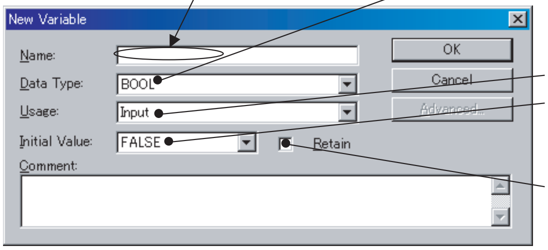
1,2,3...

1. Make the sheet for the type of variable to be registered active in the variable table. (See note.) Place the cursor in the sheet, right-click, and select ***Insert Variable*** from the popup menu.

   Note The sheet where a variable is registered can also be switched by setting the *Usage*.

   The New Variable Dialog Box shown below will be displayed.

   - **Name:** Input the name of the variable.
   - **Data Type:** Select the data type.
   - **Usage:** Select the variable type.
   - **Initial Value:** Select the initial value of the variable at the start of operation.
   - **Retain:** Select if the value of the variable is to be maintained when the power is turned ON or when the operating mode is changed from PROGRAM or MONITOR mode to RUN mode. The value will be cleared at these times if *Retain* is not selected.



Name:
Data Type: BOOL
Usage: Input
Initial Value: FALSE
□ Retain
Comment:

Input the name of the function block variable

The default data type is BOOL. Change as required.

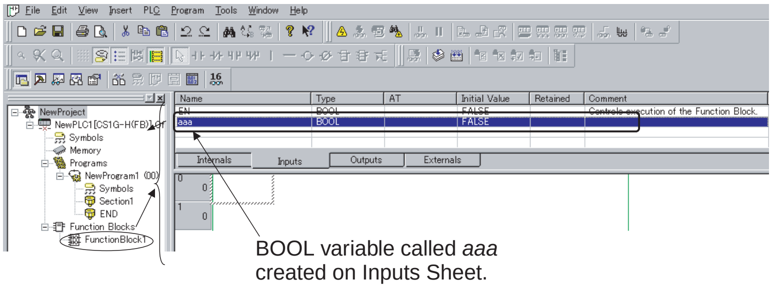
Type of variable to register (i.e., the sheet)

Initial value

Select to maintain value for power interruptions.

2. For example, input “aaa” as the variable name and click the **OK** Button.

   As shown below, a BOOL variable called *aaa* will be created on the Inputs Sheet of the Variable Table.



Name	Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
aaa	BOOL		FALSE		

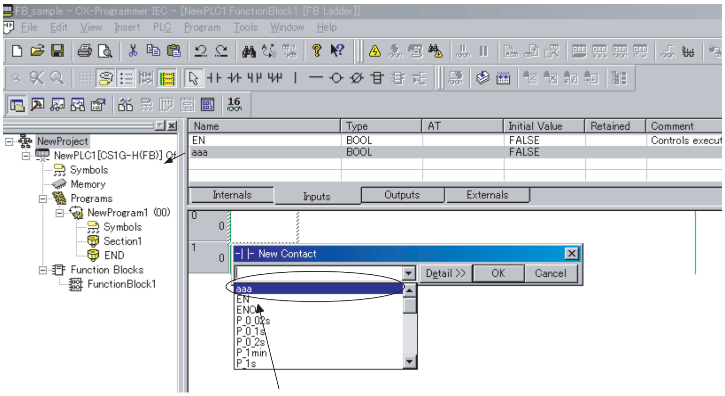
BOOL variable called *aaa* created on Inputs Sheet.

Creating the Algorithm

Using a Ladder Program

1,2,3...

1. Press the **C** Key and select *aaa* registered earlier from the pull-down menu in the New Contact Dialog Box.



Name	Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execut
aaa	BOOL		FALSE		

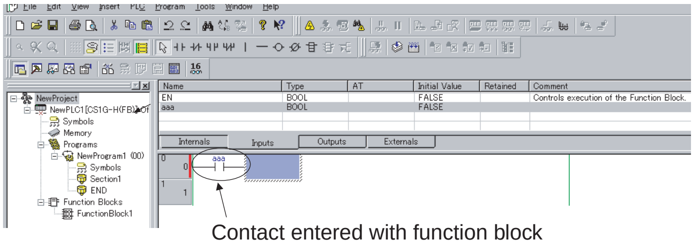
Press the C Key and select *aaa* registered earlier from the pull-down menu in the New Contact Dialog Box.

Note A name must be input for variables, even ones with AT settings (specified address). With CX-Programmer IEC, the following characters can be input as the variable name to indicate I/O memory addresses. (This is not possible with non-IEC CX-Programmer.)

- A, W, H, HR, D, DM, E, EM, T,TM, C, or CNT followed by a number (channel/word address)
- A period to differentiate between channel (word) and bit addresses.

For example, when Auxiliary Area addresses are specified as ATs, the I/O memory address (e.g., A50200) can be specified as the variable name to make assignments easier to understand. (Even when this is done, the actual address must be specified in the AT settings.)

2. Click the **OK** Button. A contact will be entered with the function block internal variable *aaa* as the operand (variable type: internal).



Name	Type	AT	Initial Value	Retained	Comment
EN	BOOL		FALSE		Controls execution of the Function Block.
aaa	BOOL		FALSE		

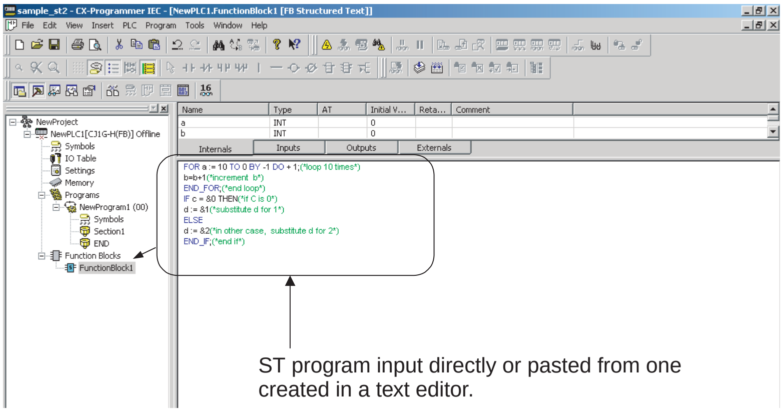
Contact entered with function block internal variable *aaa* as operand.

The rest of the ladder program is input the same as for normal programs with non-IEC CX-Programmer.

Using Structured Text

An ST language program (see note) can either be input directly into the ST input area or a program input into a general-purpose text editor can be copied and then pasted into the ST input area using the *Paste* Command on the Edit Menu.

Note The ST language conforms to IEC61131-3, but only assignment statements, selection statements (CASE and IF), iteration statements (FOR, WHILE, and REPEAT), arithmetic operations, logic operations, comparison operations, and comments. All other elements are not supported. Refer to *Appendix B Structured Text Keywords* for details.


sample_st2 - CX-Programmer IEC - [NewPLC1.FunctionBlock1 [FB Structured Text]]
File Edit View Insert PLC Program Tools Window Help
NewProject
NewPLC1[CJ1G-H(FB)] Offline
Symbols
IO Table
Settings
Memory
Programs
NewProgram1 (00)
Symbols
Section1
END
Function Blocks
FunctionBlock1
Internals
Inputs
Outputs
Externals

Name	Type	AT	Initial V...	Reta...	Comment
a	INT		0		
b	INT		0		

```
FOR a := 10 TO 0 BY -1 DO + 1;(*loop 10 times*)
b=b+1(*increment  b*)
END_FOR;(*end loop*)
IF c = &0 THEN(*if C is 0*)
d := &1(*substitute d for 1*)
ELSE
d := &2(*in other case,  substitute d for 2*)
END_IF;(*end if*)
```

ST program input directly or pasted from one created in a text editor.

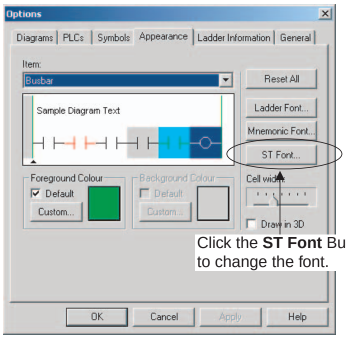
Note

(1) Tabs or spaces can be input to create indents. They will not affect the algorithm.

(2) The display size can be changed by holding down the **Ctrl** Key and turning the scrolling wheel on a wheel mouse.

(3) When an ST language program is input or pasted into the ST input area, syntax keywords will be automatically displayed in blue, errors in red, comments in green, and everything else in black.

(4) To change the font size or colors, select ***Options*** from the Tools Menu and then click the **ST Font** Button on the Appearance Tab Page.



Item: Busbar

Foreground Colour ☑ Default

Background Colour □ Default

Cell width:

□ Draw in 3D

Click the **ST Font** Button to change the font.

Registering Variables as Required

The ladder program or structured text program can be input first and variable registered as they are required.

Using a Ladder Program

When using a ladder diagram, a dialog box will be displayed to register the variable whenever a variable name that has not been registered is input. The variable is registered at that time.

Use the following procedure.

1,2,3...

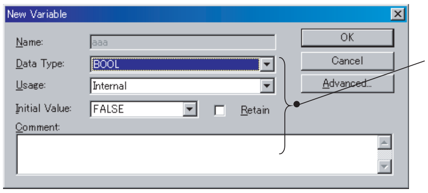
1. Press the **C** Key and input a variable name that has not been registered, such as *aaa,* in the New Contact Dialog Box.

   Note A name must be input for variables, even ones with AT settings (specified address). With CX-Programmer IEC, the following characters can be input as the variable name to indicate I/O memory addresses. (This is not possible with non-IEC CX-Programmer.)

   - A, W, H, HR, D, DM, E, EM, T,TM, C, or CNT followed by a number (channel/word address)
   - A period to differentiate between channel (word) and bit addresses.

   For example, when Auxiliary Area addresses are specified as ATs, the I/O memory address (e.g., A50200) can be specified as the variable name to make assignments easier to understand. (Even when this is done, the actual address must be specified in the AT settings.)

2. Click the **OK** Button. The New Variable Dialog Box will be displayed. With special instructions, a New Variable Dialog Box will be display for each operand in the instruction.



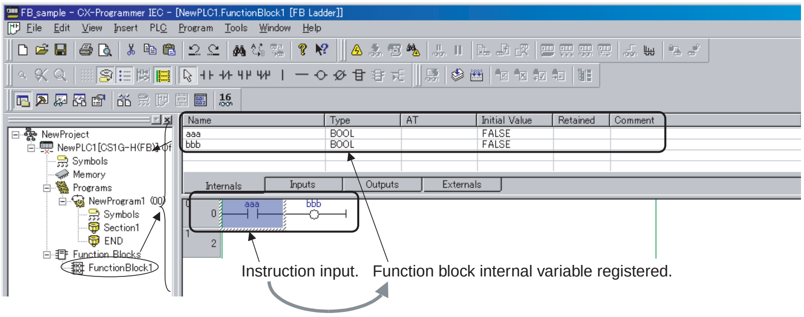
Name: aaa
Data Type: BOOL
Usage: Internal
Initial Value: FALSE
□ Retain
Comment:

Set the data type and other properties other than the name.

   The properties for all input variables will initially be displayed as follows:

   - Usage: Internal
   - Data Type: BOOL for contacts and WORD for channel (word)
   - Initial Value: The default for the data type.
   - Retain: Not selected.

3. Make any required changes and click the **OK** Button.
4. As shown below, the variable that was registered will be displayed in the variable table above the program.



Name	Type	AT	Initial Value	Retained	Comment
aaa	BOOL		FALSE		
bbb	BOOL		FALSE		

Instruction input. Function block internal variable registered.

5. If the type or properties of a variable that was input are not correct, double-click the variable in the variable table and make the required corrections.

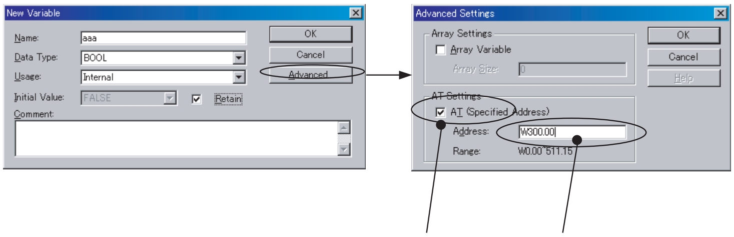
■ Reference Information

AT Settings (Specified Address)

AT settings can be made to specify CIO or DM Area addresses allocated to a Special I/O Unit or Auxiliary Area addresses not registered in the CX-Programmer IEC. A variable name is required to achieve this. Use the following procedure to specify an address.

1,2,3...

1. After inputting the variable name in the New Variable Dialog Box, click the **Advanced** Button. The Advanced Settings Dialog Box will be displayed.
2. Select *AT (Specified Address)* under *AT Settings* and input the desired address.



New Variable: Name: aaa; Data Type: BOOL; Usage: Internal; Initial Value: FALSE; ☑ Retain; Comment:

Advanced Settings: Array Settings: □ Array Variable; Array Size: 0; AT Settings: ☑ AT (Specified Address); Address: W300.00; Range: W0.00˜511.15

Select *AT*. Input address.

The variable name is used to enter variables into the algorithm in the function block definition even when they have an address specified for the AT settings (the same as for variables without a specified address).
For example, if a variable named *Restart* has an address of A50100 specified for the AT settings, *Restart* is specified for the instruction operand.

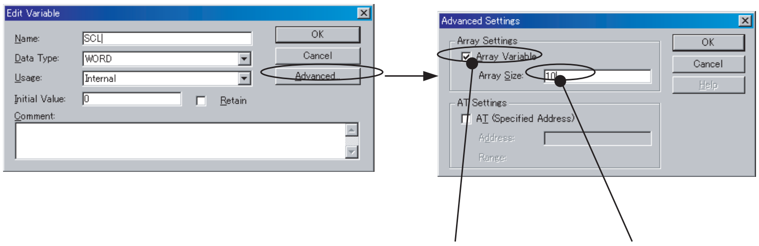
Array Settings

An array can be specified to use the same data properties for more than one variable and manage the variables as a group.

Use the following procedure to set an array.

1,2,3...

1. After inputting the variable name in the New Variable Dialog Box, click the **Advanced** Button. The Advanced Settings Dialog Box will be displayed.
2. Select *Array Variable* in the *Array Settings* and input the maximum number of elements in the array.



Edit Variable: Name: SCL; Data Type: WORD; Usage: Internal; Initial Value: 0; □ Retain; Comment:

Advanced Settings: Array Settings: ☑ Array Variable; Array Size: 10; AT Settings: □ AT (Specified Address); Address: ; Range:

Select *Array Variable.* Input the number of elements.

When the name of a variable array is entered in the algorithm in the function block definition, square brackets surrounding the index will appear after the array name.

For example, if you create a variable named PV with a maximum of 3 elements, PV[0], PV[1], and PV[2] could be specified as instruction operands.

There are three ways to specify indices.

- Directly with numbers, e.g., PV[1] in the above example (for ladder programming or ST language programming)

- With a variable, e.g., PV[a] in the above example, where “a” is the name of a variable with a data type of INT (for ladder programming or ST language programming)
- With an equation, e.g., PV[a+b] or PV[a+1} in the above example, where “a” and “b” are the names of variables with a data type of INT (for ST language programming only)

Using an Array to Specify Words Allocated to CPU Bus Units

The first DM Area word allocated to a CS-series or CJ-series CPU Bus Unit is expressed by the following formula:

D30000 + unit number × 100

Thus an array variable can be used to indirectly address DM Area words allocated to CPU Bus Units by using a formula containing the unit number as an index to the array.

For example, the following could be done if the unit number is given by the variable named N and the variable named DataMemory is an array variable for the DM Area words allocated to the CPU Bus Unit.

1,2,3...

1. Register the variable DataMemory as an array variable with a maximum of 1,600 elements.
2. To designate the DM Area word that is *s* words from the first allocated word (where *s* is either a variable or a direct offset in number of words), the following variable would be used and the AT setting for the Data Memory variable would be set to D30000.

   DataMemory[N*100+s]
3. The function block definition would then be placed in the program and words allocated to the CPU Bus Unit could be specified merely by passing the unit number (using N in the above example) to the instance. For example, if a value of 5 was passed for N, D30500 would be specified.

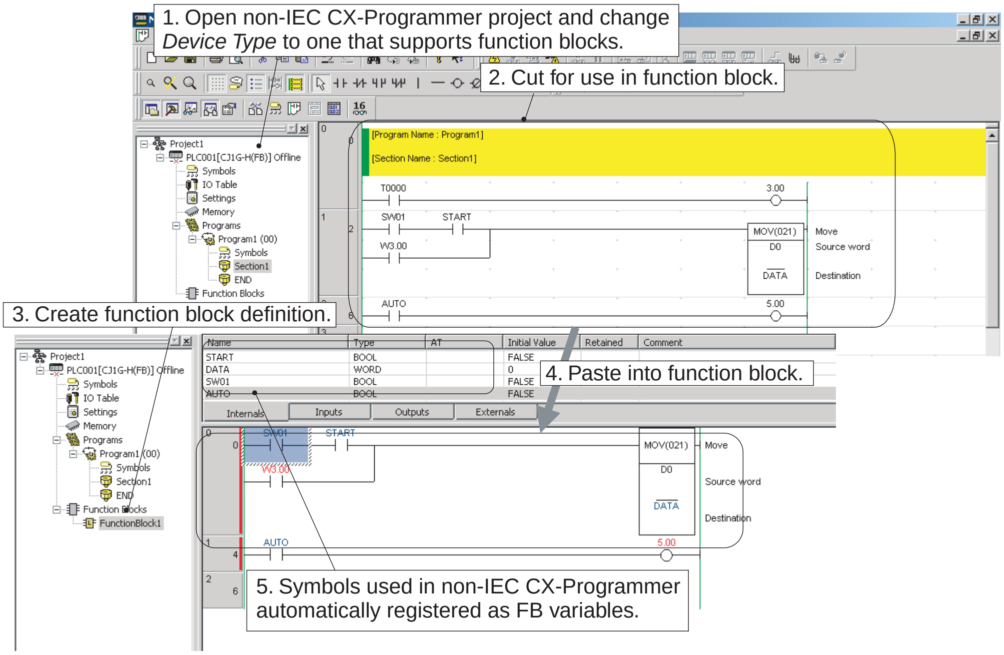
Reusing Non-IEC CX-Programmer Projects (.cxp)

1,2,3... 1. Read the non-IEC CX-Programmer project (.cxp) and change the *Device Type* to one that supports function blocks.

2. Cut the rungs to be used in the function block.

3. Create a new function block definition.

4. Paste the rungs into the function block.



1. Open non-IEC CX-Programmer project and change *Device Type* to one that supports function blocks.

2. Cut for use in function block.

3. Create function block definition.

4. Paste into function block.

5. Symbols used in non-IEC CX-Programmer automatically registered as FB variables.

Name	Type	AT	Initial Value	Retained	Comment
START	BOOL		FALSE		
DATA	WORD		0		
SW01	BOOL		FALSE		
~~AUTO~~	~~BOOL~~		FALSE		

5. When the rungs are pasted, any symbols used in non-IEC CX-Programmer will automatically be registered in the variable table of the function block. Any addresses that were specified directly in non-IEC CX-Programmer will be displayed in red and nothing will be registered for them. Change all of these to variables.

Using Structured Text

When using structured text, a dialog box will not be displayed to register the variable whenever a variable name that has not been registered is input. Be sure to always register variables used in standard text programming in the variable table, either as you need them or after completing the program. (Place the cursor in the tab page on which to register the variable, right-click, and select ***Insert Variable*** from the popup menu.

2-2-4 Creating Instances from Function Block Definitions

If a function block definition is registered in the global symbol table, either of the following methods can be used to create instances.

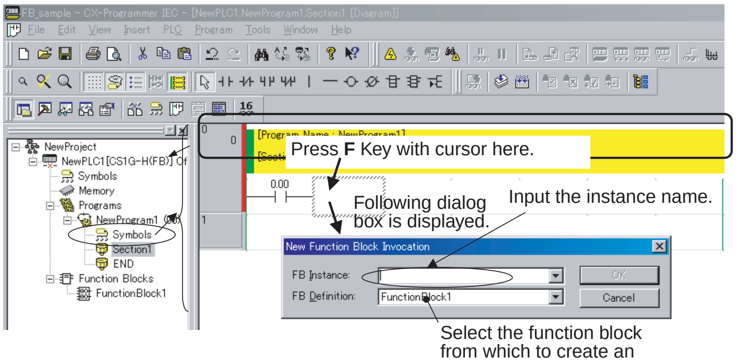
Method 1:Select the function block definition, insert it into the program, and input a new instance name. The instance will automatically be registered in the global symbol table.

Method 2: Set the data type in the global symbol table to “function block,” specify the function block definition to use, and input the instance name to register it.

■ Method 1: Using the F Key in the Ladder Section Window and Inputting the Instance Name

1,2,3...

1. In the Ladder Section Window, place the cursor in the program where the instance is to be inserted and press the **F** Key. (Alternately, select ***Function Block Invocation*** from the Insert Menu.) The New Function Block Invocation Dialog Box will be displayed.
2. Input the instance name, select the function block from which to create an instance, and click the **OK** Button.


FB_sample - CX-Programmer IEC - [NewPLC1.NewProgram1.Section1 [Diagram]]
File Edit View Insert PLC Program Tools Window Help
NewProject
NewPLC1[CS1G-H(FB)] Of
Symbols
Memory
Programs
NewProgram1 (00)
Symbols
Section1
END
Function Blocks
FunctionBlock1
[Program Name : NewProgram1]
0.00
New Function Block Invocation
FB Instance:
FB Definition:
FunctionBlock1
OK
Cancel

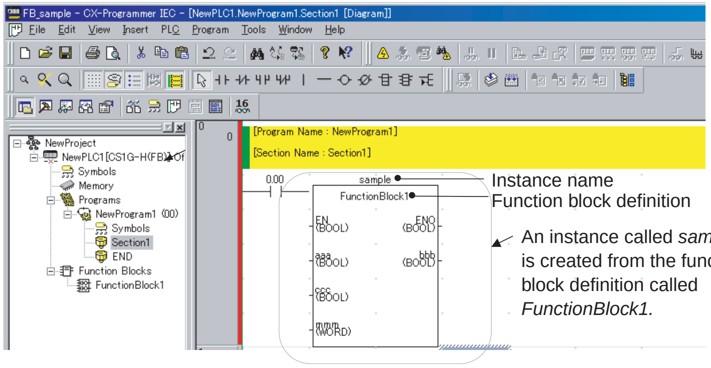
Press **F** Key with cursor here.

Following dialog box is displayed.

Input the instance name.

Select the function block from which to create an

3. As an example, set the instance name in the *FB Instance* Field to ***sample***, set the function block in the *FB Definition* Field to ***FunctionBlock1***, and click the **OK** Button. As shown below, a copy of the function block definition called *FunctionBlock1* will be created with an instance name of *sample.*



Instance name

Function block definition

An instance called *sample* is created from the function block definition called *FunctionBlock1.*

The instance will be automatically registered in the global symbol table with an instance name of *sample* and a data type of *Function block.*

■ Method 2: Registering the Instance Name in the Global Symbol Table in Advance and Then Selecting the Instance Name

If the instance name is registered in the global symbol table in advance, the instance name can be selected from the global symbol table to create other instances.

1,2,3...

1. Select a data type of *Function block* in the global symbol table, input the instance name, and registered the instance.
2. Press the **F** Key in the Ladder Section Window. The Function Block Invocation Dialog Box will be displayed.
3. Select the instance name that was previously registered from the pulldown menu on the *FB Instance* Field. The instance will be created.

Restrictions

Observe the following restrictions when creating instances. Refer to *3-3 Restrictions on Function Blocks* for details.

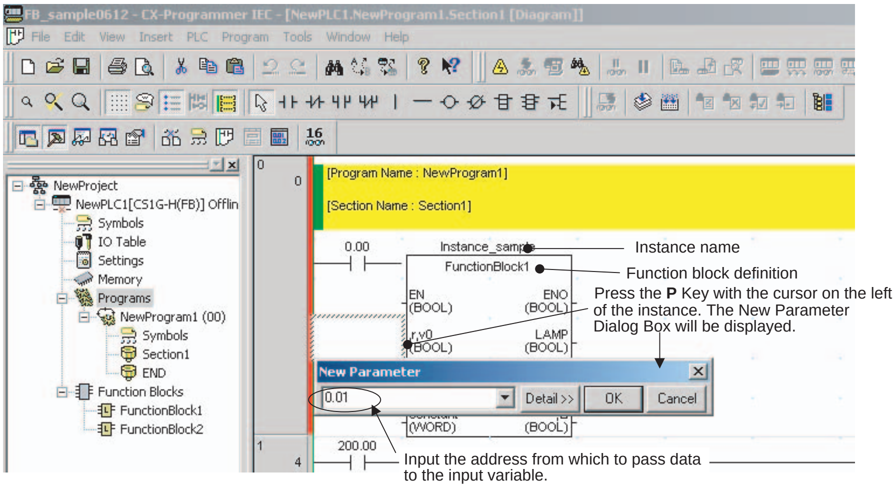
- No more than one function block can be created in each program circuit.
- The rung cannot be branched to the left of an instance.
- Instances cannot be connected directly to the left bus bar, i.e., an EN must always be inserted.

Note If changes are made in the I/O variables in a variable table for a function block definition, the bus bar to the left of all instances that have been created from that function block definition will be displayed in red to indicate an error. When this happens, select each instance, right-click, and select ***Update Invocation***. The instance will be updated for any changes that have been made in the function block definition and the red display will be cleared.

2-2-5 Setting Function Block Parameters

After an instance of a function block has been created, input parameters must be set for input variables and output parameters must be set for output variables to enable external I/O.

1,2,3... 1. Inputs are located on the left of the instance and outputs on the right. Place the cursor where the parameter is to be set and press the **P** Key. (Alternately, select ***Function Block Parameter*** from the Insert Menu.) The New Parameter Dialog Box will be displayed as shown below.



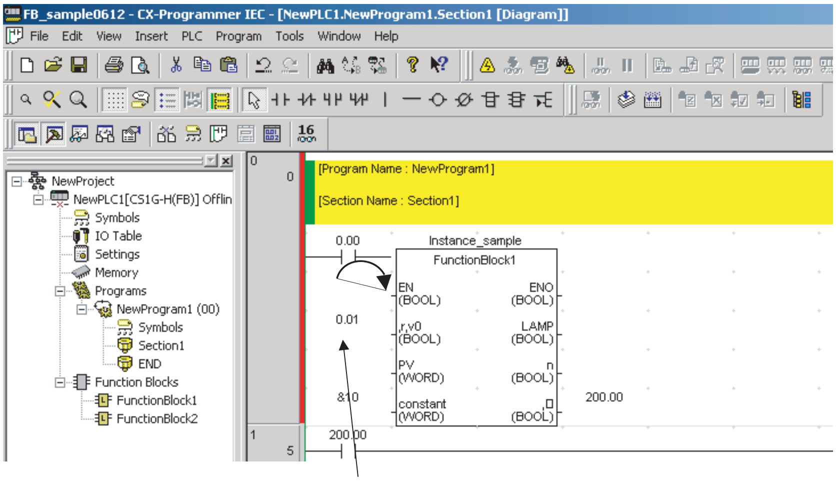
Instance name

Function block definition

Press the **P** Key with the cursor on the left of the instance. The New Parameter Dialog Box will be displayed.

Input the address from which to pass data to the input variable.

2. Input the address from which to pass status data to the input variable.



The value of 001 is passed to input variable aaa.

3. Input the addresses from/to which to pass data for the other input and output variables.

2-2-6 Setting the FB Instance Areas

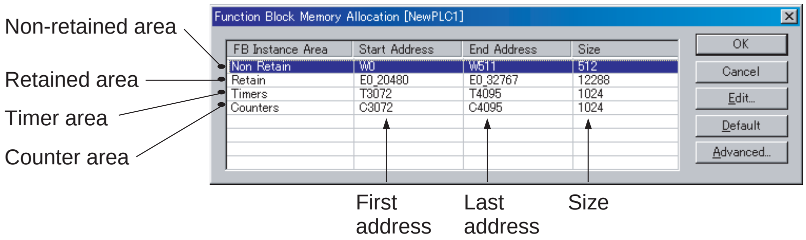
The areas where addresses for variables used in function blocks are allocated can be set. These areas are called the function block instance areas.

1,2,3...

1. Select the instance in the Ladder Section Window or in the global symbol table, and then select ***Memory - Function Block Memory Allocation*** from the PLC Menu.

   The Function Block Memory Allocation Dialog shown below will appear.

2. Set the FB instance areas.



FB Instance Area	Start Address	End Address	Size
Non Retain	W0	W511	512
Retain	E0_20480	E0_32767	12288
Timers	T3072	T4095	1024
Counters	C3072	C4095	1024

The non-retained and retained areas are set in words. The timer and counter areas are set by time and counter numbers.

The default values are as follows:

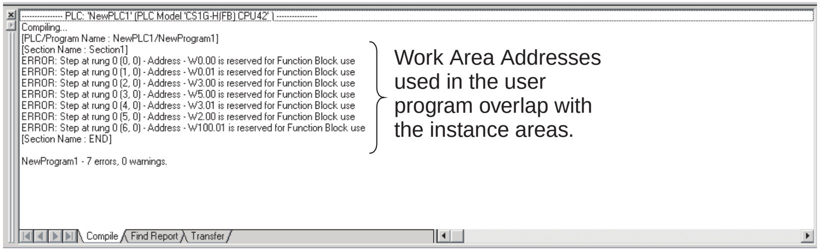
FB instance area	Start address	End address	Applicable memory areas
Non-retained area	W0	512	CIO, WR, HR, DM, EM (See note b.)
Retained area	E0_20480 (See note a.)	12,288	HR, DM, EM (See note b.)
Timer area	T3072	1,024	TIM
Counter area	C3072	1,024	CNT

Note (a) E20480 to E32767 in the last EM Area bank is the default setting. The number of the last EM Area bank depends on the model of CPU Unit being used.

(b) Bit data can be accessed even if the DM or EM Area is specified.

Note Overlapping of Instance Area Addresses and Address Used in the Program

If the addresses in the function block instance areas overlap with any of the addresses used in the user program, an error will occur when compiling. This error will also occur when a program is downloaded, edited online, or checked by the user.



```
--------------- PLC: 'NewPLC1' (PLC Model 'CS1G-H(FB) CPU42' ) ---------------
Compiling...
[PLC/Program Name : NewPLC1/NewProgram1]
[Section Name : Section1]
ERROR: Step at rung 0 (0, 0) - Address - W0.00 is reserved for Function Block use
ERROR: Step at rung 0 (1, 0) - Address - W0.01 is reserved for Function Block use
ERROR: Step at rung 0 (2, 0) - Address - W3.00 is reserved for Function Block use
ERROR: Step at rung 0 (3, 0) - Address - W5.00 is reserved for Function Block use
ERROR: Step at rung 0 (4, 0) - Address - W3.01 is reserved for Function Block use
ERROR: Step at rung 0 (5, 0) - Address - W2.00 is reserved for Function Block use
ERROR: Step at rung 0 (6, 0) - Address - W100.01 is reserved for Function Block use
[Section Name : END]

NewProgram1 - 7 errors, 0 warnings.
```

Work Area Addresses used in the user program overlap with the instance areas.

If addresses are duplicated and an error occurs, either change the function block instance areas or the addresses used in the user program.

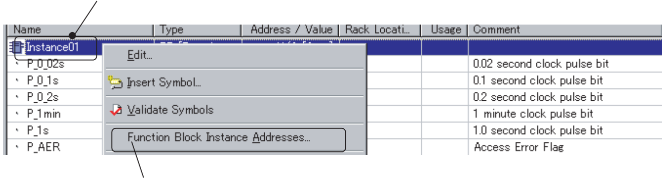
2-2-7 Checking Internal Address Allocations for Variables

The following procedure can be used to check the I/O memory addresses internally allocated to variables.

1,2,3...

1. Select ***View - Symbols - Global.***
2. Select the instance in the global symbol table, right-click, and select ***Function Block Memory Address*** from the popup menu. (Alternately, select ***Memory - Function Block Memory Address*** from the PLC Menu.)

Example: Instance name displayed in global variable table (automatically registered)

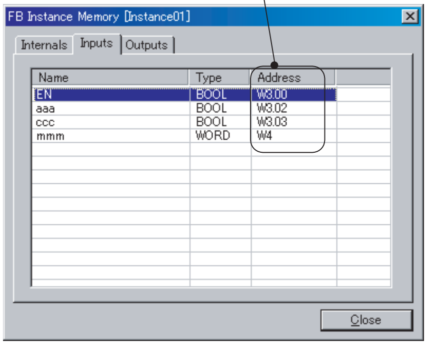


Name	Type	Address / Value	Rack Locati...	Usage	Comment
Instance01					
P_0_02s					0.02 second clock pulse bit
P_0_1s					0.1 second clock pulse bit
P_0_2s					0.2 second clock pulse bit
P_1min					1 minute clock pulse bit
P_1s					1.0 second clock pulse bit
P_AER					Access Error Flag

Right-click on the instance name and select ***Function Block Instance Address.***

3. The FB Interface Memory Dialog Box will be displayed. Check the I/O memory addresses internally allocated to variables here.

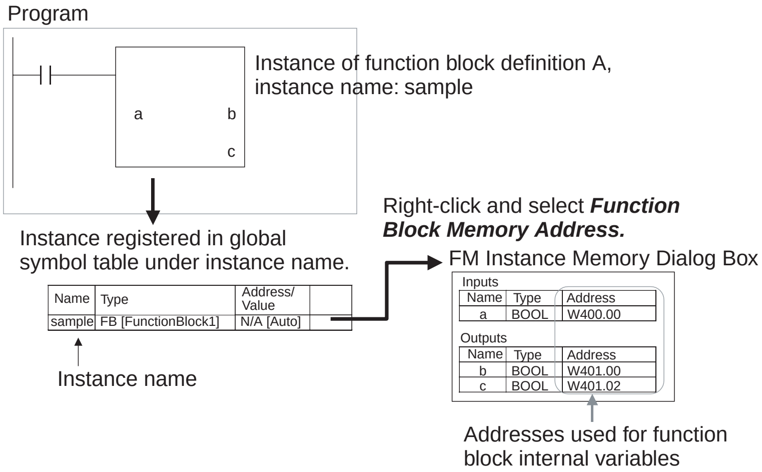
Example: Addresses used internally for the input variables.



FB Instance Memory [Instance01]

Name	Type	Address
EN	BOOL	W3.00
aaa	BOOL	W3.02
ccc	BOOL	W3.03
mmm	WORD	W4

Method Used for Checking Addresses Internally Allocated to Variables



Instance of function block definition A, instance name: sample

Instance registered in global symbol table under instance name.

Name	Type	Address/ Value
sample	FB [FunctionBlock1]	N/A [Auto]

Instance name

Right-click and select ***Function Block Memory Address.***

FM Instance Memory Dialog Box

Inputs

Name	Type	Address
a	BOOL	W400.00

Outputs

Name	Type	Address
b	BOOL	W401.00
c	BOOL	W401.02

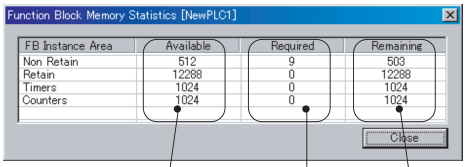
Addresses used for function block internal variables

Checking the Status of Addresses Internally Allocated to Variables

The following procedure can be used to check the number of addresses allocated to variables and the number still available for allocation in the function block instance areas.

1,2,3...

1. Select the instance in the Ladder Section Window, right-click, and select ***Memory - Function Block Memory Statistics*** from the PLC Menu.
2. The Function Block Memory Statistics Dialog Box will be displayed as shown below. Check address usage here.



FB Instance Area	Available	Required	Remaining
Non Retain	512	9	503
Retain	12288	0	12288
Timers	1024	0	1024
Counters	1024	0	1024

The total number of words in each interface area.

The number of words already used.

The number of words still available.

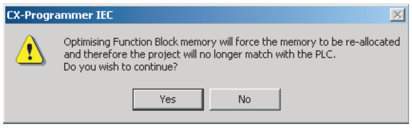
Optimizing Function Memory

When a variable is added or deleted, addresses are automatically re-allocated in the variables' instance area. Consecutive addresses are required for each instance, so all of the variables will be allocated to a different block of addresses if the original block of addresses cannot accommodate the change in variables. This will result in an unused block of addresses. The following procedure can be used to eliminate the unused areas in memory so that memory is used more efficiently.

1,2,3...

1. Select the instance in the Ladder Section Window, right-click, and select ***Memory - Optimize Function Memory*** from the PLC Menu.

   The following dialog box will be displayed.



Optimising Function Block memory will force the memory to be re-allocated and therefore the project will no longer match with the PLC.
Do you wish to continue?

2. Click the **OK** Button. Allocations to the function block instance areas will be optimized.

2-2-8 Checking the Function Block Definition for an Instance

Use the following procedure to check the function block definition from which an instance was created.

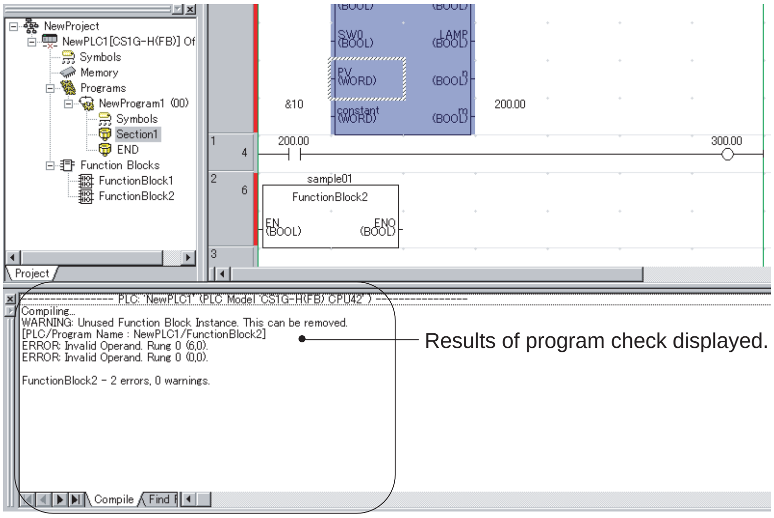
1,2,3... Right-click the instance and select ***Go To - Function Block Definition*** from the popup menu. The function block definition will be displayed.

2-2-9 Compiling Function Block Definitions

A function block definition can be compiled to perform a program check on it. Use the following procedure.

1,2,3... Select the function block definition, right-click, and select ***Compile*** from the popup menu. (Alternately, press the **Ctrl + F7** Keys.)

The function block will be compiled and the results of the program check will be automatically displayed on the Compile Table Page of the Output Window.



```
---------------- PLC: 'NewPLC1' (PLC Model 'CS1G-H(FB) CPU42' ) ----------------
Compiling...
WARNING: Unused Function Block Instance. This can be removed.
[PLC/Program Name : NewPLC1/FunctionBlock2]
ERROR: Invalid Operand. Rung 0 (6,0).
ERROR: Invalid Operand. Rung 0 (0,0).

FunctionBlock2 - 2 errors, 0 warnings.
```

Results of program check displayed.

2-2-10 Saving Function Block Definitions to Files

A function block definition can be saved as a function block library file (extension: .cxf) to enable reusing it in other projects.

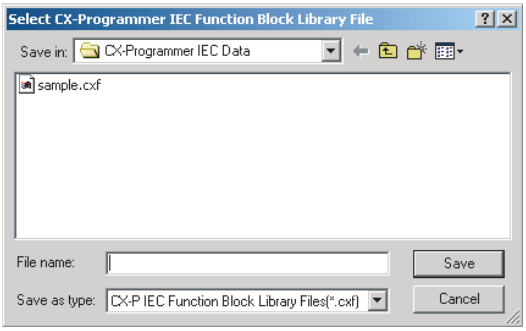
Saving a Function Block Library File

Use the following procedure to save a function block definition to a function block library file.

1,2,3...

1. Select the function block definition, right-click, and select ***Save Function Block File*** from the popup menu. (Alternately, select ***Save Function Block File*** from the File Menu.)

2. The following dialog box will be displayed. Input the file name. *CX-P IEC function block library files (*.cxf)* should be selected as the file type.



Select CX-Programmer IEC Function Block Library File

Save in: CX-Programmer IEC Data

sample.cxf

File name:

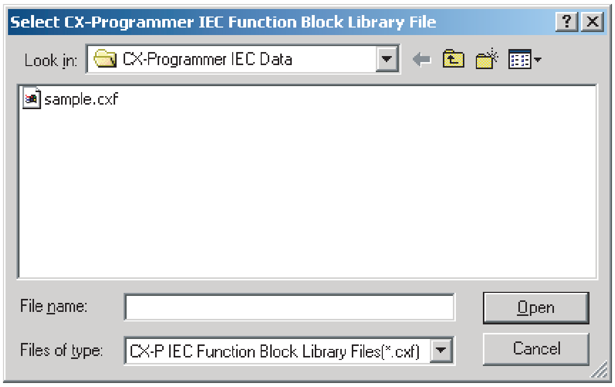
Save as type: CX-P IEC Function Block Library Files(*.cxf)

Reading Function Block Library Files into Other Projects

Use the following procedure to read a function block library file (*.cxf) into a project.

1,2,3...

1. Select the function block definition item in the Project Workspace, right-click, and select ***Insert Function Block - From File*** from the popup menu.
2. The following dialog box will be displayed. Select a function block library file (*.cxf) and click the **Open** Button.



Select CX-Programmer IEC Function Block Library File

Look in: CX-Programmer IEC Data

sample.cxf

File name:

Files of type: CX-P IEC Function Block Library Files(*.cxf)

3. A function block called FunctionBlock1 will be automatically inserted after the Function Blocks icon. This icon contains the definition of the function block.
4. Double-click the **FunctionBlock1** Icon. The variable table and algorithm will be display.

2-2-11 Downloading Programs to a CPU Unit

After a program containing function blocks has been created, it can be downloaded from the CX-Programmer IEC to a CPU Unit that is connected online. It is also possible to check if the programs on the CX-Programmer IEC and in the CPU Unit are the same.

Programs cannot be uploaded from the CPU Unit.

2-2-12 Monitoring and Debugging Function Blocks

The following procedures can be used to monitor programs containing function blocks.

Monitoring Programs in Function Block Definitions

Use the following procedure to check the program in the function block definition for an instance during monitoring.

1,2,3... Right-click the instance and select ***Go To - Function Block Definition*** from the popup menu. The function block definition will be displayed.

Monitoring Instance Variables in the Watch Window

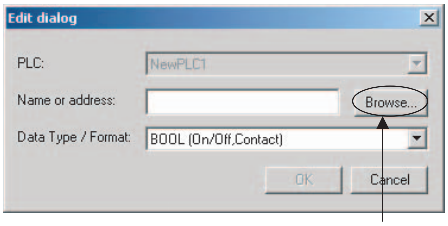
Use the following procedure to monitor instance variables.

1,2,3...

1. Select ***View - Window - Watch.***

   A Watch Window will be displayed.
2. Double-click the watch window.

   The Edit Dialog Box will be displayed as shown below.



Edit dialog

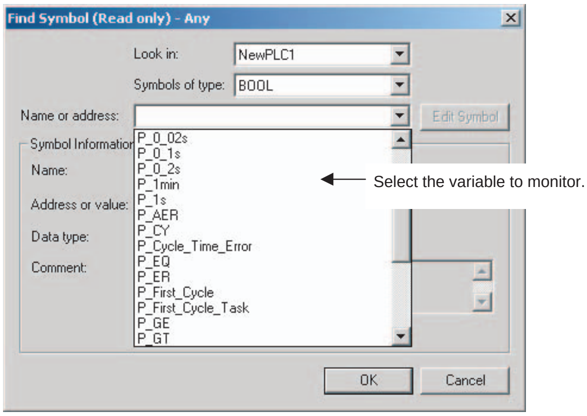
PLC: NewPLC1

Name or address:

Data Type / Format: BOOL (On/Off,Contact)

Click the **Browse** Button.

3. Click the **Browse** Button, select the variable to be monitored, and click the **OK** Button.



Find Symbol (Read only) - Any

Look in: NewPLC1

Symbols of type: BOOL

Name or address:

Symbol Information

Name:

Address or value:

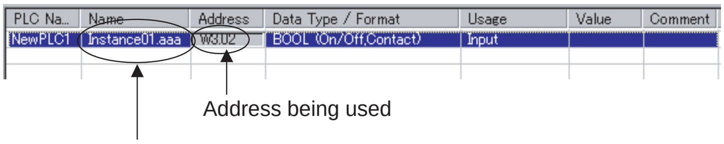
Data type:

Comment:

P_0_02s
P_0_1s
P_0_2s
P_1min
P_1s
P_AER
P_CY
P_Cycle_Time_Error
P_EQ
P_ER
P_First_Cycle
P_First_Cycle_Task
P_GE
P_GT

Select the variable to monitor.

4. Click the **OK** Button. Variable values will be display in the Watch Window as shown below.



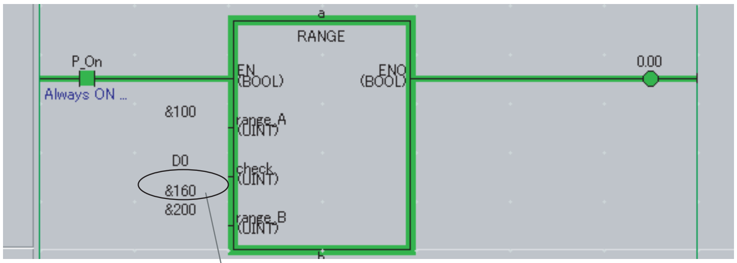
PLC Na...	Name	Address	Data Type / Format	Usage	Value	Comment
NewPLC1	Instance01.aaa	W3.02	BOOL (On/Off,Contact)	Input		

Address being used

Variable name

Monitoring Instance I/O Variables

The present values of parameters for I/O variables are displayed below the parameters.



PV of parameter for I/O variable.

Editing Function Block Definition Programs Online

Programs using function blocks can be edited online. Changes can also be made around instances.

- Instance parameters can be changed, instances can be deleted, and instructions other than those in instances can be changed.
- Instances cannot be added, instance names cannot be changed, and algorithms and variable tables in function block definitions cannot be changed.

SECTION 3
Specifications

This section provides specifications for reference when using function blocks, including specifications on function blocks, instances, and compatible PLCs, as well as usage precautions and guidelines.

3-1 Function Block Specifications

3-1-1 Function Block Specifications

Item	Description
Number of function block definitions	896 max. per CPU Unit
Number of instances	2,048 max. per CPU Unit
Number of instance nesting levels	Nesting is not supported.
Number of I/O variables	64 variables max. per function block definition

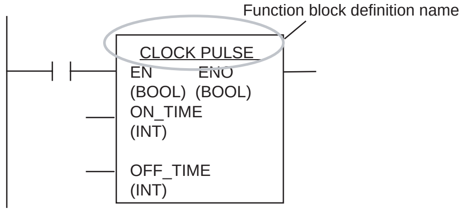
3-1-2 Function Block Elements

The following table shows the items that must be entered by the user when defining function blocks.

Item	Description
Function block definition name	The name of the function block definition
Language	The programming language used in the function block definition. Select ladder programming or structured text
Variable definitions	Variable settings, such as operands and return values, required when the function block is executed • Type (usage) of the variable • Name of the variable • Data type of the variable • Initial value of the variable
Algorithm	Enter the programming logic in ladder or structured text.
Comment	Function blocks can have comments.

Function Block Definition Name

Each function block definition has a name. The names can be up to 64 characters long and there are no prohibited characters. The default function block name is FunctionBlock□, where □ is a serial number.



Language

Select either ladder or structured text.

Variable Definitions

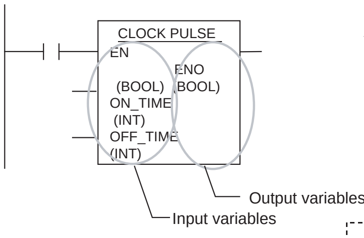
Define the operands and variables used in the function block definition.

Variable Names

- Variable names can be up to 30,000 characters long.
- Variables name cannot contain spaces or any of the following characters:
  ! “ # $ % & ‘ ( ) = - ~ ^ \ | ‘ @ { [+ ; * : }] < , > . ? /
- Variable names cannot start with a number (0 to 9).
- Variable names cannot contain two underscore characters in a row.

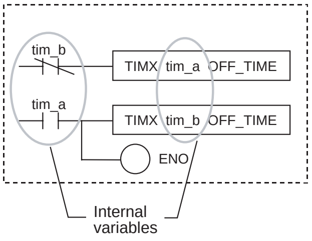
There are no other restrictions.

Variable Notation



Variable table

Usage	Name	Type
Internal	tim_a	TIMER
Internal	tim_b	TIMER
Input	ON_TIME	INT
Input	OFF_TIME	INT



Variable Type (Usage)

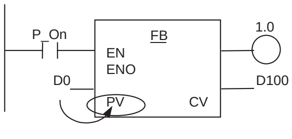
Item	Variable type			
	Inputs	**Outputs**	**Internals**	**Externals**
Definition	Operands to the instance	Return values from the instance	Variables used only within instance	Global symbols registered as variables beforehand with the CX-Programmer IEC
Status of value at next execution	The value is not passed on to the next execution.	The value is passed on to the next execution.	The value is passed on to the next execution.	The value is not passed on to the next execution.
Display	Displayed on the left side of the instance.	Displayed on the right side of the instance.	Not displayed.	Not displayed.
Number allowed	64 max. per function block (excluding EN)	64 max. per function block (excluding ENO)	Unlimited	Reserved variables only (28 total)
AT setting	No	No	Supported	No
Array setting	No	No	Supported	No
Retain setting	No	Supported	Supported	No
Variables created by default	EN (Enable): Receives an input condition.	ENO (Enable Output): Outputs the function block's execution status.	None	Global symbols registered in advance as variables in the CX-Programmer IEC, such as Condition Flags and some Auxiliary Area bits.

Note For details on Externals, refer to *Appendix C External Variables*.

■ Input Variables

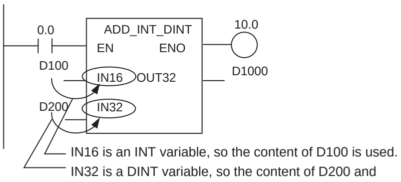
Input variables pass external operands to the instance. The input variables are displayed on the left side of the instance.

The value of the input source (data contained in the specified parameter just before the instance was called) will be passed to the input variable.

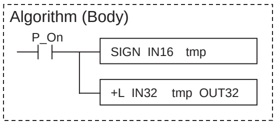


The value of the parameter specified as the input (value of D0) is passed to the instance's input variable (PV).

Example

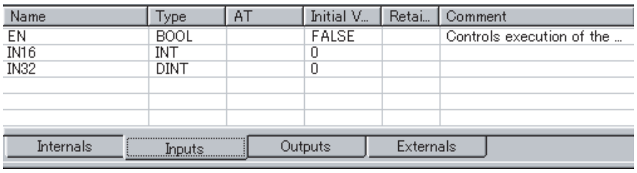


IN16 is an INT variable, so the content of D100 is used.
IN32 is a DINT variable, so the content of D200 and D201 is used.



Variable table

Usage	Name	Type
Internal	tmp	DINT
Input	EN	BOOL
Input	IN16	INT
Input	IN32	DINT
Output	ENO	BOOL
Output	OUT32	DINT



Name	Type	AT	Initial V...	Retai...	Comment
EN	BOOL		FALSE		Controls execution of the ...
IN16	INT		0		
IN32	DINT		0		

Note 1. The same name cannot be assigned to an input variable and output variable. If it is necessary to have the same variable as an input variable and output variable, register the variables with different names and transfer the value of the input variable to the output variable in the function block with an instruction such as MOV.

2. When the instance is executed, input values are passed from parameters to input variables before the algorithm is processed. Consequently, values cannot be read from parameters to input variables within the algorithm. If it is necessary to read a value within the execution cycle of the algorithm, do not pass the value from a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).

Initial Value

When you set an initial value for an input variable, that value will be written to the variable when the parameter for input variable EN goes ON and the instance is executed for the first time (and that one time only). If an initial value has not been set for an input variable, the input variable will be set to 0 when the instance is first executed.

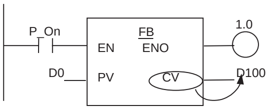
EN (Enable) Variable

When an input variable is created, the default input variable is the EN variable. The instance will be executed when the parameter for input variable EN is ON.

■ Output Variables

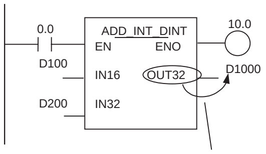
Output variables pass return values from the instance to external applications. The output variables are displayed on the right side of the instance.

After the instance is executed, the value of the output variable is passed to the specified parameter.

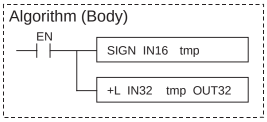


The value of the output variable (CV) is passed to the parameter specified as the output destination, which is D100 in this case.

Example

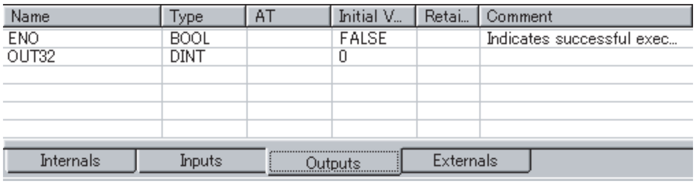


OUT32 is a DINT variable, so the variable's value is passed to D1000 and D1001.



Variable table

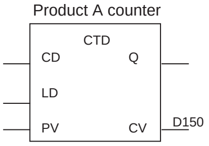
Usage	Name	Type
Internal	tmp	DINT
Input	EN	BOOL
Input	IN16	INT
Input	IN32	DINT
Output	ENO	BOOL
Output	OUT32	DINT



Name	Type	AT	Initial V...	Retai...	Comment
ENO	BOOL		FALSE		Indicates successful exec...
OUT32	DINT		0		

Like internal variables, the values of output variables are retained until the next time the instance is executed.

Example:
In the following example, the value of output variable CV will be retained until the next time the instance is executed.



Note

1. The same name cannot be assigned to an input variable and output variable. If it is necessary to have the same variable as an input variable and output variable, register the variables with different names and transfer the value of the input variable to the output variable in the function block with an instruction such as MOV.
2. When the instance is executed, output variables are passed to the corresponding parameters after the algorithm is processed. Consequently, values cannot be written from output variables to parameters within the algorithm. If it is necessary to write a value within the execution cycle of the algorithm, do not write the value to a parameter. Assign the value to an internal variable and use an AT setting (specified addresses).

Initial Value

An initial value can be set for an output variable that is not being retained, i.e., when the Retain Option is not selected. An initial value cannot be set for an output variable if the Retain Option is selected.
The initial value will not be written to the output variable if the IOM Hold Bit (A50012) is ON.

Auxiliary Area control bit		Initial value
IOM Hold Bit (A50012)	ON	The initial value will not be set.

ENO (Enable Output) Variable

The ENO variable is created as the default output variable. The ENO output variable will be turned ON when the instance is called. The user can change this value. The ENO output variable can be used as a flag to check whether or not instance execution has been completed normally.

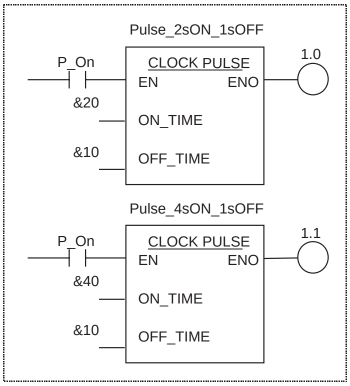
■ Internal Variables

Internal variables are used within an instance. These variables are internal to each instance. They cannot be referenced from outside of the instance and are not displayed in the instance.

The values of internal variables are retained until the next time the instance is executed. Consequently, even if instances of the same function block definition are executed with the same I/O parameters, the result will not necessarily be the same.

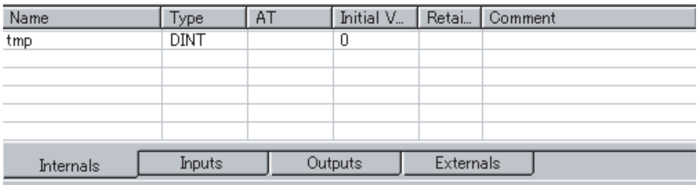
Example:

The internal variable tim_a in instance Pulse_2sON_1sOFF is different from internal variable tim_a in instance Pulse_4sON_1sOFF, so the instances cannot reference and will not affect each other's tim_a value.

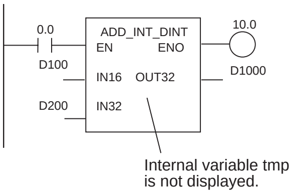


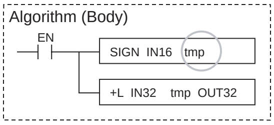
Variable table

Usage	Name	Type
Internal	tim_a	TIMER
Internal	tim_b	TIMER
Input	ON_TIME	INT
Input	OFF_TIME	INT



Name	Type	AT	Initial V...	Retai...	Comment
tmp	DINT		0		





Variable table

	Name	Type
Internal	tmp	DINT
Input	EN	BOOL
Input	IN16	INT
Input	IN32	DINT
Output	ENO	BOOL
Output	OUT32	DINT

Retain Data through Power Interruptions and Start of Operation

Internal variables retain the value from the last time that the instance was called. In addition, the Retain Option can be selected so that an internal variable will also retains its value when the power is interrupted or operation starts (the mode is switched from PROGRAM to RUN or MONITOR mode).

When the Retain Option is selected, the value of the variable is retained when the power is interrupted or operation starts unless the CPU Unit does not have a backup battery. If the CPU Unit does not have a good battery, the value will be unstable.

Variables	Condition	Status
Variables set to *Retain*	Start of operation	Retained
	Power ON	Retained

When the Retain Option is not selected, the value of the variable will not be held when the power is interrupted or operation starts. Even variables not set to be retained, however, can be held at the start of operation by turning ON the IOM Hold Bit (A50012) and can be held during power interruptions by setting the PLC Setup, as shown in the following table.

Variables	Condition	IOM Hold Bit (A50012) setting		
		OFF	ON	
			IOM Hold Bit Status at Startup (PLC Setup) selected	IOM Hold Bit Status at Startup (PLC Setup) not selected
Variables not set to *Retain*	Start of operation	Not retained	Retained	Retained
	Power ON	Not retained	Retained	Not retained

Note The IOM Hold Bit (A50012) is supported for compatibility with previous models. To hold the values of variables in function blocks, however, use the *Retain Option* and not the IOM Hold Bit.

Initial Value

An initial value can be set for an internal variable that is not being retained (i.e., when the Retain Option not selected). An initial value cannot be set for an internal variable if the Retain Option is selected.
Internal variables that are not being retained will be initialized to 0.
The initial value will not be written to the internal variable if the IOM Hold Bit (A50012) is ON.

Auxiliary Area control bit		Initial value
IOM Hold Bit (A50012)	ON	The initial value will not be set.
	OFF	The initial value will be set.

■ External Variables

External variables are global symbols registered as variables in advance with the CX-Programmer IEC. For details, refer to *Appendix C External Variables*.

Variable Properties

Variable Name

The variable name is used to identify the variable in the function block. The name can be up to 30,000 characters long. The same name can be used in other function blocks.

Note A variable name must be input for variables, even ones with AT settings (specified address).

Data Type

Any of the following types may be used.

Data type	Content	Size	Inputs	Outputs	Internals
BOOL	Bit data	1 bit	OK	OK	OK
INT	Integer	16 bits	OK	OK	OK
UNIT	Unsigned integer	16 bits	OK	OK	OK
DINT	Double integer	32 bits	OK	OK	OK
UDINT	Unsigned double integer	32 bits	OK	OK	OK
LINT	Long (8-byte) integer	64 bits	OK	OK	OK
ULINT	Unsigned long (8-byte) integer	64 bits	OK	OK	OK
WORD	16-bit data	16 bits	OK	OK	OK
DWORD	32-bit data	32 bits	OK	OK	OK
LWORD	64-bit data	64 bits	OK	OK	OK
REAL	Real number	32 bits	OK	OK	OK
LREAL	Long real number	64 bits	OK	OK	OK
TIMER	Timer (See note.)	Flag: 1 bit PV: 16 bits	OK	OK	OK
COUNTER	Counter (See note.)	Flag: 1 bit PV: 16 bits	OK	OK	OK

Note The TIMER and COUNTER data types cannot be used in ST language function blocks.

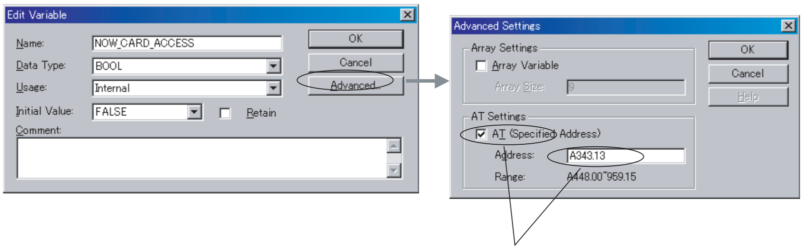
AT Settings (Allocation to Actual Addresses)

With internal variables, it is possible to set the variable to a particular I/O memory address rather than having it allocated automatically by the system. To specify a particular address, the user can input the desired I/O memory address in this property. It is still necessary to use variable name in programming even if a particular address is specified.

Note The AT property can be set for internal variables only.

Example:
If the READ DATA FILE instruction (FREAD) is being used in the function block definition and it is necessary to check the File Memory Operation Flag (A34313), use an internal variable and specify the flag's address in the AT setting.

Register an internal variable, select the AT setting option, and specify A34313 as the address. The status of the File Memory Operation Flag can be checked through this internal variable.



Edit Variable

Name: NOW_CARD_ACCESS

Data Type: BOOL

Usage: Internal

Initial Value: FALSE

□ Retain

Comment:

Advanced Settings

Array Settings

□ Array Variable

Array Size: 9

AT Settings

☑ AT (Specified Address)

Address: A343.13

Range: A448.00~959.15

Address A34313 is allocated to a boolean internal variable named NOW_CARD_ACCESS.

When the AT setting is used, the function block loses its flexibility. This function should thus be used only when necessary.

Array Setting

With internal variables, a variable can be defined as an array.

Note Only one-dimensional arrays are supported by the CX-Programmer IEC.

With the array setting, a large number of variables with the same properties can be used by registering just one variable.

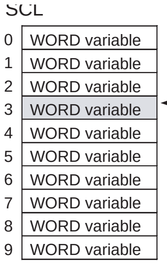
- An array can have from 1 to 32,000 array elements.
- The array setting can be set for internal variables only.
- Any data type can be specified for an array variable, as long as it is an internal variable.
- When entering an array variable name in the algorithm of a function block definition, enter the array index number in square brackets after the variable name. The following three methods can be used to specify the index. (In this case the array variable is a[].)
  - Directly with numbers (for ladder or ST language programming)
    Example: a[2]
  - With a variable (for ladder or ST language programming)
    Example: a[n], where n is a variable
  - With an equation (for ST language programming only)
    Example: a[b+c], where b and c are variables

  Note Equations can contain only arithmetic operators (+, –, *, and /).

An array is a collection of data elements that are the same type of data. Each array element is specified with the same variable name and a unique index. (The index indicates the location of the element in the array.)

A one-dimensional array is an array with just one index number.

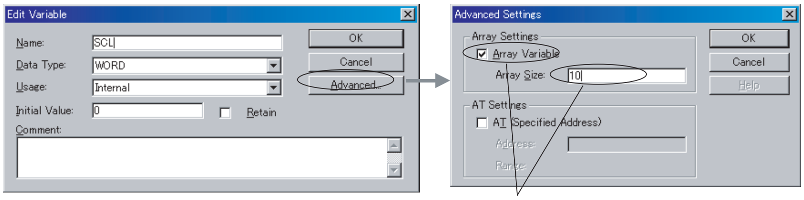
Example: When an internal variable named SCL is set as an array variable with 10 elements, the following 10 variables can be used:
SCL[0], SCL[1], SCL[2], SCL[3], SCL[4], SCL[5], SCL[6], SCL[7], SCL[8], and SCL[9]



SCL

0	WORD variable
1	WORD variable
2	WORD variable
3	WORD variable
4	WORD variable
5	WORD variable
6	WORD variable
7	WORD variable
8	WORD variable
9	WORD variable

Specify SCL[3] to access this data element.



Edit Variable

Name: SCL
Data Type: WORD
Usage: Internal
Initial Value: 0
□ Retain
Comment:

Advanced Settings

Array Settings
☑ Array Variable
Array Size: 10

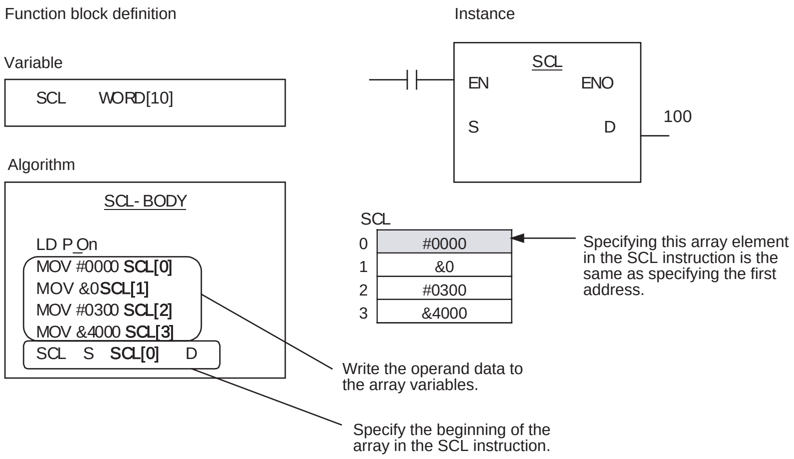
AT Settings
□ AT (Specified Address)
Address:
Range:

Settings for variable SCL as an array variable with element numbers 0 to 9.

Note When specifying the first or last word of multiple words for an instruction operand, I/O parameters cannot be used to pass data to or from I/O variables. Internal array variables must be used. This applies, for example, to the first source word for SEND(090) or the starting word or end word for BSET(071).

For multiword operands, an array variable must be prepared in advance with the required number of elements and the data must be set for the array in the function block definition. The first or last element in the array variable is then specified for the operand to set the first or last word. Refer to *3-4 Function Block Applications Guidelines* for details.

Example:



SCL	WORD[10]

```
LD P_On
MOV #0000 SCL[0]
MOV &0 SCL[1]
MOV #0300 SCL[2]
MOV &4000 SCL[3]
SCL   S   SCL[0]   D
```

SCL	
0	#0000
1	&0
2	#0300
3	&4000

Specifying this array element in the SCL instruction is the same as specifying the first address.

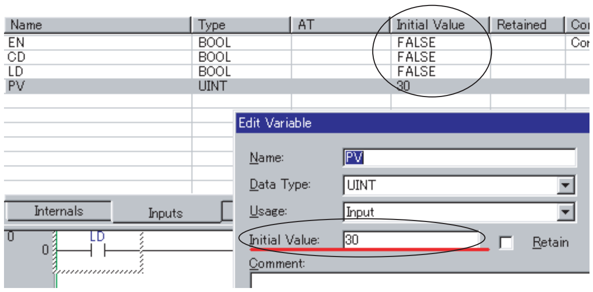
Write the operand data to the array variables.

Specify the beginning of the array in the SCL instruction.

Note For details, refer to *3-4 Function Block Applications Guidelines*.

Initial Values

When an instance is executed the first time, initial values can be set for input variables, internal variables, and output variables. For details, refer to *Initial Value* under the preceding descriptions of input variables, internal variables, and output variables.



Name	Type	AT	Initial Value	Retained	Co
EN	BOOL		FALSE		Co
CD	BOOL		FALSE		
LD	BOOL		FALSE		
PV	UINT		30		

Edit Variable

Name: PV

Data Type: UINT

Usage: Input

Initial Value: 30 □ Retain

Comment:

Retaining Data through Power Interruptions and Start of Operation

The values of internal variables can be retained through power interruptions and the start of operation. When the Retain Option is selected, the variable will be allocated to a region of memory that is retained when the power is interrupted and PLC operation starts.

Algorithm

Enter the logic programming using the registered variables.

Comment

A comment up to 30,000 characters long can be entered.

3-2 Instance Specifications

3-2-1 Composition of an Instance

The following table lists the items that the user must set when registering an instance.

Item	Description
Instance name	Name of the instance
Language Variable definitions	The programming and variables are the same as in the function block definition.
Function block instance areas	The ranges of addresses used by the variables
Comments	A comment can be entered for each instance.

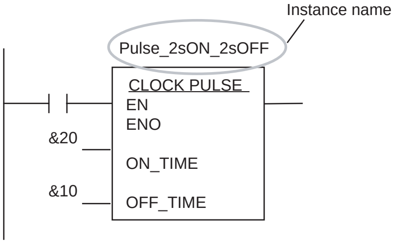
Instance Name

This is the name of the instance.

- Instance names can be up to 30,000 characters long.
- Instance names cannot contain spaces or any of the following characters: ! “ # $ % & ‘ ( ) = - ~ ^ \ | ‘ @ { [+ ; * : }] < , > . ? /
- Instance names cannot start with a number (0 to 9).
- Instance names cannot contain two underscore characters in a row.

There are no other restrictions.

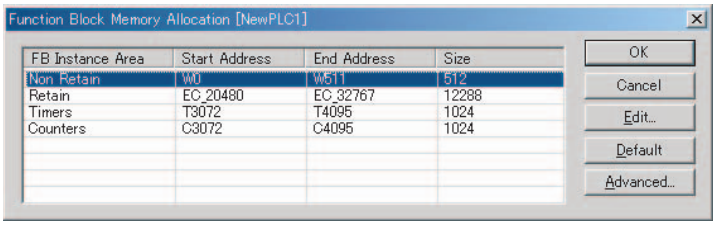
The instance name is displayed above the instance in the diagram.



Function Block Instance Areas

To use a function block, the system requires memory to store the instance’s internal variables and I/O variables. These areas are known as the function block instance areas and the user must specify the first addresses and sizes of these areas. The first addresses and area sizes can be specified in 1-word units.

When the CX-Programmer IEC compiles the function, it will output an error if there are any instructions in the user program that access words in these areas.


Function Block Memory Allocation [NewPLC1]

FB Instance Area	Start Address	End Address	Size
Non Retain	W0	W511	512
Retain	EC_20480	EC_32767	12288
Timers	T3072	T4095	1024
Counters	C3072	C4095	1024

The default values are as follows:

FB instance area	Start address	End address	Applicable memory areas
Non-retained area	W0	512	CIO, WR, HR, DM, EM
Retained area	E0_20480 in last EM Area bank	12,288	HR, DM, EM

FB instance area	Start address	End address	Applicable memory areas
Timer area	T3072	1,024	TIM
Counter area	C3072	1,024	CNT

Comments

A comment up to 30,000 characters long can be entered.

Creating Multiple Instances

Calling the Same Instance

A single instance can be called from multiple locations. In this case, the internal variables will be shared.

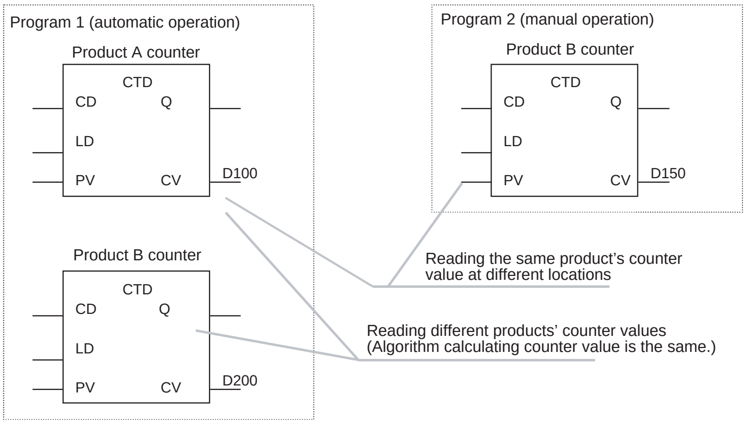
Making Multiple Instances

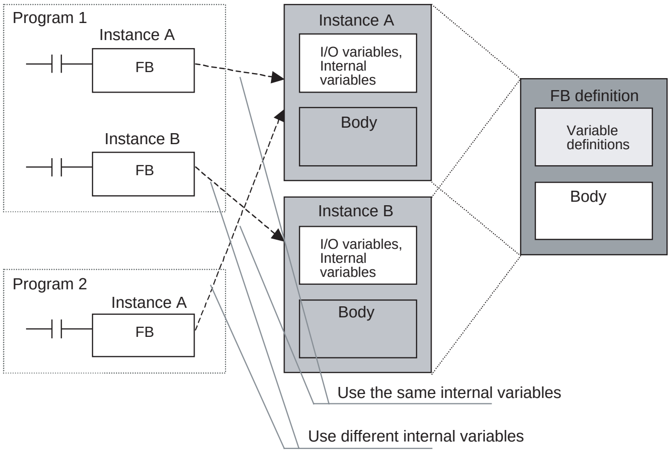
Multiple instances can be created from a single function block definition. In this case, the values of internal variables will be different in each instance.

Example: Counting Product A and Product B

Prepare a function block definition called Down Counter (CTD) and set up counters for product A and product B. There are two types of programs, one for automatic operation and another for manual operation. The user can switch to the appropriate mode of operation.

In this case, multiple instances will be created from a single function block. The same instance must be called from multiple locations.

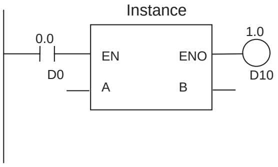

Program 1 (automatic operation)
Product A counter
CTD
CD
Q
LD
PV
CV
D100
Product B counter
CTD
CD
Q
LD
PV
CV
D200
Program 2 (manual operation)
Product B counter
CTD
CD
Q
LD
PV
CV
D150
Reading the same product's counter value at different locations
Reading different products' counter values (Algorithm calculating counter value is the same.)


Program 1
Instance A
FB
Instance B
FB
Program 2
Instance A
FB
Instance A
I/O variables, Internal variables
Body
Instance B
I/O variables, Internal variables
Body
FB definition
Variable definitions
Body
Use the same internal variables
Use different internal variables

3-2-2 Operating Specifications

Calling Instances

The user can call an instance from any location. The instance will be executed when the input to EN is ON.

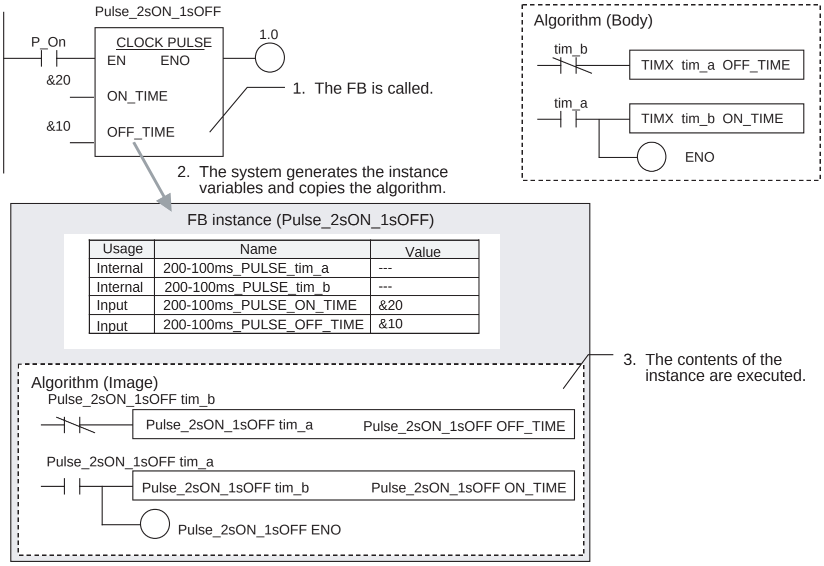


In this case, the input to EN is bit 0.0 at the left of the diagram.

- When the input to EN is ON, the instance is executed and the execution results are reflected in bit 1.0 and word D10.
- When the input to EN is OFF, the instance is not executed, bit 1.0 is turned OFF, and the content of D10 is not changed.

Operation when the Instance Is Executed

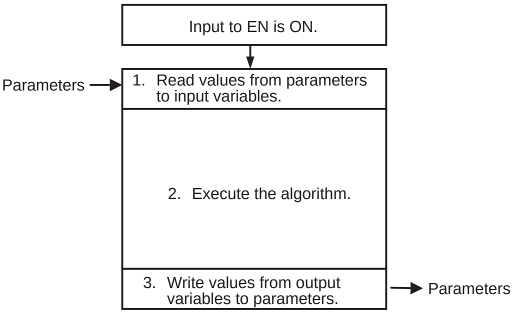
The system calls a function block when the input to the function block's EN input variable is ON. When the function block is called, the system generates the instance's variables and copies the algorithm registered in the function block. The instance is then executed.



Usage	Name	Value
Internal	200-100ms_PULSE_tim_a	---
Internal	200-100ms_PULSE_tim_b	---
Input	200-100ms_PULSE_ON_TIME	&20
Input	200-100ms_PULSE_OFF_TIME	&10

The order of execution is as follows:

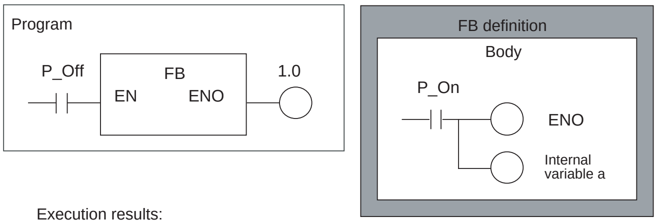
1. Read data from parameters to input variables.
2. Execute the algorithm.
3. Write data from output variables to parameters.


Input to EN is ON.
Parameters
1. Read values from parameters to input variables.
2. Execute the algorithm.
3. Write values from output variables to parameters.
Parameters

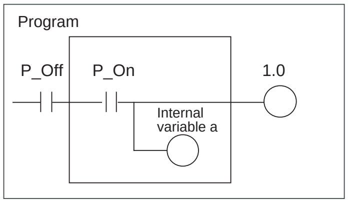
Note Data cannot be exchanged with parameters in the algorithm itself.
In addition, if an output variable is not changed by the execution of the algorithm, the output parameter will retain its previous value.

Operation when the Instance Is Not Executed

When the input to the function block's EN input variable is OFF, the function block is not called, so the internal variables of the instance do not change.



Execution results:
Output variable 1.0 is turned OFF, but
internal variable a retains its previous value.

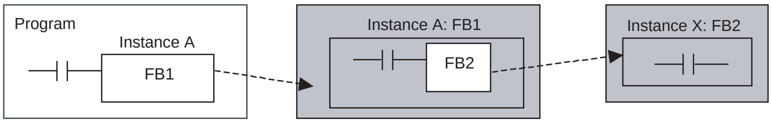


If the programming were entered directly into the program instead of in a function block definition, both bit 1.0 and variable a would be turned OFF.

⚠ **Caution** An instance will not be executed while its EN input variable is OFF, so Differentiation and Timer instructions will not be initialized while EN is OFF. If Differentiation or Timer instructions are being used, use the Always ON Flag (P_On) for the EN input condition and include the instruction's input condition within the function block definition.

Nesting

A function block cannot be called from another function block, i.e., nesting is not supported.



3-3 Restrictions on Function Blocks

Ladder Programming Restrictions

There are some restrictions on instructions used in ladder programs.

Restrictions in Program (Outside of Instances)

Subroutine Instructions (SBS, GSBS, RET, MCRO, and SBN):

Subroutine numbers 128 to 1,023 cannot be used. Only 0 to 127 can be used.

Instructions Prohibited in Function Block Definitions

The following instructions cannot be used in function block definitions. A compile error will occur if any of these instructions is used.

- Block Programming Instructions (BPRG and BEND)
- Subroutine Instructions (SBS, GSBS, RET, MCRO, and SBN)
- Jump Instructions (JMP, CJP, CJPN, JMP0, and JME0)
- Step Instructions (STEP and SNXT)
- Immediate Refresh Instructions (!)
- I/O REFRESH Instruction (IORF)
- TMHH and TIMH Instructions
- CV Address Conversion Instructions (FRMCV and TOCV)

- Instructions manipulating record positions (PUSH, FIFO, LIFO, SETR, and GETR)
- FAILURE POINT DETECTION Instruction (FPD)
- Index Register Read Instructions (MOVR and MOVRW)

AT Setting Restrictions (Unsupported Data Areas)

Addresses in the following areas cannot be used for AT settings.

- Index Registers and Data Registers (Neither indirect nor direct addressing is supported.)
- Indirect addressing of DM or EM Area addresses (Neither binary-mode nor BCD-mode indirect addressing is supported.)

I/O Variable Restrictions (Unsupported Data Areas)

Addresses in the following data areas cannot be used as parameters for input and output variables.

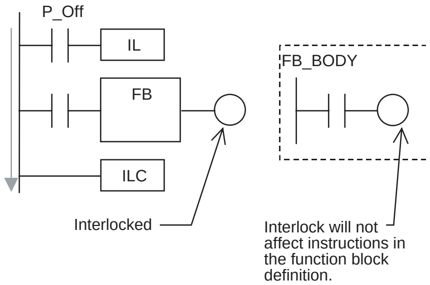
- Index Registers and Data Registers (Neither indirect nor direct addressing is supported.)
- Indirect addressing of DM or EM Area addresses (Neither binary-mode nor BCD-mode indirect addressing is supported.)

Refreshing Timer and Counter PVs

Timer and counter PVs are always stored in binary mode, so PVs of all Timer and Counter Instructions must be treated as binary data whether or not the instructions are in function blocks.

Interlocks

When a function block is called from an interlocked program section, the contents of the function block definition will not be executed. The interlocked function block will behave just like an interlocked subroutine.

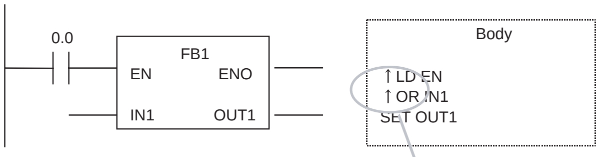

P_Off
IL
FB
ILC
FB_BODY
Interlocked
Interlock will not affect instructions in the function block definition.

Differentiation Instructions in Function Block Definitions

An instance will not be executed while its EN input variable is OFF, so the following precautions are essential when using a Differentiation Instruction in a function block definition. (Differentiation Instructions include DIFU, DIFD, and any instruction with an @ or % prefix.)

- As long as the instance's EN input variable is OFF, the execution condition will retain its previous status (the last status when the EN input variable was ON) and the Differentiation Instruction will not operate.
- When the instance's EN input variable goes ON, the present execution condition status will not be compared to the last cycle's status. The present execution condition will be compared to the last condition when the EN input variable was ON, so the Differentiation Instruction will not operate properly. (If the EN input variable remains ON, the Differentiation Instruction will operate properly when the next rising edge or falling edge occurs.)

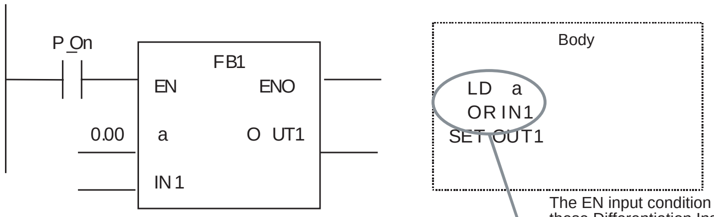
Example:



These Differentiation Instructions do not operate when input condition 0.00 goes from OFF to ON the first time.

The instructions do not operate while input condition 0.00 is OFF.

If Differentiation Instructions are being used, always use the Always ON Flag (P_On) for the EN input condition and include the instruction's input condition within the function block definition.

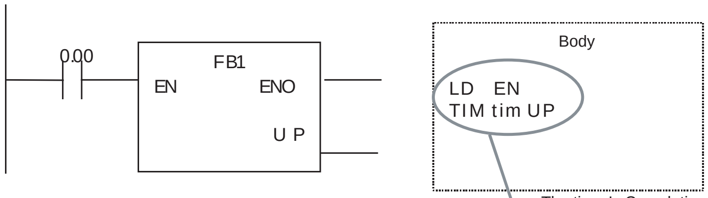


The EN input condition is always ON, so these Differentiation Instructions operate normally.

Timer Instructions in Function Block Definitions

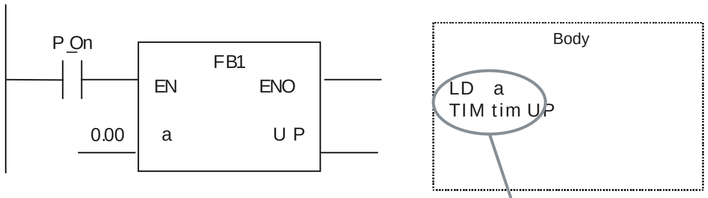
An instance will not be executed while its EN input variable is OFF, so the following precautions are essential when using a Timer Instruction in a function block definition.

The Timer Instruction will not be initialized even though the instance's EN input variable goes OFF. Consequently, the timer's Completion Flag will not be turned OFF if the EN input variable goes OFF after the timer started operating.



The timer's Completion Flag (UP) will not be turned OFF even though input condition 0.00 goes OFF.

If Timer Instructions are being used, always use the Always ON Flag (P_On) for the EN input condition and include the instruction's input condition within the function block definition.



The timer's completion flag (UP) is turned OFF when input condition a (0.00) goes OFF.

- If the same instance containing a timer is used in multiple locations at the same time, the timer will be duplicated.

ST Programming Restrictions

- Only the following statements and operators are supported.
  - Assignment statements
  - Selection statements (CASE and IF statements)
  - Iteration statements (FOR, WHILE, and REPEAT statements)
  - Arithmetic operators
  - Logical operators
  - Comparison operators
  - Comments
- The TIMER and COUNTER data types cannot be used.
- Use parentheses to indicate the priority of arithmetic operations.
  Example: D:= (A+B) *C
- Tabs and spaces can be used to indent text.

EM Current Bank

The EM current bank function cannot be used. The EM bank number must be specified in all EM Area addresses.

Online Editing Restrictions

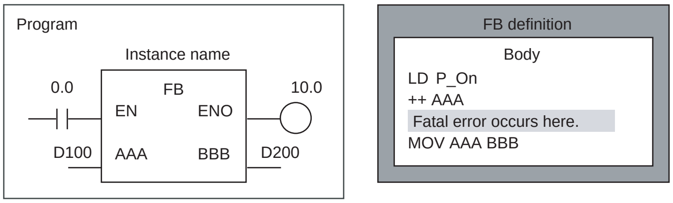
The following online editing operations cannot be performed on the user program in the CPU Unit.

- Changing or deleting function block definitions (variable table or algorithm)
- Inserting instances or changing instance names

  Note The instance's I/O parameters can be changed, instances can be deleted, and instructions outside of an instance can be changed.

Error-related Restrictions

If a fatal error occurs in the CPU Unit while a function block definition is being executed, ladder program execution will stop at the point where the error occurred.



```
LD P_On
++ AAA
Fatal error occurs here.
MOV AAA BBB
```

In this case, the MOV AAA BBB instruction will not be executed and output variable D200 will retain the same value that it had before the function block was executed.

Programming Console Displays

When a user program created in the CX-Programmer IEC is downloaded to the CPU Unit and read by a Programming Console, the instances will all be displayed as question marks. (The instance names will not be displayed.)

Prohibiting Access to FB Instance Areas

To use a function block, the system requires memory areas to store the instance's internal variables and I/O variables.

FB instance area	Initial value of Start Address	Initial value of Size	Allowed data areas
Non-retained	W0	512	CIO, WR, HR, DM, EM
Retained	E20480 in last EM bank	12,288	HR, DM, EM
Timer	T3072	1,024	TIM
Counter	C3072	1,024	CNT

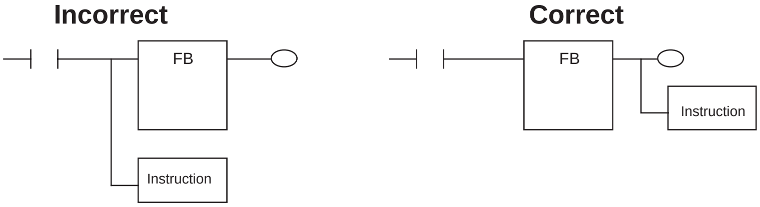
If there is an instruction in the user program that accesses an address in an FB instance area, the CX-Programmer IEC will output an error in the following cases.

- When a program check is performed by the user by selecting ***Program - Compile*** or ***Compile All Programs*** from the *PC* Menu.
- When attempting to download the user program to the PLC or attempting to write the program through online editing. (Neither downloading or editing will be possible.)

Program Structure Precautions

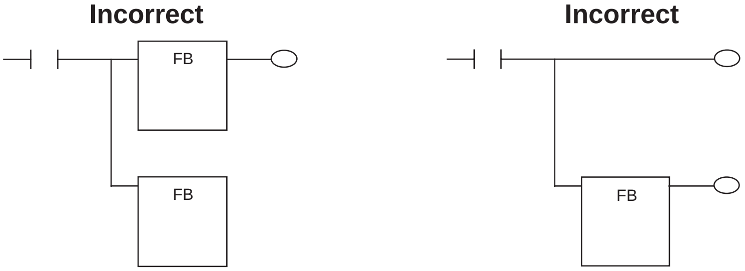
No Branches to the Left of the Instance

Branches are not allowed on the left side of the instance. Branches are allowed on the right side.



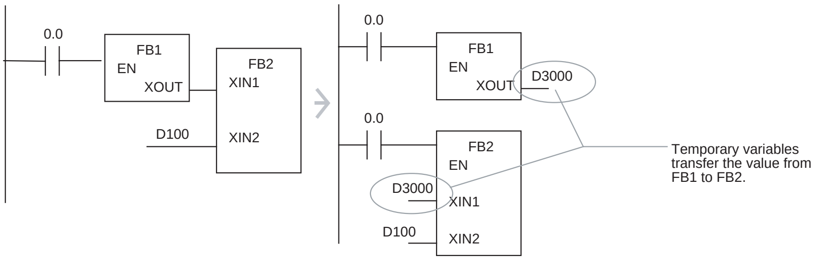
Only One Instance per Rung

A program rung cannot have more than one instance.



No Function Block Connections

A function block's input cannot be connected to another function block's output. In this case, a variable must be registered to transfer the execution status from the first function block's output to the second function blocks input.



Uploading Restriction

Programs cannot be uploaded from the CPU Unit to the CX-Programmer IEC.

PT Ladder Monitoring Restriction

The Programmable Terminal ladder monitoring function cannot be used with the CS1-H (FB)/CJ1-H (FB).

3-4 Function Block Applications Guidelines

This section provides guidelines for using function blocks with the CX-Programmer IEC.

3-4-1 Deciding on Variable Data Types

Integer Data Types (1, 2, or 4-word Data)

Use the following data types when handling single numbers in 1, 2, or 4-word units.

- INT and UINT
- DINT and DINT
- LINT and ULINT

Note Use signed integers if the numbers being used will fit in the range.

Word Data Types (1, 2, or 4-word Data)

Use the following data types when handling groups of data (non-numeric data) in 1, 2, or 4-word units.

- WORD
- DWORD
- LWORD

3-4-2 Array Settings

Array Variables Use for First or End Addresses of Word Ranges

When specifying an instruction operand that is the first address or end address of a range of words (see note), the required values cannot be passed to variables through input parameters or output parameters.

Note Refer to *Appendix D Instruction Support and Operand Restrictions* to determine which instruction operands must have array variables because they specify the first/end address of a range of words.

In this case, prepare an array variable with the required number of array elements, set the data in each array element in the function block, and specify the beginning (or end) array variable in the operand. Using an array variable allows you to specify the first address or end address of a range of words.

Handling a Single String of Data in Multiple Words

In this example, an array contains the directory and filename (operand S2) for an FREAD instruction.

- Variable table
  Internal variable, data type = WORD, array setting with 10 elements, variable names = filename[0] to filename[9]
- Ladder programming

```
MOV #5C31 file_name[0] ⎫
MOV #3233 file_name[1] ⎬← Set data in each array element.
MOV #0000 file_name[2])⎭
FREAD (omitted) (omitted) file_name[0] (omitted)← Specify the first element
                                                  of the array in the instruction
                                                  operand.
```

Handling Control Data in Multiple Words

In this example, an array contains the number of words and first source word (operand S1) for an FREAD instruction.

- Variable table
  Internal variable, data type = DINT, array setting with 3 elements, variable names = read_num[0] to read_num[9]
- Ladder programming

```
MOVL &100 read_num[0] (No._of_words)     }<— Set data in each array element.
MOVL &0 read_num[1] (1st_source_word)    }
FREAD (omitted) (omitted) file_name[0] (omitted) <— Specify the first element of the array
                                                    in the instruction operand.
```

Handling a Block of Read Data in Multiple Words

The allowed amount of read data must be determined in advance and an array must be prepared that can handle the maximum amount of data. In this example, an array receives the FREAD instruction's read data (operand D).

- Variable table
  Internal variable, data type = WORD, array setting with 100 elements, variable names = read_data[0] to read_data[99]
- Ladder programming

```
FREAD (omitted) (omitted) (omitted) read_data[0]
```

Division Using Integer Array Variables (Ladder Programming Only)

A two element array can be used to store the result from a ladder program's SIGNED BINARY DIVIDE (/) instruction. The result from the instruction is D (quotient) and D+1 (remainder). This method can be used to obtain the remainder from a division operation in ladder programming.

Note When ST language is used, it isn't necessary to use an array to receive the result of a division operation. Also, the remainder can't be calculated directly in ST language. The remainder must be calculated as follows:
Remainder = Dividend – (Divisor × Quotient)

3-4-3 AT Settings

Use the AT setting in the following cases.

- When setting the first destination word at the remote node for SEND(090) and the first source word at the remote node for RECV(098)
- When you want to read or write an Auxiliary Area bit within the execution cycle of an algorithm and the bit is not registered as an external variable. (If it isn't necessary to read or write the bit in the same cycle, use an I/O variable and I/O parameter.)

3-5 CPU Unit Specifications and Battery Replacement

The specifications of the CS1-H (FB)/CJ1-H (FB) CPU Units and the battery replacement procedure are given in this section. Refer to the *CS Series PLC Operation Manual* or the *CS Series PLC Operation Manual* for other items.

3-5-1 Specifications

CPU Unit Specifications

CS1-H (FB) CPU Units

CPU	CS1H-CPU67H (FB)	CS1H-CPU65H (FB)	CS1G-CPU44H (FB)	CS1G-CPU42H (FB)
I/O bits	5120		1280	960
User program memory (steps) (See note.)	250K	60K	30K	10K
Data memory	32K words			
Extended data memory	32K words x 13 banks E0_00000 to E6_32767	32K words x 3 banks E0_00000 to E2_32767	32K words x 1 bank E0_00000 to E2_32767	
Current consumption	0.82 A at 5 V DC		0.78 A at 5 V DC	

Note The number of steps in a program is not the same as the number of instructions. Some instructions require only 1 step, whereas others required 7 steps. (For example, LD and OUT require 1 step each, but MOV(021) requires 3 steps.) The program capacity indicates the total number of steps for all instructions in the program. Refer to *10-5 Instruction Execution Times and Number of Steps* in the *Operation Manual* for the number of steps required for each instruction.

CJ1-H (FB) CPU Units

CPU	CJ1G-CPU44H (FB)	CJ1G-CPU43H (FB)	CJ1G-CPU42H (FB)
I/O bits	1,280	960	
User program memory (steps) (See note.)	30 K	20 K	10 K
Data Memory	32 Kwords		
Extended Data Memory	32Kwords x 1 bank E0_00000 to E0_32767		
Current consumption	0.91 A at 5 V DC		

Common Specifications

Item	Specification	Reference
Control method	Stored program	---
I/O control method	Cyclic scan and immediate processing are both possible.	---
Programming	Ladder diagram	---
CPU processing mode	Normal Mode, Parallel Processing Mode with Asynchronous Memory Access, Parallel Processing Mode with Synchronous Memory Access, or Peripheral Servicing Priority Mode	---

Item	Specification	Reference
Instruction length	1 to 7 steps per instruction	Steps and number of steps per instruction: *10-5 Instruction Execution Times and Number of Steps* in *Operation Manual*
Ladder instructions	Approx. 400 different instructions (3-digit function codes) The following instructions cannot be used in function block definitions. • Block programming instructions (BPRG and BEND) • Subroutine instructions (SBS, GSBS, RET, MCRO, and SBN) • Jump instructions (JMP, CJP, and CJPN) • Step ladder instructions (STEP and SNXT) • Immediate refresh instructions (!) • I/O REFRESH (IORF) • ONE-MS TIMER (TMHH) and HIGH-SPEED TIMER (TIMH)	---
Execution time	Basic instructions: 0.02 μs min. Special instructions: 0.06 μs min.	Instruction execution times: *10-5 Instruction Execution Times and Number of Steps* in *Operation Manual*
Overhead processing time	Normal mode: 0.3 ms min. Parallel processing: 0.3 ms min.	---
Number of Expansion Racks	CS1-H (FB) CPU Unit: 7 Racks max. (C200H Expansion I/O Racks: 3 max.) CJ1-H (FB) CPU Unit: 3 Racks max.	Expansion Racks
Number of tasks	288 (cyclic tasks: 32, interrupt tasks: 256) Interrupt tasks can be executed every cycle the same as cycle cyclic tasks and are called “extra cyclic tasks” when they are used this way.If extra cyclic tasks are used, up to 288 cyclic tasks can be executed. **Note** Cyclic tasks are executed each cycle and are controlled with TKON(820) and TKOF(821) instructions. **Note** The following 4 types of interrupt tasks are supported. Power OFF interrupt tasks: 1 max. Scheduled interrupt tasks: 2 max. I/O interrupt tasks: 32 max. External interrupt tasks: 256 max.	Tasks: *Programming Manual* (W394)
Interrupt types	Scheduled Interrupts: Interrupts generated at a time scheduled by the CPU Unit’s built-in timer. I/O Interrupts: Interrupts from Interrupt Input Units. Power OFF Interrupts: Interrupts executed when the CPU Unit’s power is turned OFF. External I/O Interrupts: Interrupts from the Special I/O Units, CS-series CPU Bus Units, or the Inner Board (CS1-H (FB) only).	
Calling subroutines from more than one task	Supported using global subroutines.	

Item			**Specification**	**Reference**	
CIO (Core I/O) Area	**I/O Area**		5,120: CIO 000000 to CIO 031915 (320 words from CIO 0000 to CIO 0319) The setting of the first word can be changed from the default (CIO 0000) so that CIO 0000 to CIO 0999 can be used. I/O bits are allocated to Basic I/O Units, such as CS-series Basic I/O Units, C200H Basic I/O Units, and C200H Group-2 High-density I/O Units.	Input and output bits: *9-4 CIO Area* in *Operation Manual*	The CIO Area can be used as work bits if the bits are not used as shown here.
		C200H DeviceNet Area	1,600 (100 words): Outputs: CIO 005000 to CIO 009915 (words CIO 0050 to CIO 0099) Inputs: CIO 035000 to CIO 039915 (words CIO 0350 to CIO 0399) C200H DeviceNet Area bits are allocated to Slaves according to C200HW-CRW21-V1 DeviceNet Unit remote I/O communications.	*9-5 C200H DeviceNet Area* in *Operation Manual*	
		PLC Link Area (CS1-H (FB) only)	80 bits (5 words): CIO 024700 to CIO 025015 (words CIO 0247 to CIO 0250 and CIO A442) When a PLC Link Unit is used in a PLC Link, use these bits to monitor PLC Link errors and the operating status of other CPU Units in the PLC Link.	*9-7 PLC Link Area* in *Operation Manual*	
	Link Area		3,200 (200 words): CIO 10000 to CIO 119915 (words CIO 1000 to CIO 1199) Link bits are used for data links and are allocated to Units in Controller Link Systems and PLC Link Systems (CS1-H (FB) only).	*9-8 Data Link Area* in *Operation Manual*	
	CPU Bus Unit Area		6,400 (400 words): CIO 150000 to CIO 189915 (words CIO 1500 to CIO 1899) CS-series CPU Bus Unit bits store the operating status of CS-series CPU Bus Units. (25 words per Unit, 16 Units max.)	*9-9 CPU Bus Unit Area* in *Operation Manual*	
	Special I/O Unit Area		15,360 (960 words): CIO 200000 to CIO 295915 (words CIO 2000 to CIO 2959) Special I/O Unit bits are allocated to CS-series Special I/O Units and C200H Special I/O Units. (See Note.) (10 words per Unit, 96 Units max.) **Note** For the CS1-H (FB), there are I/O Units that are treated as Special I/O Units. Examples: C200H-ID215/0D215/MD215	*9-11 Special I/O Unit Area* in *Operation Manual*	
	Inner Board Area (CS1-H (FB) only)		1,600 (100 words): CIO 190000 to CIO 199915 (words CIO 1900 to CIO 1999) Inner Board bits are allocated to Inner Boards. (100 I/O words max.)	*9-10 Inner Board Area* in *Operation Manual*	
	SYSMAC BUS Area (CS1-H (FB) only)		800 (50 words): CIO 300000 to CIO 304915 (words CIO 3000 to CIO 3049) SYSMAC BUS bits are allocated to Slave Racks connected to SYSMAC BUS Remote I/O Master Units. (10 words per Rack, 5 Racks max.)	*9-12 SYSMAC BUS Area* in *Operation Manual*	
	I/O Terminal Area (CS1-H (FB) only)		512 (32 words): CIO 310000 to CIO 313115 (words CIO 3100 to CIO 3131) I/O Terminal bits are allocated to I/O Terminal Units (but not to Slave Racks) connected to SYSMAC BUS Remote I/O Master Units. (1 word per Terminal, 32 Terminals max.)	*9-13 I/O Terminal Area* in *Operation Manual*	

<table>
<tr><th colspan="2">Item</th><th>Specification</th><th>Reference</th></tr>
<tr><td><b>CIO (Core I/O) Area, contin-ued</b></td><td><b>CS-series DeviceNet Area</b></td><td>9,600 (600 words): CIO 320000 to CIO 379915 (words CIO 3200 to CIO 3799)
CS-series DeviceNet Area bits are allocated to Slaves according to C200HW-CRW21-V1 DeviceNet Unit remote I/O communications.
<table>
<tr><td>Fixed allocation 1</td><td>Outputs: CIO 3200 to CIO 3263
Inputs: CIO 3300 to CIO 3363</td></tr>
<tr><td>Fixed allocation 2</td><td>Outputs: CIO 3400 to CIO 3463
Inputs: CIO 3500 to CIO 3563</td></tr>
<tr><td>Fixed allocation 3</td><td>Outputs: CIO 3600 to CIO 3663
Inputs: CIO 3700 to CIO 3763</td></tr>
</table>
The following words are allocated to the CS-Series DeviceNet Unit functioning as a master when fixed allocations are used for the CS1W-DRM21 DeviceNet Unit.
<table>
<tr><th>Setting</th><th>Master to slave</th><th>Slave to master</th></tr>
<tr><td>Fixed allocation 1</td><td>Outputs: CIO 3370</td><td>Inputs: CIO 3270</td></tr>
<tr><td>Fixed allocation 2</td><td>Outputs: CIO 3570</td><td>Inputs: CIO 3470</td></tr>
<tr><td>Fixed allocation 3</td><td>Outputs: CIO 3770</td><td>Inputs: CIO 3670</td></tr>
</table></td><td><i>9-6 CS-series DeviceNet Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Internal I/O Area</b></td><td>4,800 (300 words): CIO 120000 to CIO 149915 (words CIO 1200 to CIO 1499)
37,504 (2,344 words): CIO 380000 to CIO 614315 (words CIO 3800 to CIO 6143)
These bits in the CIO Area are used as work bits in programming to control program execution. They cannot be used for external I/O.</td><td>---</td></tr>
<tr><td colspan="2"><b>Work Area</b></td><td>8,192 bits (512 words): W00000 to W51115 (W000 to W511)
Controls the programs only. (I/O from external I/O terminals is not possible.)
<b>Note</b> When using work bits in programming, use the bits in the Work Area first before using bits from other areas.</td><td><i>9-14 Work Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Holding Area</b></td><td>8,192 bits (512 words): H00000 to H51115 (H000 to H511)
Holding bits are used to control the execution of the program, and maintain their ON/OFF status when the PLC is turned OFF or the operating mode is changed.</td><td><i>9-15 Holding Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Auxiliary Area</b></td><td>Read only: 7,168 bits (448 words): A00000 to A44715 (words A000 to A447)
Read/write: 8,192 bits (512 words): A44800 to A95915 (words A448 to A959)
Auxiliary bits are allocated specific functions.</td><td><i>9-16 Auxiliary Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Temporary Area</b></td><td>16 bits (TR0 to TR15)
Temporary bits are used to temporarily store the ON/OFF execution conditions at program branches.</td><td><i>9-17 TR (Temporary Relay) Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Timer Area</b></td><td>4,096: T0000 to T4095 (used for timers only)</td><td><i>9-18 Timer Area</i> in <i>Operation Manual</i></td></tr>
<tr><td colspan="2"><b>Counter Area</b></td><td>4,096: C0000 to C4095 (used for counters only)</td><td><i>9-19 Counter Area</i> in <i>Operation Manual</i></td></tr>
</table>

Item	Specification	Reference
DM Area	32K words: D00000 to D32767 Used as a general-purpose data area for reading and writing data in word units (16 bits). Words in the DM Area maintain their status when the PLC is turned OFF or the operating mode is changed. Internal Special I/O Unit DM Area: D20000 to D29599 (100 words × 96 Units) Used to set parameters for Special I/O Units. CPU Bus Unit DM Area: D30000 to D31599 (100 words × 16 Units) Used to set parameters for CPU Bus Units. Inner Board DM Area: D32000 to D32099 Used to set parameters for Inner Boards.	*9-20 Data Memory (DM) Area* in *Operation Manual*
EM Area	32K words per bank, 13 banks max.: E0_00000 to EC_32767 max. Used as a general-purpose data area for reading and writing data in word units (16 bits). Words in the EM Area maintain their status when the PLC is turned OFF or the operating mode is changed. The EM Area is divided into banks, and the addresses can be set by either of the following methods. Changing the current bank using the EMBC(281) instruction and setting addresses for the current bank. Setting bank numbers and addresses directly. EM data can be stored in files by specifying the number of the first bank.	*9-21 Extended Data Memory (EM) Area* in *Operation Manual*
Data Registers	DR0 to DR15 Store offset values for indirect addressing. One register is 16 bits (1 word). CS1 CPU Units: Data registers used independently in each task. CS1-H CPU Units: Setting to use data registers either independently in each task or to share them between tasks.	*9-23 Data Registers* in *Operation Manual*
Index Registers	IR0 to IR15 Store PLC memory addresses for indirect addressing. One register is 32 bits (2 words). Setting to use index registers either independently in each task or to share them between tasks.	*9-22 Index Registers* in *Operation Manual*
Task Flag Area	32 (TK0000 to TK0031) Task Flags are read-only flags that are ON when the corresponding cyclic task is executable and OFF when the corresponding task is not executable or in standby status.	*9-24 Task Flags* in *Operation Manual*
Trace Memory	40,000 words (trace data: 31 bits, 6 words)	*Programming Manual* (W394)
File Memory	Memory Cards: Use OMRON HMC-EF□□□ Memory Cards. (Commercially available compact flash memory cards can not be used.) EM file memory: Part of the EM Area can be converted to file memory (MS-DOS format).	*Programming Manual* (W394)

Function Specifications

Item	Specification	Reference
Constant cycle time	1 to 32,000 ms (Unit: 1 ms) When a parallel processing mode is used, the cycle time for executing instructions is constant.	Cycle time:*10-4 Computing the Cycle Time* in *Operation Manual* Constant cycle time: *Programming Manual* (W394)
Cycle time monitoring	Possible (Unit stops operating if the cycle is too long): 1 to 40,000 ms (Unit: 10 ms) When a parallel processing mode is used, the instruction execution cycle is monitored. CPU Unit operation will stop if the peripheral servicing cycle time exceeds 2 s (fixed).	Cycle time:*10-4 Computing the Cycle Time* in *Operation Manual* Cycle time monitoring: *Programming Manual* (W394)
I/O refreshing	Cyclic refreshing, immediate refreshing, refreshing by IORF(097). IORF(097) refreshes I/O bits allocated to Basic I/O Units and Special I/O Units. The CPU BUS UNIT I/O REFRESH (DLNK(226)) instruction can be used to refresh bits allocated to CPU Bus Units in the CIO and DM Areas.	I/O refreshing:*10-4 Computing the Cycle Time* in *Operation Manual* I/O refresh methods: *Programming Manual* (W394)
Timing of special refreshing for CPU Bus Units	Data links for Controller Link Units and SYSMAC LINK Units, remote I/O for DeviceNet Units, and other special refreshing for CPU Bus Units is performed at the following times: I/O refresh period and when the CPU BUS UNIT I/O REFRESH (DLNK(226)) instruction is executed	---
I/O memory holding when changing operating modes	Depends on the ON/OFF status of the IOM Hold Bit in the Auxiliary Area.	I/O memory: *SECTION 9 Memory Areas* in *Operation Manual* Holding memory areas when changing operating modes: *Programming Manual* (W394) Holding I/O memory: *9-2-3 Data Area Properties* in *Operation Manual*
Load OFF	All outputs on Output Units can be turned OFF when the CPU Unit is operating in RUN, MONITOR, or PROGRAM mode.	Load OFF: *Programming Manual* (W394)
Timer/counter PV refresh method	Binary only. **Note** BCD is not supported.	*Programming Manual* (W394)
Input response time setting	Time constants can be set for inputs from Basic I/O Units. The time constant can be increased to reduce the influence of noise and chattering or it can be decreased to detect shorter pulses on the inputs.	Input response time: *10-4-6 I/O Response Time* in *Operation Manual* Input response settings: *Programming Manual* (W394)
Startup mode setting	Supported. The CPU Unit will start in RUN mode if the PLC Setup is set to use the Programming Console mode (default) and a Programming Console is not connected.	Startup mode: *Programming Manual* (W394)
Flash memory	The user program and parameter area data (e.g., PLC Setup) are always backed up automatically in flash memory.	---

Item	Specification		Reference
Memory Card functions	Automatically reading programs (autoboot) from the Memory Card when the power is turned ON.	Supported	Memory Cards and file memory: *3-2 File Memory* in *Operation Manual* and *Programming Manual* (W394) Automatic file transfer at startup and file operations using CMND: *Programming Manual* (W394)
	Program replacement during PLC operation	Supported	Replacing the program with CMND: *Programming Manual* (W394)
	Format in which data is stored in Memory Card	User program: Program file format PLC Setup and other parameters: Data file format I/O memory: Data file format (binary format), text format, or CSV format	Data stored in the Memory Card: *Programming Manual* (W394)
	Functions for which Memory Card read/write is supported	User program instructions, Programming Devices (including Programming Consoles), Host Link computers, AR Area control bits, easy backup operation	Memory Card read/write operations: *Programming Manual* (W394)
Filing	Memory Card data and the EM (Extended Data Memory) Area can be handled as files.		File memory: *Programming Manual* (W394)
Debugging	Control set/reset, differential monitoring, data tracing (scheduled, each cycle, or when instruction is executed), storing location generating error when a program error occurs		Debugging, set/reset, differential monitoring, data tracing: *Programming Manual* (W394)
Online editing	User programs can be overwritten in program-block units when the CPU Unit is in MONITOR or PROGRAM mode. This function is not available for block programming areas. With the CX-Programmer, more than one program block can be edited at the same time. **Note** The following operations cannot be performed using online editing. • Changing function block definitions (variable tables or algorithms) • Inserting or deleting instances (Instance I/O parameters and instructions not in instances can be changed.)		Operating modes: *Programming Manual* (W394)
Program protection	Overwrite protection: Set using DIP switch. Copy protection: Password set using Programming Device.		Program protection: *Programming Manual* (W394)
Error check	User-defined errors (i.e., user can define fatal errors and non-fatal errors) The FPD(269) instruction can be used to check the execution time and logic of each programming block. FAL and FALS instructions can be used to simulate errors.		Failure diagnosis: *Programming Manual* (W394) Fatal and nonfatal errors: *11-2-4 Error Processing Flowchart* in *Operation Manual* User-defined errors: *Programming Manual* (W394)
Error log	Up to 20 errors are stored in the error log. Information includes the error code, error details, and the time the error occurred. The CPU Unit can be set so that user-defined FAL errors are not stored in the error log.		Error log: *Programming Manual* (W394)

Item	Specification	Reference
Serial communications	Built-in peripheral port: Programming Device (including Programming Console) connections, Host Links, NT Links Built-in RS-232C port: Programming Device (excluding Programming Console) connections, Host Links, no-protocol communications, NT Links	Serial communications systems: *2-5-1 Serial Communications System* in *Operation Manual* Serial communications: *Programming Manual* (W394)
	Serial Communications Board (sold separately): Protocol macros, Host Links, NT Links	
Clock	Provided on all models. Accuracy: ± 1 min. 30 s/mo. at 25°C (accuracy varies with the temperature) **Note** Used to store the time when power is turned ON and when errors occur.	Clock: *Programming Manual* (W394)
Power OFF detection time	10 to 25 ms (not fixed)	Power OFF operation and power OFF detection time: *10-3 Power OFF Operation* in *Operation Manual*
Power OFF detection delay time	0 to 10 ms (user-defined, default: 0 ms)	Power OFF detection delay time: *Programming Manual* (W394)
Memory protection	Held Areas: Holding bits, contents of Data Memory and Extended Data Memory, and status of the counter Completion Flags and present values. **Note** If the IOM Hold Bit in the Auxiliary Area is turned ON, and the PLC Setup is set to maintain the IOM Hold Bit status when power to the PLC is turned ON, the contents of the CIO Area, the Work Area, part of the Auxiliary Area, timer Completion Flag and PVs, Index Registers, and the Data Registers will be saved.	Memory protection: *9-2-3 Data Area Properties* in *Operation Manual*
Sending commands to a Host Link computer	FINS commands can be sent to a computer connected via the Host Link System by executing Network Communications Instructions from the PLC.	Host Links and non-solicited communications: *2-5-2 Systems* in *Operation Manual*
Remote programming and monitoring	Host Link communications can be used for remote programming and remote monitoring through a Controller Link System or Ethernet network.	Remote programming and monitoring: *Programming Manual* (W394) Controller Link *2-5-3 Communications Network System* in *Operation Manual*
Three-level communications	Host Link communications can be used for remote programming and remote monitoring from devices on networks up to two levels away (Controller Link Network or Ethernet Network).	Host Links and FINS message service: *2-5-2 Systems* in *Operation Manual*
Storing comments in CPU Unit	I/O comments can be stored in the CPU Unit in Memory Cards or EM file memory.	I/O comments: *CX-Programmer User Manual*
Program check	Program checks are performed at the beginning of operation for items such as no END instruction and instruction errors. CX-Programmer can also be used to check programs.	Program check: *Programming Manual* (W394)
Control output signals	RUN output: The internal contacts will turn ON (close) while the CPU Unit is operating. For CS1-H (HB) CPU Units, these terminals are provided only on the C200HW-PA204R and C200HW-PA209R Power Supply Units. For CJ1-H (HB) CPU Units, these terminals are provided only on the CJ1W-PA205R Power Supply Units.	RUN output: *Programming Manual* (W394)
Battery life	CS1-H (FB) CPU Units: Battery Set: CS1W-BAT01 CJ1-H (FB) CPU Units: Battery Set: CPM2A-BAT01	Battery life and replacement period: *12-2-1 Battery Replacement* in *Operation Manual*

Item	Specification	Reference
Self-diagnostics	CPU errors (watchdog timer), I/O verification errors, I/O bus errors, memory errors, and battery errors.	CPU, I/O bus, memory, and battery errors: *11-2-4 Error Processing Flowchart* in *Operation Manual*
Other functions	Storage of number of times power has been interrupted. (Stored in A514.)	Number of power interruptions: *10-3 Power OFF Operation* in *Operation Manual*

3-5-2 General Specifications

CS1-H (FB) CPU Units

Item	Specifications				
Power Supply Unit	C200HW-PA204	C200HW-PA204S	C200HW-PA204R	C200HW-PA209R	C200HW-PD024
Supply voltage	100 to 120 V AC or 200 to 240 V AC, 50/60 Hz				24 V DC
Operating voltage range	85 to 132 V AC or 170 to 264 V AC				19.2 to 28.8 V DC
Power consumption	120 VA max.			180 VA max.	40 W max.
Inrush current	30 A max.			30 A max./100 to 120 V AC 40 A max./200 to 240 V AC	30 A max.
Output capacity	4.6 A, 5 V DC (including the CPU Unit power supply)			9 A, 5 V DC (including the CPU Unit power supply)	4.6 A, 5 V DC (including the CPU Unit power supply)
	0.625 A, 26 V DC Total: 30 W max.	0.625 A, 26 V DC 0.8 A, 24 V DC Total: 30 W max.	0.625 A, 26 V DC Total: 30 W max.	1.3 A, 26 V DC Total: 45 W max.	0.625 A, 26 V DC Total: 30 W max.
Output terminal (service supply)	Not provided	Provided. At consumption of less than 0.3 A, 24-V DC supply will be +17% /–11%; at 0.3 A or greater, +10% /–11% (lot 0197 or later)	Not provided		
RUN output (See note 2.)	Not provided		Contact configuration: SPST-NO Switch capacity: 250 V AC, 2A (resistive load) 250 V AC, 0.5 A (induction load), 24 V DC, 2A	Contact configuration: SPST-NO Switch capacity: 240 V AC, 2A (resistive load) 120 V AC, 0.5 A (induction load) 24 V DC, 2A (resistive load) 24 V DC, 2 A (induction load)	Not provided
Insulation resistance	20 MΩ min. (at 500 V DC) between AC external and GR terminals (See note 1.)				20 MΩ min. (at 500 V DC) between DC external and GR terminals (See note 1.)

Item	Specifications	
Dielectric strength	2,300 V AC 50/60 Hz for 1 min between AC external and GR terminals (See note 1.) Leakage current: 10 mA max.	1,000 V AC 50/60 Hz for 1 min between DC external and GR terminals, leakage current: 10 mA max.
	1,000 V AC 50/60 Hz for 1 min between AC external and GR terminals (See note 1.) Leakage current: 10 mA max.	
Noise immunity	2 kV on power supply line (conforming to IEC61000-4-4)	
Vibration resistance	10 to 57 Hz, 0.075-mm amplitude, 57 to 150 Hz, acceleration: 9.8 m/s^2 in X, Y, and Z directions for 80 minutes (Time coefficient: 8 minutes ×coefficient factor 10 = total time 80 min.) CPU Unit mounted to a DIN track: 2 to 55 Hz, 2.94 m/s^2 in X, Y, and Z directions for 20 minutes	
Shock resistance	147 m/s^2 3 times each in X, Y, and Z directions (according to JIS 0041)	
Ambient operating temperature	0 to 55°C	
Ambient operating humidity	10% to 90% (with no condensation)	
Atmosphere	Must be free from corrosive gases.	
Ambient storage temperature	–20 to 75°C (excluding battery)	
Grounding	Less than 100 Ω	
Enclosure	Mounted in a panel.	
Weight	All models are each 6 kg max.	
CPU Rack dimensions (mm) (See note 3.)	2 slots: 198.5 × 157 × 123 (W x H x D) 3 slots: 260 × 130 × 123 (W x H x D) 5 slots: 330 × 130 × 123 (W x H x D) 8 slots: 435 × 130 × 123 (W x H x D) 10 slots:505 × 130 × 123 (W x H x D)	
Safety measures	Conforms to cULus and EC directives.	

Note

1. Disconnect the Power Supply Unit's LG terminal from the GR terminal when testing insulation and dielectric strength.

   Testing the insulation and dielectric strength with the LG terminal and the GR terminals connected will damage internal circuits in the CPU Unit.
2. Supported only when mounted to CPU Backplane.
3. The depth is 153 mm for the C200HW-PA209R Power Supply Unit.

CJ1-H (FB) CPU Units

Item	Specifications		
Power Supply Unit	CJ1W-PA205R	CJ1W-PA202	CJ1W-PD025
Supply voltage	100 to 240 V AC (wide-range), 50/60 Hz		24 V DC
Operating voltage and frequency ranges	85 to 264 V AC, 47 to 63 Hz		19.2 to 28.8 V DC
Power consumption	100 VA max.	50 VA max.	50 W max.
Inrush current (See note 3.)	At 100 to 120 V AC: 15 A/8 ms max. for cold start at room temperature At 200 to 240 V AC: 30 A/8 ms max. for cold start at room temperature	At 100 to 120 V AC: 20 A/8 ms max. for cold start at room temperature At 200 to 240 V AC: 40 A/8 ms max. for cold start at room temperature	At 24 V DC: 30 A/2 ms max. for cold start at room temperature
Output capacity	5.0 A, 5 V DC (including supply to CPU Unit)	2.8 A, 5 V DC (including supply to CPU Unit)	5.0 A, 5 V DC (including supply to CPU Unit)
	0.8 A, 24 V DC Total: 25 W max.	0.4 A, 24 V DC Total: 14 W max.	0.8 A, 24 V DC Total: 25 W max.

Item	Specifications	
Output terminal (service supply)	Not provided	
RUN output (See note 2.)	Contact configuration: SPST-NO Switch capacity: 250 V AC, 2 A (resistive load) 120 V AC, 0.5 A (inductive load), 24 V DC, 2A (resistive load) 24 V DC, 2 A (inductive load)	Not provided.
Insulation resistance	20 MΩ min. (at 500 V DC) between AC external and GR terminals (See note 1.)	20 MΩ min. (at 500 V DC) between DC external and GR terminals (See note 1.)
Dielectric strength	2,300 V AC 50/60 Hz for 1 min between AC external and GR terminals (See note 1.) Leakage current: 10 mA max.	
	1,000 V AC 50/60 Hz for 1 min between AC external and GR terminals (See note 1.) Leakage current: 10 mA max.	
Noise immunity	2 kV on power supply line (conforming to IEC61000-4-4)	
Vibration resistance	10 to 57 Hz, 0.075-mm amplitude, 57 to 150 Hz, acceleration: 9.8 m/s^2 in X, Y, and Z directions for 80 minutes (Time coefficient: 8 minutes ×coefficient factor 10 = total time 80 min.) (according to JIS C0040)	
Shock resistance	147 m/s^2 3 times each in X, Y, and Z directions (Relay Output Unit: 100 m/s^2) (according to JIS C0041)	
Ambient operating temperature	0 to 55°C	
Ambient operating humidity	10% to 90% (with no condensation)	
Atmosphere	Must be free from corrosive gases.	
Ambient storage temperature	–20 to 70°C (excluding battery)	
Grounding	Less than 100 Ω	
Enclosure	Mounted in a panel.	
Weight	All models are each 5 kg max.	
CPU Rack dimensions	90.7 to 466.7 × 90 × 65 mm (W x H x D) (not including cables) Note: W = a + b +20 x n + 31 x m + 14.7 a: Power Supply Unit: PA205R = 80; PA202 = 45; PD025 = 60 b: CPU Unit: CJ1-H = 62 n: Number of 32-point I/O Units or I/O Control Units m: Number of other Units.	
Safety measures	Conforms to cULus and EC Directives.	

Note

1. Disconnect the Power Supply Unit's LG terminal from the GR terminal when testing insulation and dielectric strength. Testing the insulation and dielectric strength with the LG terminal and the GR terminals connected will damage internal circuits in the CPU Unit.
2. Supported only when mounted to CPU Rack.
3. The inrush current is given for an AC Power Supply and cold start at room temperature. The inrush control circuit for an AC Power Supply uses a thermistor element with a low-temperature current control characteristic. If the ambient temperature is high or the PLC is hot-started, the thermistor will not be sufficiently cool, and the inrush current given in the table may be exceeded by up to twice the given value. When selecting fuses or breakers for external circuits, allow sufficient margin in shut-off performance.

   The inrush control circuit for an DC Power Supply uses a delay circuit with a capacitor. If the PLC is hot-started after a short power-OFF time, the capacitor will not be charged, and the inrush current given in the table may be exceeded by up to twice the given value.

3-5-3 Operation of Timer Instructions

There is an option called *Apply the same spec as TO-2047 to T2048-4095* in the PLC properties of CPU Units. This setting affects the operation of timers as described in this section.

Selecting the Option (Default)

If this option is selected, all timers will operate the same regardless of timer number, as shown in the following table.

Timer Operation for Timer Numbers T0000 to T4095

Refresh	Description
When instruction is executed	The PV is refreshed each time the instruction is executed. If the PV is 0, the Completion Flag is turned ON. If it is not 0, the Completion Flag is turned OFF.
When execution of all tasks is completed	All PV are refreshed once each cycle.
Every 80 ms	If the cycle time exceeds 80 ms, all PV are refreshed once every 80 ms.

Not Selecting the Option

If this option is not selected, the refreshing of timer instructions with timer numbers T0000 to T2047 will be different from those with timer numbers T2048 to T4095, as given below. This behavior is the same for CPU Units that do not support function blocks. (Refer to the descriptions of individual instruction in the *CS/CJ Series Instruction Reference* for details.)

Timer Operation for Timer Numbers T0000 to T2047

Refresh	Description
When instruction is executed	The PV is refreshed each time the instruction is executed. If the PV is 0, the Completion Flag is turned ON. If it is not 0, the Completion Flag is turned OFF.
When execution of all tasks is completed	All PV are refreshed once each cycle.
Every 80 ms	If the cycle time exceeds 80 ms, all PV are refreshed once every 80 ms.

Timer Operation for Timer Numbers T2048 to T4095

Refresh	Description
When instruction is executed	The PV is refreshed each time the instruction is executed. If the PV is 0, the Completion Flag is turned ON. If it is not 0, the Completion Flag is turned OFF
When execution of all tasks is completed	PV are not updated.
Every 80 ms	PV are not updated even if the cycle time exceeds 80 ms.

Select the *Apply the same spec as TO-2047 to T2048-4095* Option to ensure consistent operation when using the timer numbers allocated by default to function block variables (T3072 to T4095).

3-5-4 Battery Replacement Procedure

CJ1-H (FB) CPU Units

The battery replacement method is the same as for CJ1-H CPU Units.

CS1-H (FB) CPU Units

The battery replacement method is the same as for CS1 CPU Units. There are two battery connectors. Connect a new battery to the open connector first and then remove the old battery from the other connector. This enables periodic replacement of the battery while the CPU Unit is turned ON without a battery error being detected.

Note

(1) If the old battery is removed from the CS1-H (FB) CPU Unit first without power turned ON, an internal capacitor will back up memory even though no battery is connected. The capacitor, however, will back up memory for only 3 minutes after the power supply is turned OFF. Connect the new battery within 3 minutes.

(2) If the old battery is removed from the CS1-H (FB) CPU Unit first while power is turned ON, memory will be retained even though no battery is connected.

(3) Both the top and bottom battery connectors are equivalent. It does not matter which is used. Also, no problems will occur if a battery is connected to both connectors, e.g., the battery with the lower voltage will not receive a charge.

⚠ **Caution** The battery can be replaced while the power is turned ON even if communications are being performed. In this case, always touch a grounded piece of metal to discharge any static electricity from your body before touching any part of the PLC. Whenever possible, we recommend turning OFF the power supply to the CPU Unit before replacing the battery. Refer to the *CS Series Operation Manual* for the battery replacement procedure (either with or without power supplied).

Battery Life and Replacement Period

The effective life of the battery is 5 years at 20 °C regardless of how long power is supplied to the CPU Unit. The battery life will be reduced at higher temperatures. The battery life will also depend on the ratio of time that power is supplied. Refer to the *Operation Manual* for the CPU Unit for details. The CPU Unit models to refer to are listed in the following table.

CPU Unit	Reference CPU Unit
CS1G-CPU□□H(FB)	CS1G-CPU□□H
CS1H-CPU□□H(FB)	CS1H-CPU□□H
CJ1G-CPU□□H(FB)	CJ1G-CPU□□H

Replacement Batteries

CPU Unit	Replacement Battery Set
CS1G-CPU□□H(FB)	CS1W-BAT01
CS1H-CPU□□H(FB)	
CJ1G-CPU□□H(FB)	CPM2A-BAT01

Appendix A

Data Types

Basic Data Types

Data type	Content	Size	Range of values
BOOL	Bit data	1	0, 1
INT	Integer	16	–32,768 to 32,767
DINT	Double integer	32	–2,147,483,648 to 2,147,483,647
LINT	Long (8-byte) integer	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
UINT	Unsigned integer	16	0 to 65,535
UDINT	Unsigned double integer	32	0 to 4,294,967,295
ULINT	Unsigned long (8-byte) integer	64	0 to 18,446,744,073,709,551,615
REAL	Real number	32	-3.402823×10^{38} to $-1.175494 \times 10^{-38}$, 0, 1.175494×10^{-38} to 3.402823×10^{38}
LREAL	Long real number	64	$-1.79769313486232 \times 10^{308}$ to $-2.22507385850720 \times 10^{-308}$, 0, $2.22507385850720 \times 10^{-308}$ to $1.79769313486232 \times 10^{308}$
WORD	16-bit data	16	0 to 65,535
DWORD	32-bit data	32	0 to 4,294,967,295
LWORD	64-bit data	64	0 to 18,446,744,073,709,551,615
TIMER (See note.)	Timer (See note.)	Flag: 1 bit PV: 16 bits	Timer number: 0 to 4095 Completion Flag: 0 or 1 PV: 0 to 65536 (binary refreshing only)
COUNTER (See note.)	Counter (See note.)	Flag: 1 bit PV: 16 bits	Counter number: 0 to 4095 Completion Flag: 0 or 1 PV: 0 to 65536 (binary refreshing only)

Note The TIMER and COUNTER data types cannot be used in ST language function blocks.

Derivative Data Types

Array	1-dimensional array; 32,000 elements max.

Appendix B

Structured Text Keywords

Operators

Operation	Symbol	Data types supported by operator	CX-Programmer IEC support	Priority 1: Lowest 11: Highest
Parentheses and brackets	(*expression*), *array*[*index*]	---	Supported.	1
Function evaluation	*identifier* (*operand_list*)	---	Not supported.	2
Exponential	**	---	Not supported.	3
Complement	–	---	Not supported.	4
Negation	NOT	BOOL, WORD, DWORD, LWORD	Supported.	4
Multiplication	*	INT, DINT, UINT,UDINT, ULINT, REAL, LREAL	Supported.	5
Division	/	INT, DINT, LINT, UNIT,UDINT, ULINT, REAL, LREAL	Supported.	5
Remainder calculation	MOD	---	Not supported.	5
Addition	+	INT, DINT, LINT, UNIT,UDINT, ULINT, REAL, LREAL	Supported.	6
Subtraction	–	INT, DINT, LINT, UNIT,UDINT, ULINT, REAL, LREAL	Supported.	6
Comparisons	<, >, <=, >=	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	Supported.	7
Equality	=	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	Supported.	8
Non-equality	<>	BOOL, INT, DINT, LINT, UINT, UDINT, ULINT, WORD, DWORD, LWORD, REAL, LREAL	Supported.	8
Boolean AND	&	BOOL, WORD, DWORD, LWORD	Supported.	9
Boolean AND	AND	BOOL, WORD, DWORD, LWORD	Supported.	9
Boolean exclusive OR	XOR	BOOL, WORD, DWORD, LWORD	Supported.	10
Boolean OR	OR	BOOL, WORD, DWORD, LWORD	Supported.	11

Note Restrictions in Data Types for Structured Text Programming

- Integers can be assigned only to the WORD, DWORD, INT, DINT, UINT, UDINT, and ULINT data types. For example, if A is an INT, then A:=1 is acceptable. A syntax error will occur if anything other than an integer is assigned. For example, an error will occur for A:=2.5 if A is an INT.
- Real numbers (floating-point decimal) can be assigned only to the READ and LREAD data types. For example, if A is a REAL, then A:=1.5 is acceptable. A syntax error will occur if anything other than a real number is assigned. For example, an error will occur for A:=2 if A is an REAL.
- Contacts (TRUE/FALSE) can be assigned only to the BOOL data type. For example, if A is a BOOL, then A:=FALSE is acceptable. A syntax error will occur if a contact is assigned to anything else. For example, an error will occur for A:=FALSE if A is an INT.
- The same data type must be used in a single ST statement. For example, if A, B, and C are INT, then A;=B+C is acceptable. A syntax error will occur if different data types are mixed. For example, an error will occur for A;=B+C if A and B are INT but C is a LINT.
- The following type of data type conversion functions can be used in structured text.
  Syntax: *CurrentDataType*_TO_*NewDataType* (*VariableName*)
  Example: REAL_TO_INT (C)
  The above example changes the data type of variable C from REAL to INT.

The combinations of data types that can be converted are given in the following table.
(YES = Conversion possible, No = Conversion not possible.

FROM	TO											
	BOOL	INT	DINT	LINT	UINT	UDINT	ULINT	WORD	DWORD	LWORD	REAL	LREAL
BOOL	No	No	No	No	No	No	No	No	No	No	No	No
INT	No	No	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
DINT	No	YES	No	YES	YES	YES	YES	YES	YES	YES	YES	YES
LINT	No	YES	YES	No	YES	YES	YES	YES	YES	YES	YES	YES
UINT	No	YES	YES	YES	No	YES	YES	YES	YES	YES	YES	YES
UDINT	No	YES	YES	YES	YES	No	YES	YES	YES	YES	YES	YES
ULINT	No	YES	YES	YES	YES	YES	No	YES	YES	YES	YES	YES
WORD	No	YES	YES	YES	YES	YES	YES	No	YES	YES	No	No
DWORD	No	YES	YES	YES	YES	YES	YES	YES	No	YES	No	No
LWORD	No	YES	YES	YES	YES	YES	YES	YES	YES	No	No	No
REAL	No	YES	YES	YES	YES	YES	YES	No	No	No	No	YES
LREAL	No	YES	YES	YES	YES	YES	YES	No	No	No	YES	No

Control Statements

Control statement	Function	Example	CS-Programmer IEC
Assignment	Substitutes the results of the expression, variable, or value on the right for the variable on the left.	A:=B;	Supported
Function block call	Calls a function block.	FB_INST (*augument_list*)	Not supported
RETURN	Returns to the point from which a function block was called.	RETURN;	Not supported
IF/THEN/ELSIF/ELSE/END_IF	Evaluates an expression when the condition for it is true.	IF (*condition_1*) THEN (*expression 1*) ELSIF (*condition_2*) THEN (*expression 2*) ELSE (*expression 3*) END_IF;	Supported
CASE/ELSE/END_CASE	Evaluates an express based on the value of a variable.	CASE (*variable*) OF 1: (*expression 1*) 2: (*expression 2*) 3: (*expression 3*) ELSE (*expression 4*) END_CASE;	Supported
FOR/TO/BY/DO/END_FOR	Repeatedly evaluates an expression according to the initial value, final value, and increment.	FOR (*identifier*) := (*initial_value*) TO (*final_value*) BY (*increment*) DO (*expression*) END_FOR;	Supported
WHILE/DO/END_WHILE	Repeatedly evaluates an expression as long as a condition is true.	WHILE (*condition*) DO (*expression)* END_WHILE;	Supported
REPEAT/UNTIL/END_REPEAT	Repeatedly evaluates an expression until a condition is true.	REPEAT (*expression)* UNTIL (*condition*) END_REPEAT;	Supported
EXIT	Stops repeated processing.	EXIT;	Not supported
End of statement	Ends a statement.	;	Supported
Comment	All text between (* and *) is treated as a comment.	(**comment**)	Supported

Appendix C

External Variables

Classification	Name	External variable in CX-Programmer IEC	Data type	Address
Conditions Flags	Greater Than or Equals (GE) Flag	P_GE	BOOL	CF00
	Not Equals (NE) Flag	P_NE	BOOL	CF001
	Less Than or Equals (LE) Flag	P_LE	BOOL	CF002
	Instruction Execution Error (ER) Flag	P_ER	BOOL	CF003
	Carry (CY) Flag	P_CY	BOOL	CF004
	Greater Than (GT) Flag	P_GT	BOOL	CF005
	Equals (EQ) Flag	P_EQ	BOOL	CF006
	Less Than (LT) Flag	P_LT	BOOL	CF007
	Negative (N) Flag	P_N	BOOL	CF008
	Overflow (OF) Flag	P_OF	BOOL	CF009
	Underflow (UF) Flag	P_UF	BOOL	CF010
	Access Error Flag	P_AER	BOOL	CF011
	Always OFF Flag	P_Off	BOOL	CF114
	Always ON Flag	P_On	BOOL	CF113
Clock Pulses	0.02 second clock pulse bit	P_0_02s	BOOL	CF103
	0.1 second clock pulse bit	P_0_1s	BOOL	CF100
	0.2 second clock pulse bit	P_0_2s	BOOL	CF101
	1 minute clock pulse bit	P_1mim	BOOL	CF104
	1.0 second clock pulse bit	P_1s	BOOL	CF102
Auxiliary Area Flags/ Bits	First Cycle Flag	P_First_Cycle	BOOL	A200.11
	Step Flag	P_Step	BOOL	A200.12
	First Task Execution Flag	P_First_Cycle_Task	BOOL	A200.15
	Maximum Cycle Time	P_Max_Cycle_Time	UDINT	A262
	Present Scan Time	P_Cycle_Time_Value	UDINT	A264
	Cycle Time Error Flag	P_Cycle_Time_Error	BOOL	A401.08
	Low Battery Flag	P_Low_Battery	BOOL	A402.04
	I/O VerIFication Error Flag	P_IO_Verify_Error	BOOL	A402.09
	Output OFF Bit	P_Output_Off_Bit	BOOL	A500.15

Appendix D
Instruction Support and Operand Restrictions

The tables in this appendix tell which instructions can be used in function blocks and provide any restrictions that apply to operands, including the use of array variables and AT settings.

Instruction Support

- Instructions that are not supported by the CX-Programmer IEC or the CS1-H (FB)/CJ1-H (FB) either in function blocks or the main program are given as *Not supported* in the *Symbol* column.
- Instructions that are not supported by the CX-Programmer IEC or the CS1-H (FB)/CJ1-H (FB) in function blocks but that can be used in the main program are given as *Not supported in function blocks* in the *Symbol* column.

Restrictions on Operands

- Operands that specify the first or last of multiple words and that require specification of array variables are indicated as follows in the *Array required?* column:
  Yes: An array variable must be specified for the operand for the first or last oF multiple words.
  ---: Operands that do not require specification of array variables.

Note When specifying the first or last word of multiple words for an instruction operand, I/O parameters cannot be used to pass data to or from I/O variables. Internal array variables must be used. For multiword operands, an array variable must be prepared in advance with the required number of elements and the data must be set for the array in the function block definition. The first or last element in the array variable is then specified for the operand to set the first or last word.

- Any operands for which an AT setting is required for an I/O memory address on a remote node are indicated as *Specify address at remote node with AT setting* in the *Array required?* column.

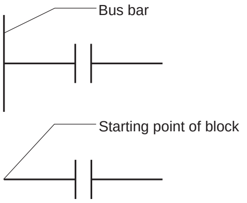
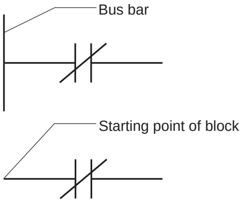
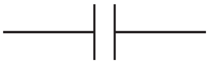
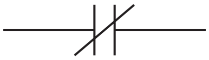
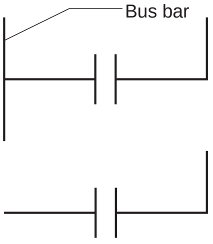
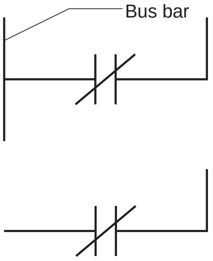
Instruction Functions

Sequence Input Instructions

*1: Not supported by CS1D

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

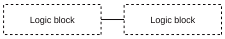
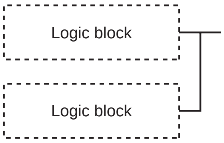
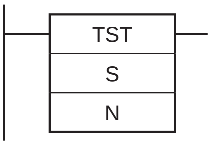
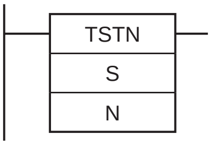
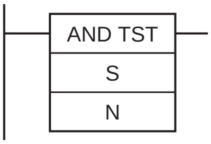
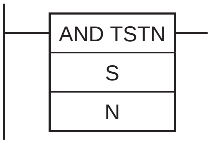
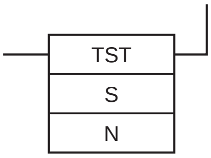
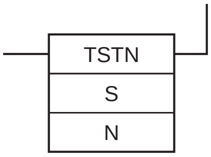
*1: CS1-H, CJ1-H, or CJ1M only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
LOAD	LD @LD %LD !LD (*1) !@LD (*1) !%LD (*1)			B: Bit	---
LOAD NOT	LD NOT !LD NOT (*1) @LD NOT (*2) %LD NOT (*2) !@LD NOT (*3) !%LD NOT (*3)			B: Bit	---
AND	AND @AND %AND !AND (*1) !@AND (*1) !%AND (*1)			B: Bit	---
AND NOT	AND NOT !AND NOT (*1) @AND NOT (*2) %AND NOT (*2) !@AND NOT (*3) !%AND NOT (*3)			B: Bit	---
OR	OR @OR %OR !OR (*1) !@OR (*1) !%OR (*1)			B: Bit	---
OR NOT	OR NOT !OR NOT(*1) @OR NOT (*2) %OR NOT (*2) !@OR NOT (*3) !%OR NOT (*3)			B: Bit	---

*1: Not supported by CS1D

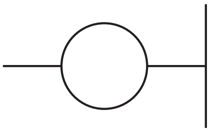
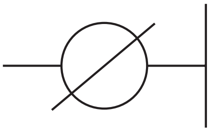
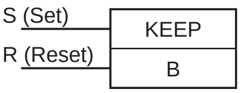
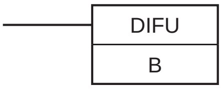
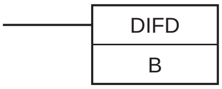
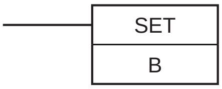
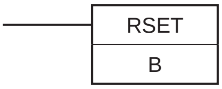
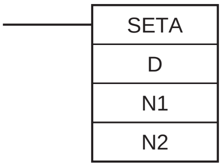
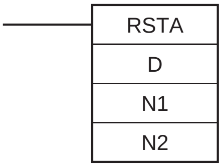
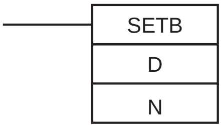
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*1: CS1-H, CJ1-H, or CJ1M only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
AND LOAD	AND LD			---	---
OR LOAD	OR LD			---	---
NOT	NOT	520		B: Bit	---
CONDITION ON	UP	521		B: Bit	---
CONDITION OFF	DOWN	522		B: Bit	---
BIT TEST	LD TST	350		S: Source word N: Bit number	--- ---
BIT TEST	LD TSTN	351		S: Source word N: Bit number	--- ---
BIT TEST	AND TST	350		S: Source word N: Bit number	--- ---
BIT TEST	AND TSTN	351		S: Source word N: Bit number	--- ---
BIT TEST	OR TST	350		S: Source word N: Bit number	--- ---
BIT TEST	OR TSTN	351		S: Source word N: Bit number	--- ---

Sequence Output Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
OUTPUT	OUT !OUT			B: Bit	---
OUTPUT NOT	OUT NOT !OUT NOT			B: Bit	---
KEEP	KEEP !KEEP	011		B: Bit	---
DIFFERENTIATE UP	DIFU !DIFU	013		B: Bit	---
DIFFERENTIATE DOWN	DIFD !DIFD	014		B: Bit	---
SET	SET @SET %SET !SET !@SET !%SET			B: Bit	---
RESET	RSET @RSET %RSET !RSET !@RSET !%RSET			B: Bit	---
MULTIPLE BIT SET	SETA @SETA	530		D: Beginning word	---
				N1: Beginning bit	---
				N2: Number of bits	---
MULTIPLE BIT RESET	RSTA @RSTA	531		D: Beginning word	---
				N1: Beginning bit	---
				N2: Number of bits	---
SINGLE BIT SET *1	SETB @SETB !SETB	532		D: Word address	---
				N: Bit number	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SINGLE BIT RESET *1	RSTB @RSTB !RSTB	533	RSTB D N	D: Word address	---
				N: Bit number	---
SINGLE BIT OUTPUT *1	OUTB @OUTB !OUTB	534	OUTB D N	D: Word address	---
				N: Bit number	---

Sequence Control Instructions

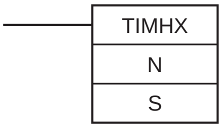
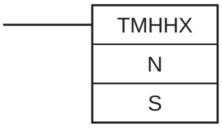
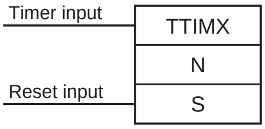
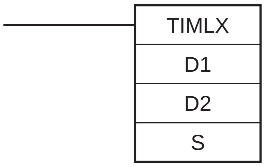
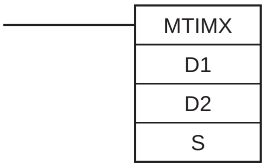
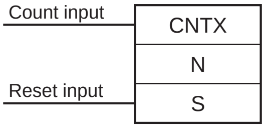
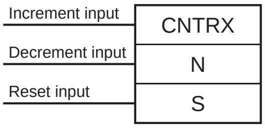
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
END	END	001	END	---	---
NO OPERATION	NOP	000	---	---	---
INTERLOCK	IL	002	IL	B: Bit	---
INTERLOCK CLEAR	ILC	003	ILC	B: Bit	---
JUMP	JMP	004	Not supported in function blocks	N: Jump number	---
JUMP END	JME	005	Not supported in function blocks	N: Jump number	---
CONDITIONAL JUMP	CJP	510	Not supported in function blocks	N: Jump number	---
CONDITIONAL JUMP	CJPN	511	Not supported in function blocks	N: Jump number	---
MULTIPLE JUMP	JMP0	515	Not supported in function blocks	---	---
MULTIPLE JUMP END	JME0	516	Not supported in function blocks	---	---
FOR-NEXT LOOPS	FOR	512	FOR N	N: Number of loops	---
BREAK LOOP	BREAK	514	BREAK	---	---
FOR-NEXT LOOPS	NEXT	513	NEXT	---	---

Timer and Counter Instructions

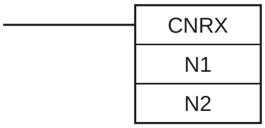
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
TIMER	TIM (BCD)		Not supported	N: Timer number	---
				S: Set value	---
	TIMX (BIN) *1	550	TIMX N S	N: Timer number	---
				S: Set value	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
HIGH-SPEED TIMER	TIMH (BCD)	015	Not supported	N: Timer number	---
				S: Set value	---
	TIMHX (BIN) *1	551		N: Timer number	---
				S: Set value	---
ONE-MS TIMER	TMHH (BCD)	540	Not supported	N: Timer number	---
				S: Set value	---
	TMHHX (BIN) *1	552		N: Timer number	---
				S: Set value	---
ACCUMULATIVE TIMER	TTIM (BCD)	087	Not supported	N: Timer number	---
				S: Set value	---
	TTIMX (BIN) *1	555		N: Timer number	---
				S: Set value	---
LONG TIMER	TIML (BCD)	542	Not supported	D1: Completion Flag	---
				D2: PV word	---
				S: SV word	---
	TIMLX (BIN) *1	553		D1: Completion Flags	---
				D2: PV word	---
				S: SV word	---
MULTI-OUTPUT TIMER	MTIM (BCD)	543	Not supported	D1: Completion Flags	---
				D2: PV word	---
				S: 1st SV word	---
	MTIMX (BIN) *1	554		D1: Completion Flags	---
				D2: PV word	---
				S: 1st SV word	---
COUNTER	CNT (BCD)		Not supported	N: Counter number	---
				S: Set value	---
	CNTX (BIN) *1	546		N: Counter number	---
				S: Set value	---
REVERSIBLE COUNTER	CNTR (BCD)	012	Not supported	N: Counter number	---
				S: Set value	---
	CNTRX (BIN) *1	548	 Increment input Decrement input Reset input CNTRX N S	N: Counter number	---
				S: Set value	---

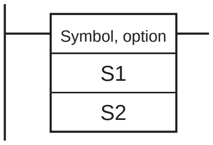
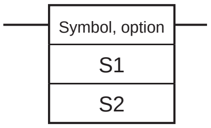
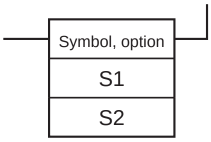
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
RESET TIMER/ COUNTER	CNR @CNR (BCD)	545	Not supported	N1: 1st number in range	---
				N2: Last number in range	---
	CNRX @CNRX (BIN) *1	547		N1: 1st number in range	---
				N2: Last number in range	---

Comparison Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*2: CJ1M only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
Symbol Comparison (Unsigned)	LD,AND, OR + =, <>, <, <=, >, >=	300 (=) 305 (<>) 310 (<) 315 (<=) 320 (>) 325 (>=)	Using LD:  Using AND:  Using OR: 	S1: Comparison data 1	---
				S2: Comparison data 2	---
Symbol Comparison (Double-word, unsigned)	LD,AND, OR + =, <>, <, <=, >, >= + L	301 (=) 306 (<>) 311 (<) 316 (<=) 321 (>) 326 (>=)	---	S1: Comparison data 1	---
				S2: Comparison data 2	---
Symbol Comparison (Signed)	LD,AND, OR + =, <>, <, <=, >, >= + S	302 (=) 307 (<>) 312 (<) 317 (<=) 322 (>) 327 (>=)	---	S1: Comparison data 1	---
				S2: Comparison data 2	---
Symbol Comparison (Double-word, signed)	LD,AND, OR + =, <>, <, <=, >, >= + SL	303 (=) 308 (<>) 313 (<) 318 (<=) 323 (>) 328 (>=)	---	S1: Comparison data 1	---
				S2: Comparison data 2	---

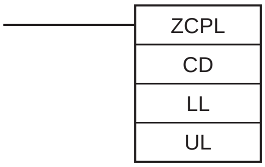
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*2: CJ1M only

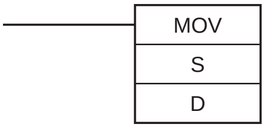
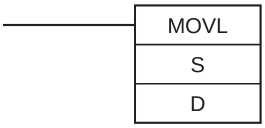
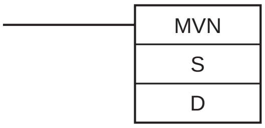
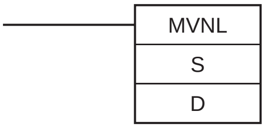
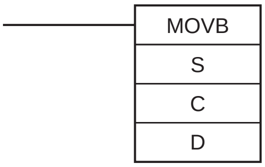
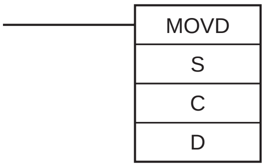
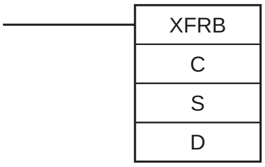
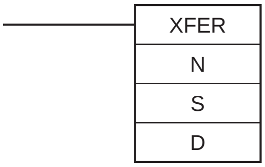
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
UNSIGNED COMPARE	CMP !CMP	020	CMP S1 S2	S1: Comparison data 1	---
				S2: Comparison data 2	---
DOUBLE UNSIGNED COMPARE	CMPL	060	CMPL S1 S2	S1: Comparison data 1	---
				S2: Comparison data 2	---
SIGNED BINARY COMPARE	CPS !CPS	114	CPS S1 S2	S1: Comparison data 1	---
				S2: Comparison data 2	---
DOUBLE SIGNED BINARY COMPARE	CPSL	115	CPSL S1 S2	S1: Comparison data 1	---
				S2: Comparison data 2	---
TABLE COMPARE	TCMP @TCMP	085	TCMP S T R	S: Source data	---
				T: 1st word of table	Yes
				R: Result word	---
MULTIPLE COMPARE	MCMP @MCMP	019	MCMP S1 S2 R	S1: 1st word of set 1	Yes
				S2: 1st word of set 2	Yes
				R: Result word	
UNSIGNED BLOCK COMPARE	BCMP @BCMP	068	BCMP S T R	S: Source data	---
				T: 1st word of table	Yes
				R: Result word	---
EXPANDED BLOCK COMPARE *2	BCMP2 @BCMP2	502	BCMP2 S T R	S: Source data	---
				T: 1st word of block	---
				R: Result word	---
AREA RANGE COMPARE *1	ZCP	088	ZCP CD LL UL	CD: Compare data (1 word)	---
				LL: Lower limit of range	---
				UL: Upper limit of range	---

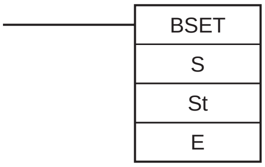
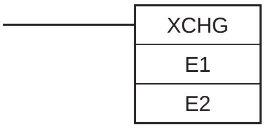
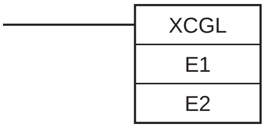
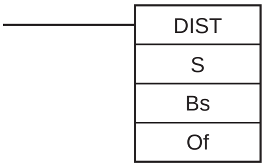
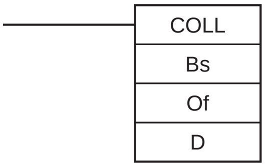
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*2: CJ1M only

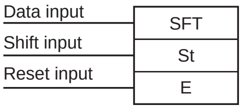
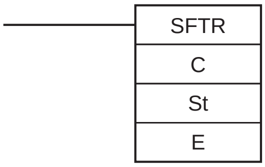
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE AREA RANGE COMPARE *1	ZCPL	116	 ZCPL CD LL UL	CD: Compare data (2 words)	---
				LL: Lower limit of range	---
				UL: Upper limit of range	---

Data Movement Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
MOVE	MOV @MOV !MOV !@MOV	021	 MOV S D	S: Source	---
				D: Destination	---
DOUBLE MOVE	MOVL @MOVL	498	 MOVL S D	S: 1st source word	---
				D: 1st destination word	---
MOVE NOT	MVN @MVN	022	 MVN S D	S: Source	---
				D: Destination	---
DOUBLE MOVE NOT	MVNL @MVNL	499	 MVNL S D	S: 1st source word	---
				D: 1st destination word	---
MOVE BIT	MOVB @MOVB	082	 MOVB S C D	S: Source word or data	---
				C: Control word	---
				D: Destination word	---
MOVE DIGIT	MOVD @MOVD	083	 MOVD S C D	S: Source word or data	---
				C: Control word	---
				D: Destination word	---
MULTIPLE BIT TRANSFERÅ@	XFRB @XFRB	062	 XFRB C S D	C: Control word	---
				S: 1st source word	Yes
				D: 1st destination word	Yes
BLOCK TRANSFER	XFER @XFER	070	 XFER N S D	N: Number of words	---
				S: 1st source word	Yes
				D: 1st destination word	Yes

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BLOCK SET	BSET @BSET	071		S: Source word	---
				St: Starting word	Yes
				E: End word	Yes
DATA EXCHANGE	XCHG @XCHG	073		E1: 1st exchange word	---
				E2: Second exchange word	---
DOUBLE DATA EXCHANGE	XCGL @XCGL	562		E1: 1st exchange word	---
				E2: Second exchange word	---
SINGLE WORD DISTRIBUTE	DIST @DIST	080		S: Source word	---
				Bs: Destination base address	Yes
				Of: Offset	---
DATA COLLECT	COLL @COLL	081		Bs: Source base address	Yes
				Of: Offset	---
				D: Destination word	---
MOVE TO REGISTER	MOVR @MOVR	560	Not supported in function blocks	S: Source (desired word orbit)	---
				D: Destination (Index Register)	---
MOVE TIMER/ COUNTER PV TO REGISTER	MOVRW @MOVRW	561	Not supported in function blocks	S: Source (desired TC number)	---
				D: Destination (Index Register)	---

Data Shift Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SHIFT REGISTER	SFT	010		St: Starting word	Yes
				E: End word	Yes
REVERSIBLE SHIFT REGISTER	SFTR @SFTR	084		C: Control word	---
				St: Starting word	Yes
				E: End word	Yes

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
ASYNCHRONOUS SHIFT REGISTER	ASFT @ASFT	017	ASFT C St E	C: Control word	---
				St: Starting word	Yes
				E: End word	Yes
WORD SHIFT	WSFT @WSFT	016	WSFT S St E	S: Source word	
				St: Starting word	Yes
				E: End word	Yes
ARITHMETIC SHIFT LEFT	ASL @ASL	025	ASL Wd	Wd: Word	---
DOUBLE SHIFT LEFT	ASLL @ASLL	570	ASLL Wd	Wd: Word	---
ARITHMETIC SHIFT RIGHT	ASR @ASR	026	ASR Wd	Wd: Word	---
DOUBLE SHIFT RIGHT	ASRL @ASRL	571	ASRL Wd	Wd: Word	---
ROTATE LEFT	ROL @ROL	027	ROL Wd	Wd: Word	---
DOUBLE ROTATE LEFT	ROLL @ROLL	572	ROLL Wd	Wd: Word	---
ROTATE LEFT WITHOUT CARRY	RLNC @RLNC	574	RLNC Wd	Wd: Word	---
DOUBLE ROTATE LEFT WITHOUT CARRY	RLNL @RLNL	576	RLNL Wd	Wd: Word	---
ROTATE RIGHT	ROR @ROR	028	ROR Wd	Wd: Word	---
DOUBLE ROTATE RIGHT	RORL @RORL	573	RORL Wd	Wd: Word	---
ROTATE RIGHT WITHOUT CARRY	RRNC @RRNC	575	RRNC Wd	Wd: Word	---
DOUBLE ROTATE RIGHT WITHOUT CARRY	RRNL @RRNL	577	RRNL Wd	Wd: Word	---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
ONE DIGIT SHIFT LEFT	SLD @SLD	074	SLD St E	St: Starting word E: End word	Yes Yes
ONE DIGIT SHIFT RIGHT	SRD @SRD	075	SRD St E	St: Starting word E: End word	Yes Yes
SHIFT N-BIT DATA LEFT	NSFL @NSFL	578	NSFL D C N	D: Beginning word for shift C: Beginning bit N: Shift data length	--- --- ---
SHIFT N-BIT DATA RIGHT	NSFR @NSFR	579	NSFR D C N	D: Beginning word for shift C: Beginning bit N: Shift data length	--- --- ---
SHIFT N-BITS LEFT	NASL @NASL	580	NASL D C	D: Shift word C: Control word	--- ---
DOUBLE SHIFT N-BITS LEFT	NSLL @NSLL	582	NSLL D C	D: Shift word C: Control word	--- ---
SHIFT N-BITS RIGHT	NASR @NASR	581	NASR D C	D: Shift word C: Control word	--- ---
DOUBLE SHIFT N-BITS RIGHT	NSRL @NSRL	583	NSRL D C	D: Shift word C: Control word	--- ---

Increment/Decrement Instructions

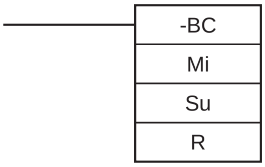
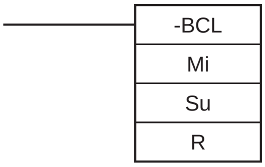
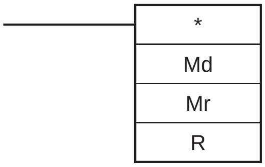
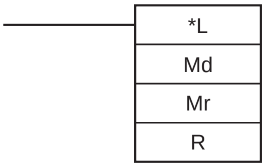
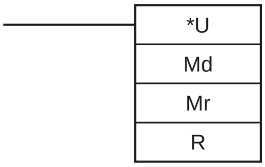
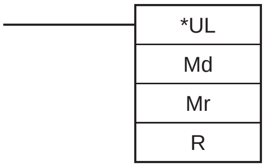
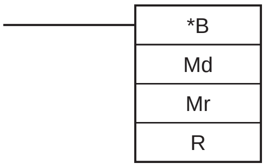
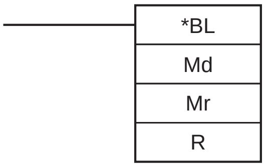
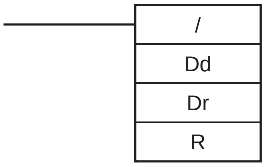
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
INCREMENT BINARY	++ @++	590	+ + Wd	Wd: Word	---
DOUBLE INCREMENT BINARY	++L @++L	591	++L Wd	Wd: Word	---
DECREMENT BINARY	-- @--	592	-- Wd	Wd: Word	---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE DECREMENT BINARY	--L @--L	593	--L Wd	Wd: 1st word	---
INCREMENT BCD	++B @++B	594	++B Wd	Wd: Word	---
DOUBLE INCREMENT BCD	++BL @++BL	595	++BL Wd	Wd: 1st word	---
DECREMENT BCD	--B @--B	596	--B Wd	Wd: Word	---
DOUBLE DECREMENT BCD	--BL @--BL	597	--BL Wd	Wd: 1st word	---

Symbol Math Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SIGNED BINARY ADD WITHOUT CARRY	+ @+	400	+ Au Ad R	Au: Augend word Ad: Addend word R: Result word	--- --- ---
DOUBLE SIGNED BINARY ADD WITHOUT CARRY	+L @+L	401	+L Au Ad R	Au: 1st augend word Ad: 1st addend word R: 1st result word	--- --- ---
SIGNED BINARY ADD WITH CARRY	+C @+C	402	+C Au Ad R	Au: Augend word Ad: Addend word R: Result word	--- --- ---
DOUBLE SIGNED BINARY ADD WITH CARRY	+CL @+CL	403	+CL Au Ad R	Au: 1st augend word Ad: 1st addend word R: 1st result word	--- --- ---
BCD ADD WITHOUT CARRY	+B @+B	404	+B Au Ad R	Au: Augend word Ad: Addend word R: Result word	--- --- ---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE BCD ADD WITHOUT CARRY	+BL @+BL	405	+BL Au Ad R	Au: 1st augend word Ad: 1st addend word R: 1st result word	--- --- ---
BCD ADD WITH CARRY	+BC @+BC	406	+BC Au Ad R	Au: Augend word Ad: Addend word R: Result word	--- --- ---
DOUBLE BCD ADD WITH CARRY	+BCL @+BCL	407	+BCL Au Ad R	Au: 1st augend word Ad: 1st addend word R: 1st result word	--- --- ---
SIGNED BINARY SUBTRACT WITHOUT CARRY	- @-	410	- Mi Su R	Mi: Minuend word Su: Subtrahend word R: Result word	--- --- ---
DOUBLE SIGNED BINARY SUBTRACT WITHOUT CARRY	-L @-L	411	-L Mi Su R	Mi: Minuend word Su: Subtrahend word R: Result word	--- --- ---
SIGNED BINARY SUBTRACT WITH CARRY	-C @-C	412	-C Mi Su R	Mi: Minuend word Su: Subtrahend word R: Result word	--- --- ---
DOUBLE SIGNED BINARY WITH CARRY	-CL @-CL	413	-CL Mi Su R	Mi: Minuend word Su: Subtrahend word R: Result word	--- --- ---
BCD SUBTRACT WITHOUT CARRY	-B @-B	414	-B Mi Su R	Mi: Minuend word Su: Subtrahend word R: Result word	--- --- ---
DOUBLE BCD SUBTRACT WITHOUT CARRY	-BL @-BL	415	-BL Mi Su R	Mi: 1st minuend word Su: 1st subtrahend word R: 1st result word	--- --- ---

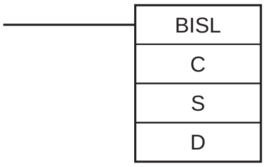
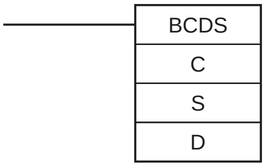
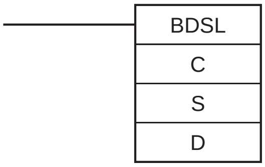
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BCD SUBTRACT WITH CARRY	-BC @-BC	416		Mi: Minuend word	---
				Su: Subtrahend word	---
				R: Result word	---
DOUBLE BCD SUBTRACT WITH CARRY	-BCL @-BCL	417		Mi: 1st minuend word	---
				Su: 1st subtrahend word	---
				R: 1st result word	---
SIGNED BINARY MULTIPLY	* @*	420		Md: Multiplicand word	---
				Mr: Multiplier word	---
				R: Result word	---
DOUBLE SIGNED BINARY MULTIPLY	*L @*L	421		Md: 1st multiplicand word	---
				Mr: 1st multiplier word	---
				R: 1st result word	---
UNSIGNED BINARY MULTIPLY	*U @*U	422		Md: Multiplicand word	---
				Mr: Multiplier word	---
				R: Result word	---
DOUBLE UNSIGNED BINARY MULTIPLY	*UL @*UL	423		Md: 1st multiplicand word	---
				Mr: 1st multiplier word	---
				R: 1st result word	---
BCD MULTIPLY	*B @*B	424		Md: Multiplicand word	---
				Mr: Multiplier word	---
				R: Result word	---
DOUBLE BCD MULTIPLY	*BL @*BL	425		Md: 1st multiplicand word	---
				Mr: 1st multiplier word	---
				R: 1st result word	---
SIGNED BINARY DIVIDE	/ @/	430		Dd: Dividend word	---
				Dr: Divisor word	---
				R: Result word	Yes

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE SIGNED BINARY DIVIDE	/L @/L	431	/L Dd Dr R	Dd: 1st dividend word	---
				Dr: 1st divisor word	---
				R: 1st result word	Yes
UNSIGNED BINARY DIVIDE	/U @/U	432	/U Dd Dr R	Dd: Dividend word	---
				Dr: Divisor word	---
				R: Result word	Yes
DOUBLE UNSIGNED BINARY DIVIDE	/UL @/UL	433	/UL Dd Dr R	Dd: 1st dividend word	---
				Dr: 1st divisor word	---
				R: 1st result word	Yes
BCD DIVIDE	/B @/B	434	/B Dd Dr R	Dd: Dividend word	---
				Dr: Divisor word	---
				R: Result word	Yes
DOUBLE BCD DIVIDE	/BL @/BL	435	/BL Dd Dr R	Dd: 1st dividend word	---
				Dr: 1st divisor word	---
				R: 1st result word	Yes

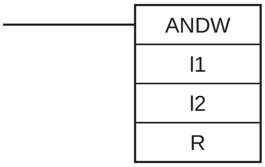
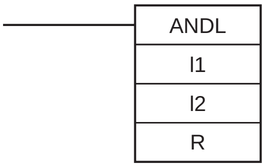
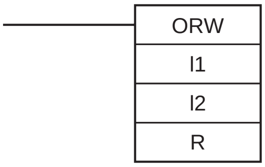
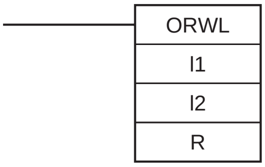
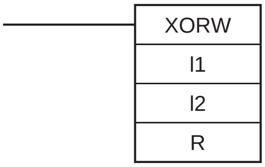
Conversion Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BCD-TO-BINARY	BIN @BIN	023	BIN S R	S: Source word	---
				R: Result word	---
DOUBLE BCD-TO-DOUBLE BINARY	BINL @BINL	058	BINL S R	S: 1st source word	---
				R: 1st result word	---
BINARY-TO-BCD	BCD @BCD	024	BCD S R	S: Source word	---
				R: Result word	---
DOUBLE BINARY-TO-DOUBLE BCD	BCDL @BCDL	059	BCDL S R	S: 1st source word	---
				R: 1st result word	---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
2'S COMPLEMENT	NEG @NEG	160	NEG S R	S: Source word	---
				R: Result word	---
DOUBLE 2'S COMPLEMENT	NEGL @NEGL	161	NEGL S R	S: 1st source word	---
				R: 1st result word	---
16-BIT TO 32-BIT SIGNED BINARY	SIGN @SIGN	600	SIGN S R	S: Source word	---
				R: 1st result word	---
DATA DECODER	MLPX @MLPX	076	MLPX S C R	S: Source word	---
				C: Control word	---
				R: 1st result word	Yes
DATA ENCODER	DMPX @DMPX	077	DMPX S R C	S: 1st source word	Yes
				R: Result word	---
				C: Control word	---
ASCII CONVERT	ASC @ASC	086	ASC S Di D	S: Source word	Yes
				Di: Digit designator	---
				D: 1st destination word	Yes
ASCII TO HEX	HEX @HEX	162	HEX S Di D	S: 1st source word	Yes
				Di: Digit designator	---
				D: Destination word	Yes
COLUMN TO LINE	LINE @LINE	063	LINE S N D	S: 1st source word	Yes
				N: Bit number	---
				D: Destination word	---
LINE TO COLUMN	COLM @COLM	064	COLM S D N	S: Source word	---
				D: 1st destination word	Yes
				N: Bit number	---
SIGNED BCD-TO-BINARY	BINS @BINS	470	BINS C S D	C: Control word	---
				S: Source word	---
				D: Destination word	---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE SIGNED BCD-TO-BINARY	BISL @BISL	472		C: Control word	---
				S: 1st source word	---
				D: 1st destination word	---
SIGNED BINARY-TO-BCD	BCDS @BCDS	471		C: Control word	---
				S: Source word	---
				D: Destination word	---
DOUBLE SIGNED BINARY-TO-BCD	BDSL @BDSL	473		C: Control word	---
				S: 1st source word	---
				D: 1st destination word	---

Logic Instructions

Instruction	Mnemonic	Function code	Symbol	Operand	Array required?
LOGICAL AND	ANDW @ANDW	034		I1: Input 1	---
				I2: Input 2	---
				R: Result word	---
DOUBLE LOGICAL AND	ANDL @ANDL	610		I1: Input 1	---
				I2: Input 2	---
				R: Result word	---
LOGICAL OR	ORW @ORW	035		I1: Input 1	---
				I2: Input 2	---
				R: Result word	---
DOUBLE LOGICAL OR	ORWL @ORWL	611		I1: Input 1	---
				I2: Input 2	---
				R: Result word	---
EXCLUSIVE OR	XORW @XORW	036		I1: Input 1	---
				I2: Input 2	---
				R: Result word	---

Instruction	Mnemonic	Function code	Symbol	Operand	Array required?
DOUBLE EXCLUSIVE OR	XORL @XORL	612	XORL I1 I2 R	I1: Input 1 I2: Input 2 R: Result word	--- --- ---
EXCLUSIVE NOR	XNRW @XNRW	037	XNRW I1 I2 R	I1: Input 1 I2: Input 2 R: Result word	--- --- ---
DOUBLE EXCLUSIVE NOR	XNRL @XNRL	613	XNRL I1 I2 R	I1: Input 1 I2: Input 2 R: Result word	--- --- ---
COMPLEMENT	COM @COM	029	COM Wd	Wd: Word	---
DOUBLE COMPLEMENT	COML @COML	614	COML Wd	Wd: Word	---

Special Math Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BINARY ROOT	ROTB @ROTB	620	ROTB S R	S: 1st source word R: Result word	--- ---
BCD SQUARE ROOT	ROOT @ROOT	072	ROOT S R	S: 1st source word R: Result word	--- ---
ARITHMETIC PROCESS	APR @APR	069	APR C S R	C: Control word S: Source data R: Result word	Yes --- ---
FLOATING POINT DIVIDE	FDIV @FDIV	079	FDIV Dd Dr R	Dd: 1st dividend word Dr: 1st divisor word R: 1st result word	--- --- ---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BIT COUNTER	BCNT @BCNT	067	BCNT N S R	N: Number of words	---
				S: 1st source word	Yes
				R: Result word	---

Floating-point Math Instructions

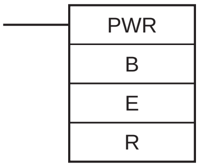
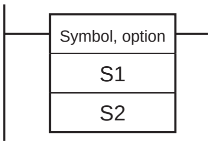
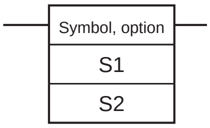
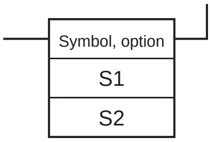
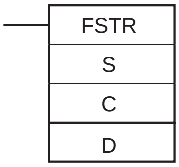
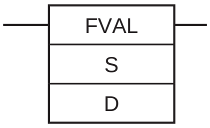
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
FLOATING TO 16-BIT	FIX @FIX	450	FIX S R	S: 1st source word	---
				R: Result word	---
FLOATING TO 32-BIT	FIXL @FIXL	451	FIXL S R	S: 1st source word	---
				R: Result word	---
16-BIT TO FLOATING	FLT @FLT	452	FLT S R	S: Source word	---
				R: 1st result word	---
32-BIT TO FLOATING	FLTL @FLTL	453	FLTL S R	S: 1st source word	---
				R: Result word	---
FLOATING-POINT ADD	+F @+F	454	+F Au Ad R	Au: 1st augend word	---
				Ad: 1st addend word	---
				R: 1st result word	---
FLOATING-POINT SUBTRACT	-F @-F	455	-F Mi Su R	Mi: 1st Minuend word	---
				Su: 1st Subtrahend word	---
				R: 1st result word	---
FLOATING- POINT MULTIPLY	*F @*F	456	* F Md Mr R	Md: 1st Multiplicand word	---
				Mr: 1st Multiplier word	---
				R: 1st result word	---
FLOATING- POINT DIVIDE	/F @/F	457	/F Dd Dr R	Dd: 1st Dividend word	---
				Dr: 1st Divisor word	---
				R: 1st result word	---

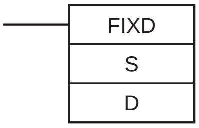
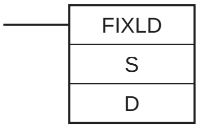
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

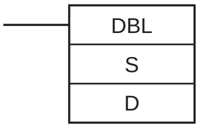
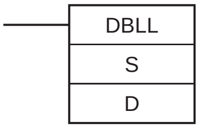
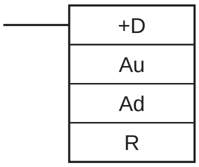
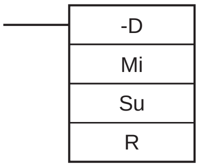
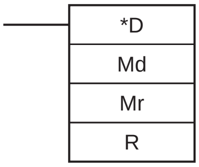
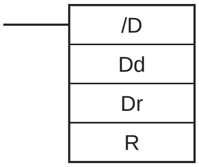
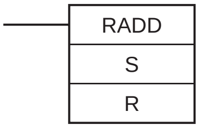
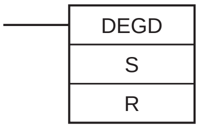
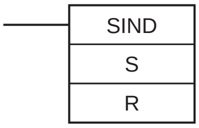
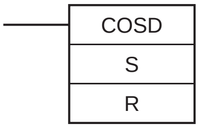
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DEGREES TO RADIANS	RAD @RAD	458	RAD S R	S: 1st source word	---
				R: 1st result word	---
RADIANS TO DEGREES	DEG @DEG	459	DEG S R	S: 1st source word	---
				R: 1st result word	---
SINE	SIN @SIN	460	SIN S R	S: 1st source word	---
				R: 1st result word	---
COSINE	COS @COS	461	COS S R	S: 1st source word	---
				R: 1st result word	---
TANGENT	TAN @TAN	462	TAN S R	S: 1st source word	---
				R: 1st result word	---
ARC SINE	ASIN @ASIN	463	ASIN S R	S: 1st source word	---
				R: 1st result word	---
ARC COSINE	ACOS @ACOS	464	ACOS S R	S: 1st source word	---
				R: 1st result word	---
ARC TANGENT	ATAN @ATAN	465	ATAN S R	S: 1st source word	---
				R: 1st result word	---
SQUARE ROOT	SQRT @SQRT	466	SQRT S R	S: 1st source word	---
				R: 1st result word	---
EXPONENT	EXP @EXP	467	EXP S R	S: 1st source word	---
				R: 1st result word	---
LOGARITHM	LOG @LOG	468	LOG S R	S: 1st source word	---
				R: 1st result word	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
EXPONENTIAL POWER	PWR @PWR	840		B: 1st base word E: 1st exponent word R: 1st result word	--- --- ---
Floating Symbol Comparison *1	LD, AND, OR + =F, <>F, <F, <=F, >F, >=F	329 (=F) 330 (<>F) 331 (<F) 332 (<=F) 333 (>F) 334 (>=F)	Using LD:  Using AND:  Using OR: 	S1:Comparoson data 1 S2:Comparison data 2	--- ---
FLOATING- POINT TO ASCII *1	FSTR @FSTR	448		S: 1st source word C: Control word D: Destination word	--- --- Yes
ASCII TO FLOATING-POINT *1	FVAL @FVAL	449		S: Source word D: 1st destination word	Yes ---

Double-precision Floating-point Instructions (CS1-H, CJ1-H, CJ1M, or CS1D Only)

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE FLOATING TO 16-BIT BINARY	FIXD @FIXD	841		S: 1st source word D: Destination word	--- ---
DOUBLE FLOATING TO 32-BIT BINARY	FIXLD @FIXLD	842		S: 1st source word D: 1st destination word	--- ---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
16-BIT BINARY TO DOUBLE FLOATING	DBL @DBL	843		S: Source word D: 1st destination word	--- ---
32-BIT BINARY TO DOUBLE FLOATING	DBLL @DBLL	844		S: 1st source word D: 1st destination word	--- ---
DOUBLE FLOATING-POINT ADD	+D @+D	845		Au: 1st augend word Ad: 1st addend word R: 1st result word	--- --- ---
DOUBLE FLOATING-POINT SUBTRACT	-D @-D	846		Mi: 1st minuend word Su: 1st subtrahend word R: 1st result word	--- --- ---
DOUBLE FLOATING-POINT MULTIPLY	*D @*D	847		Md: 1st multiplicand word Mr: 1st multiplier word R: 1st result word	--- --- ---
DOUBLE FLOATING-POINT DIVIDE	/D @/D	848		Dd: 1st Dividend word Dr: 1st divisor word R: 1st result word	--- --- ---
DOUBLE DEGREES TO RADIANS	RADD @RADD	849		S: 1st source word R: 1st result word	--- ---
DOUBLE RADIANS TO DEGREES	DEGD @DEGD	850		S: 1st source word R: 1st result word	--- ---
DOUBLE SINE	SIND @SIND	851		S: 1st source word R: 1st result word	--- ---
DOUBLE COSINE	COSD @COSD	852		S: 1st source word R: 1st result word	--- ---

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE TANGENT	TAND @TAND	853	TAND S R	S: 1st source word R: 1st result word	--- ---
DOUBLE ARC SINE	ASIND @ASIND	854	ASIND S R	S: 1st source word R: 1st result word	--- ---
DOUBLE ARC COSINE	ACOSD @ACOSD	855	ACOSD S R	S: 1st source word R: 1st result word	--- ---
DOUBLE ARC TANGENT	ATAND @ATAND	856	ATAND S R	S: 1st source word R: 1st result word	--- ---
DOUBLE SQUARE ROOT	SQRTD @SQRTD	857	SQRTD S R	S: 1st source word R: 1st result word	--- ---
DOUBLE EXPONENT	EXPD @EXPD	858	EXPD S R	S: 1st source word R: 1st result word	--- ---
DOUBLE LOGARITHM	LOGD @LOGD	859	LOGD S R	S: 1st source word R: 1st result word	--- ---
DOUBLE EXPONENTIAL POWER	PWRD @PWRD	860	PWRD B E R	B: 1st base word E: 1st exponent word R: 1st result word	--- --- ---

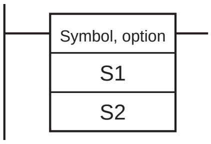
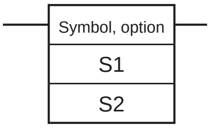
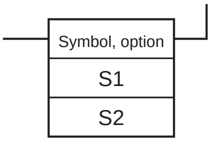
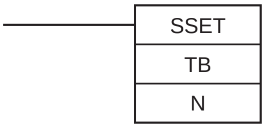
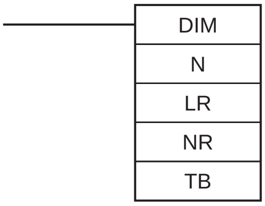
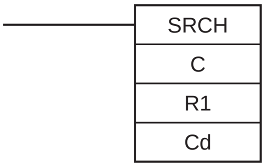
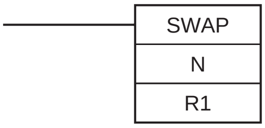
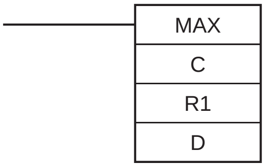
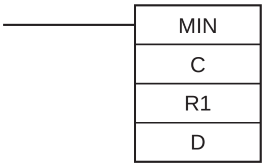
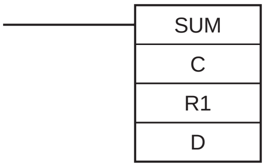
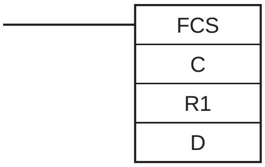
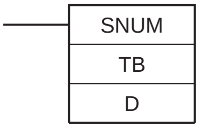
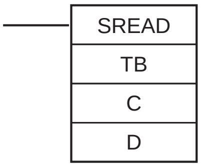
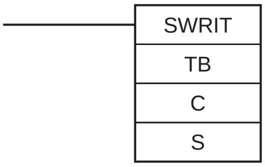
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DOUBLE SYMBOL COMPARISON	LD, AND, OR + =D, <>D, <D, <=D, >D, >=D	335 (=D) 336 (<>D) 337 (<D) 338 (<=D) 339 (>D) 340 (>=D)	Using LD:  Symbol, option S1 S2 Using AND:  Symbol, option S1 S2 Using OR:  Symbol, option S1 S2	S1:Comparoson data 1	---
				S2:Comparison data 2	---

Table Data Processing Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SET STACK	SSET @SSET	630	 SSET TB N	TB: 1st stack address	Yes
				N: Number of words	-
PUSH ONTO STACK	PUSH @PUSH	632	Not supported in function blocks	TB: 1st stack address	Yes
				S: Source word	-
FIRST IN FIRST OUT	FIFO @FIFO	633	Not supported in function blocks	TB: 1st stack address	Yes
				D: Destination word	-
LAST IN FIRST OUT	LIFO @LIFO	634	Not supported in function blocks	TB: 1st stack address	Yes
				D: Destination word	---
DIMENSION RECORD TABLE	DIM @DIM	631	 DIM N LR NR TB	N: Table number	---
				LR: Length of each record	---
				NR: Number of records	---
				TB: 1st table word	Yes
SET RECORD LOCATION	SETR @SETR	635	Not supported in function blocks	N: Table number	---
				R: Record number	---
				D: Destination Index Register	---
GET RECORD NUMBER	GETR @GETR	636	Not supported in function blocks	N: Table number	---
				IR: Index Register	---
				D: Destination word	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DATA SEARCH	SRCH @SRCH	181		C: 1st control word	---
				R1: 1st word in range	Yes
				Cd: Comparison data	---
SWAP BYTES	SWAP @SWAP	637		N: Number of words	---
				R1: 1st word in range	Yes
FIND MAXIMUM	MAX @MAX	182		C: 1st control word	---
				R1: 1st word in range	Yes
				D: Destination word	---
FIND MINIMUM	MIN @MIN	183		C: 1st control word	---
				R1: 1st word in range	Yes
				D: Destination word	---
SUM	SUM @SUM	184		C: 1st control word	---
				R1: 1st word in range	Yes
				D: 1st destination word	---
FRAME CHECK SUM	FCS @FCS	180		C: 1st control word	---
				R1: 1st word in range	Yes
				D: 1st destination word	---
STACK SIZE READ *1	SNUM @SNUM	638		TB: First stack address	Yes
				D: Destination word	---
STACK DATA READ *1	SREAD @SREAD	639		TB: First stack address	Yes
				C: Offset value	---
				D: Destination word	---
STACK DATA OVER-WRITE *1	SWRIT @SWRIT	640		TB: First stack address	Yes
				C: Offset value	---
				S: Source data	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

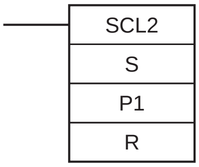
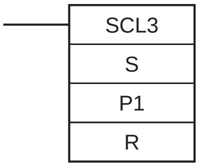
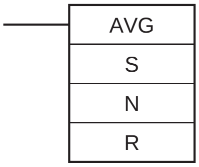
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
STACK DATA INSERT *1	SINS @SINS	641	SINS TB C S	TB: First stack address	Yes
				C: Offset value	---
				S: Source data	---
STACK DATA DELETE *1	SDEL @SDEL	642	SDEL TB C D	TB: First stack address	Yes
				C: Offset value	---
				D: Destination word	---

Data Control Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
PID CONTROL	PID	190	PID S C D	S: Input word	---
				C: 1st parameter word	Yes
				D: Output word	---
PID CONTROL WITH AUTO TUNING *1	PIDAT	191	PIDAT S C D	S: Input word	---
				C: 1st parameter word	Yes
				D: Output word	---
LIMIT CONTROL	LMT @LMT	680	LMT S C D	S: Input word	---
				C: 1st limit word	Yes
				D: Output word	---
DEAD BAND CONTROL	BAND @BAND	681	BAND S C D	S: Input word	---
				C: 1st limit word	Yes
				D: Output word	---
DEAD ZONE CONTROL	ZONE @ZONE	682	ZONE S C D	S: Input word	---
				C: 1st limit word	Yes
				D: Output word	---
SCALING	SCL @SCL	194	SCL S P1 R	S: Input word	---
				P1: 1st parameter word	Yes
				R: Result word	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SCALING 2	SCL2 @SCL2	486		S: Source word	---
				P1: 1st parameter word	Yes
				R: Result word	---
SCALING 3	SCL3 @SCL3	487		S: Source word	---
				P1: 1st parameter word	Yes
				R: Result word	---
AVERAGE	AVG	195		S: Source word	---
				N: Number of cycles	---
				R: Result word	Yes

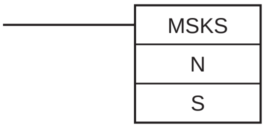
Subroutine Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

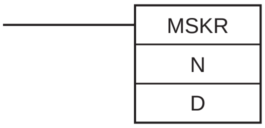
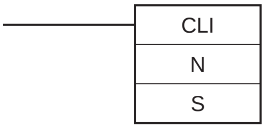
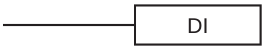
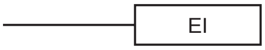
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SUBROUTINE CALL	SBS @SBS	091	Not supported in function blocks	N: Subroutine number	---
SUBROUTINE ENTRY	SBN	092	Not supported in function blocks	N: Subroutine number	---
SUBROUTINE RETURN	RET	093	Not supported in function blocks		---
MACRO	MCRO @MCRO	099	Not supported in function blocks	N: Subroutine number	---
				S: 1st input parameter word	---
				D: 1st output parameter word	---
GLOBAL SUBROUTINE CALL *1	GSBS @GSBS	750	Not supported in function blocks	N: Subroutine number	---
GLOBAL SUBROUTINE ENTRY *1	GSBN	751	Not supported in function blocks	N: Subroutine number	---
GLOBAL SUBROUTINE RETURN *1	GRET	752	Not supported in function blocks		---

Interrupt Control Instructions

*1: Not supported by CS1D.

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SET INTERRUPT MASK *1	MSKS @MSKS	690		N: Interrupt identifier	-
				S: Interrupt data	-

*1: Not supported by CS1D.

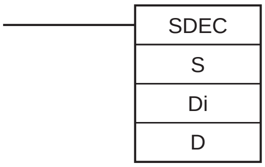
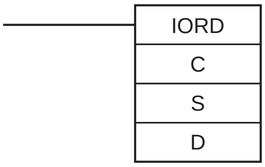
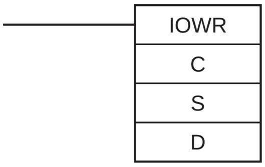
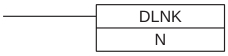
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
READ INTERRUPT MASK *1	MSKR @MSKR	692		N: Interrupt identifier	-
				D: Destination word	-
CLEAR INTERRUPT *1	CLI @CLI	691		N: Interrupt identifier	-
				S: Interrupt data	-
DISABLE INTERRUPTS *1	DI @DI	693			-
ENABLE INTERRUPTS *1	EI	694			-

Step Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
STEP DEFINE	STEP	008	Not supported in function blocks	B: Bit	---
STEP START	SNXT	009	Not supported in function blocks	B: Bit	---

Basic I/O Unit Instructions

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
I/O REFRESH	IORF @IORF	097	Not supported in function blocks	St: Starting word	---
				E: End word	---
7-SEGMENT DECODER	SDEC @SDEC	078		S: Source word	---
				Di: Digit designator	---
				D: 1st destination word	Yes
INTELLIGENT I/O READ	IORD @IORD	222		C: Control data	---
				S: Transfer source and number of words	Yes
				D: Transfer destination and number of words	Yes
INTELLIGENT I/O WRITE	IOWR @IOWR	223		C: Control data	---
				S: Transfer source and number of words	Yes
				D: Transfer destination and number of words	Yes
CPU BUS UNIT I/O REFRESH *1	DLNK @DLNK	226		N: Unit number	---

Serial Communications Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
PROTOCOL MACRO	PMCR @PMCR	260	PMCR C1 C2 S R	C1:Control word 1	---
				C2: Control word 2	---
				S: 1st send word	Yes
				R: 1st receive word	Yes
TRANSMIT	TXD @TXD	236	TXD S C N	S: 1st source word	Yes
				C: Control word	---
				N: Number of bytes 0000 to 0100 hex(0 to 256 decimal)	---
RECEIVE	RXD @RXD	235	RXD D C N	D: 1st destination word	Yes
				C: Control word	---
				N: Number of bytes to store 0000 to 0100 hex(0 to 256 decimal)	---
CHANGE SERIAL PORT SETUP	STUP @STUP	237	STUP C S	C: Control word (port)	---
				S: First source word	Yes

Network Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
NETWORK SEND	SEND @SEND	090	SEND S D C	S: 1st source word	Yes
				D: 1st destination word	Specify address at remote node with AT setting.
				C: 1st control word	Yes
NETWORK RECEIVE	RECV @RECV	098	RECV S D C	S: 1st source word	Specify address at remote node with AT setting.
				D: 1st destination word	Yes
				C: 1st control word	Yes
DELIVER COMMAND	CMND @CMND	490	CMND S D C	S: 1st command word	Yes
				D: 1st response word	Yes
				C: 1st control word	Yes

File Memory Instructions

Instruction	Mnemonic	Function code	Symbol	Operand	Array required?
READ DATA FILE	FREAD @FREAD	700	FREAD C S1 S2 D	C: Control word	---
				S1: 1st source word	Yes
				S2: Filename	Yes
				D: 1st destination word	Yes
WRITE DATA FILE	FWRIT @FWRIT	701	FWRIT C D1 D2 S	C: Control word	---
				D1: 1st destination word	Yes
				D2: Filename	Yes
				S: 1st source word	Yes

Display Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DISPLAY MESSAGE	MSG @MSG	046	MSG N M	N: Message number	---
				M: 1st message word	Yes

Clock Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
CALENDAR ADD	CADD @CADD	730	CADD C T R	C: 1st calendar word	Yes
				T: 1st time word	Yes
				R: 1st result word	Yes
CALENDAR SUBTRACT	CSUB @CSUB	731	CSUB C T R	C: 1st calendar word	Yes
				T: 1st time word	Yes
				R: 1st result word	Yes
HOURS TO SECONDS	SEC @SEC	065	SEC S D	S: 1st source word	Yes
				D: 1st destination word	Yes
SECONDS TO HOURS	HMS @HMS	066	HMS S D	S: 1st source word	Yes
				D: 1st destination word	Yes
CLOCK ADJUSTMENT	DATE @DATE	735	DATE S	S: 1st source word	Yes

Debugging Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
TRACE MEMORY SAMPLING	TRSM	045	TRSM		---

Failure Diagnosis Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
FAILURE ALARM	FAL @FAL	006	FAL N M	N: FAL number	---
				M: 1st message word or error code to generate(#0000 to #FFFF)	---
SEVERE FAILURE ALARM	FALS	007	FALS N M	N: FALS number	---
				M: 1st message word or error code to generate(#0000 to #FFFF)	---
FAILURE POINT DETECTION	FPD	269	Not supported in function blocks	C: Control word	---
				T: Monitoring time	---
				R: 1st register word	Yes

Other Instructions

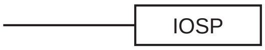
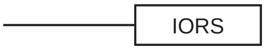
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*2: CS1-H, CJ1-H, or CJ1M only (Not supported by CS1D, CS1, or CJ1.)

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
SET CARRY	STC @STC	040	STC		---
CLEAR CARRY	CLC @CLC	041	CLC		---
SELECT EM BANK	EMBC @EMBC	281	Not supported	N: EM bank number.	---
EXTEND MAXIMUM CYCLE TIME	WDT @WDT	094	WDT T	T: Timer setting	---
SAVE Condition FlagS *1	CCS @CCS	282	CCS		---
LOAD Condition FlagS *1	CCL @CCL	283	CCL		---
CONVERT ADDRESS FROM CV *1	FRMCV @FRMCV	284	Not supported in function blocks	S: Word containing CV-series memory address	---
				D: Destination Index Register	---
CONVERT ADDRESS TO CV *1	TOCV @TOCV	285	Not supported in function blocks	S: Index Register containing CS Series memory address	---
				D: Destination word	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

*2: CS1-H, CJ1-H, or CJ1M only (Not supported by CS1D, CS1, or CJ1.)

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
DISABLE PERIPHERAL SERVICING *2	IOSP @IOSP	287			---
ENABLE PERIPHERAL SERVICING *2	IORS	288			---

Block Programming Instructions

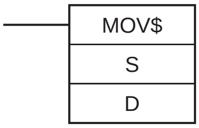
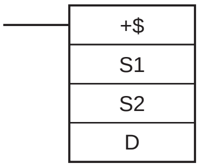
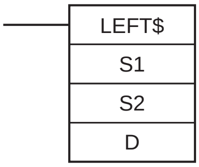
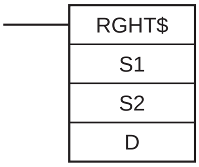
*1: CS1-H, CJ1-H, CJ1M, or CS1D only

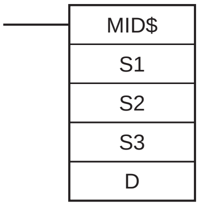
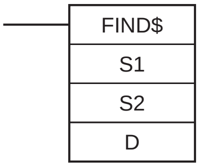
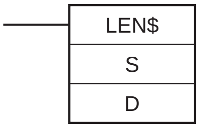
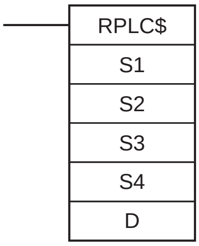
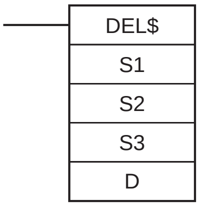
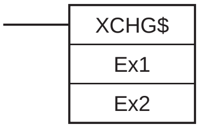
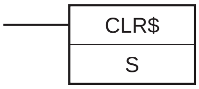
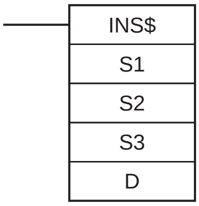
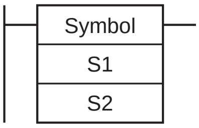
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
BLOCK PROGRAM BEGIN	BPRG	096	Not supported in function blocks	N: Block program number	---
BLOCK PROGRAM END	BEND	801	Not supported in function blocks		---
BLOCK PROGRAM PAUSE	BPPS	811	Not supported in function blocks	N: Block program number	---
BLOCK PROGRAM RESTART	BPRS	812	Not supported in function blocks	N: Block program number	---
CONDITIONAL BLOCK EXIT	CONDITION EXIT	806	Not supported in function blocks		---
CONDITIONAL BLOCK EXIT	EXIT Bit operand	806	Not supported in function blocks	B: Bit operand	---
CONDITIONAL BLOCK EXIT (NOT)	EXIT NOT Bit operand	806	Not supported in function blocks	B: Bit operand	---
CONDITIONAL BLOCK BRANCHING	CONDITION IF	802	Not supported in function blocks		---
CONDITIONAL BLOCK BRANCHING	IF Bit operand	802	Not supported in function blocks	B: Bit operand	---
CONDITIONAL BLOCK BRANCHING (NOT)	IF NOT Bit operand	802	Not supported in function blocks	B: Bit operand	---
CONDITIONAL BLOCK BRANCHING (ELSE)	ELSE	803	Not supported in function blocks		---
CONDITIONAL BLOCK BRANCHING END	IEND	804	Not supported in function blocks		---
ONE CYCLE AND WAIT	CONDITION WAIT	805	Not supported in function blocks		---
ONE CYCLE AND WAIT	WAIT Bit operand	805	Not supported in function blocks	B: Bit operand	---
ONE CYCLE AND WAIT (NOT)	WAIT NOT Bit operand	805	Not supported in function blocks	B: Bit operand	---
TIMER WAIT	TIMW (BCD)	813	Not supported in function blocks	N: Timer number	---
				SV: Set value	---
	TIMWX (BIN) *1	816	Not supported in function blocks	N: Timer number	---
				SV: Set value	---

*1: CS1-H, CJ1-H, CJ1M, or CS1D only

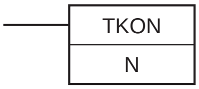
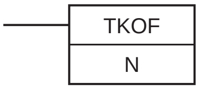
Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
COUNTER WAIT	CNTW (BCD)	814	Not supported in function blocks	N: Counter number	---
				SV: Set value	---
				I: Count input	---
	CNTWX (BIN) *1	817	Not supported in function blocks	N: Counter number	---
				SV: Set value	---
				I: Count input	---
HIGH-SPEED TIMER WAIT	TMHW (BCD)	815	Not supported in function blocks	N: Timer number	---
				SV: Set value	---
	TMHWX (BIN) *1	818	Not supported in function blocks	N: Timer number	---
				SV: Set value	---
LOOP	LOOP	809	Not supported in function blocks		---
LEND	LEND	810	Not supported in function blocks		---
LEND	LEND Bit operand	810	Not supported in function blocks	B: Bit operand	---
LEND NOT	LEND NOT Bit operand	810	Not supported in function blocks	B: Bit operand	---

Text String Processing Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
MOV STRING	MOV$ @MOV$	664		S: 1st source word	Yes
				D: 1st destination word	Yes
CONCATENATE STRING	+$ @+$	656		S1: Text string 1	Yes
				S2: Text string 2	Yes
				D: First destination word	Yes
GET STRING LEFT	LEFT$ @LEFT$	652		S1: Text string first word	Yes
				S2: Number of characters	---
				D: First destination word	Yes
GET STRING RIGHT	RGHT$ @RGHT$	653		S1: Text string first word	Yes
				S2: Number of characters	---
				D: First destination word	Yes

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
GET STRING MIDDLE	MID$ @MID$	654		S1: Text string first word	Yes
				S2: Number of characters	---
				S3: Beginning position	---
				D: First destination word	Yes
FIND IN STRING	FIND$ @FIND$	660		S1: Source text string first word	Yes
				S2: Found text string first word	Yes
				D: First destination word	---
STRING LENGTH	LEN$ @LEN$	650		S: Text string first word	Yes
				D: 1st destination word	---
REPLACE IN STRING	RPLC$ @RPLC$	661		S1: Text string first word	Yes
				S2: Replacement text string first word	Yes
				S3: Number of characters	---
				S4: Beginning position	---
				D: First destination word	Yes
DELETE STRING	DEL$ @DEL$	658		S1: Text string first word	Yes
				S2: Number of characters	---
				S3: Beginning position	---
				D: First destination word	Yes
EXCHANGE STRING	XCHG$ @XCHG$	665		Ex1: 1st exchange word 1	Yes
				Ex2: 1st exchange word 2	Yes
CLEAR STRING	CLR$ @CLR$	666		S: Text string first word	Yes
INSERT INTO STRING	INS$ @INS$	657		S1: Base text string first word	Yes
				S2: Inserted text string first word	Yes
				S3: Beginning position	---
				D: First destination word	Yes
String Comparison	LD,AND, OR + =$,<>$,<$,<=$,>$,>=$	670 (=$) 671 (<>$) 672 (<$) 673 (<=$) 674 (>$) 675 (>=$)		S1: Text string 1	Yes
				S2: Text string 2	Yes

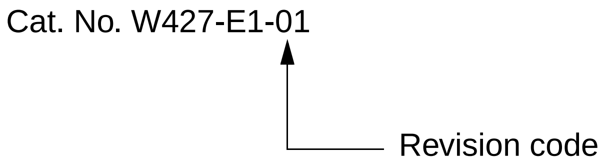
Task Control Instructions

Instruction	Mnemonic	Function code	Symbol	Operands	Array required?
TASK ON	TKON @TKON	820		N: Task number	---
TASK OFF	TKOF @TKOF	821		N: Task number	---

Index

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.



Cat. No. W427-E1-01

Revision code

The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	September 2003	Original production

OMRON CORPORATION
FA Systems Division H.Q.
66 Matsumoto
Mishima-city, Shizuoka 411-8511
Japan
Tel: (81)55-977-9181/Fax: (81)55-977-9045

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, NL-2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
1 East Commerce Drive, Schaumburg, IL 60173
U.S.A.
Tel: (1)847-843-7900/Fax: (1)847-843-8568

OMRON ASIA PACIFIC PTE. LTD.
83 Clemenceau Avenue,
#11-01, UE Square,
Singapore 239920
Tel: (65)6835-3011/Fax: (65)6835-2711

Authorized Distributor:

Cat. No. W427-E1-01

Note: Specifications subject to change without notice

Printed in Japan
0803-??M

Cat. No. W427-E1-01 SYSMAC CX-Programmer IEC Ver, 1.0 (WS02-CPIC1-E) OPERATION MANUAL

OMRON