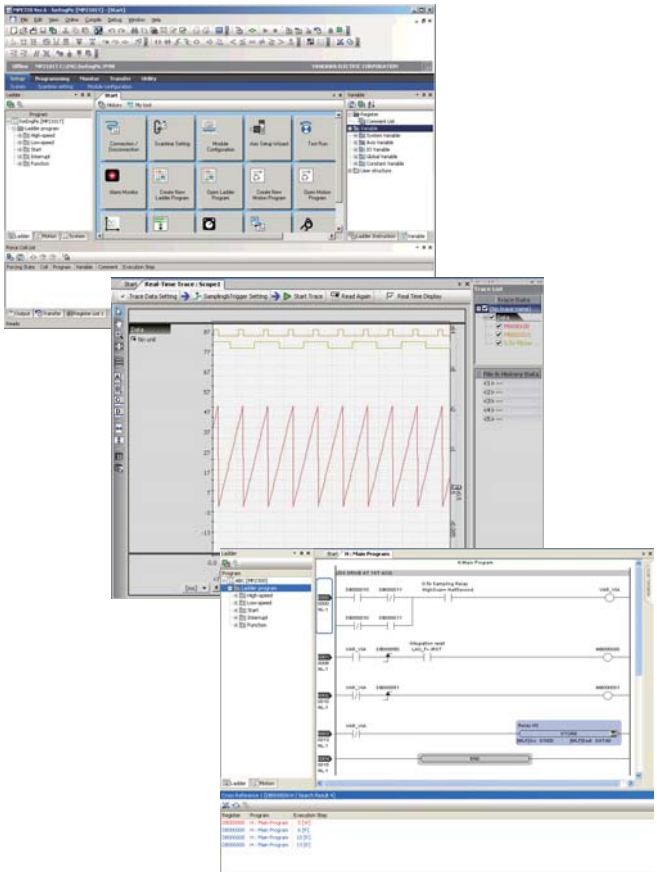
YASKAWA

Machine Controller MP2000 Series

# USER'S MANUAL LADDER PROGRAMMING




Introduction to Ladder Programming 1

Specifications for Ladder Programs 2

Ladder Program Development Flow 3

Programming 4

Instructions 5

Features of the MPE720 Engineering Tool 6

Troubleshooting 7

System Registers AppA

CP (Previous) Ladder Instructions and New Ladder Instructions AppB

Sample Programming AppC

Format for EXPRESSION Instruction AppD

Precautions AppE



MANUAL NO. SIE-C887-1.2E

Copyright © 1998 YASKAWA ELECTRIC CORPORATION

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of Yaskawa. No patent liability is assumed with respect to the use of the information contained herein. Moreover, because Yaskawa 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, Yaskawa 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.

# About this Manual

■ This manual provides comprehensive information on ladder programming for MP2000-series Machine Controllers. It provides the following information on MP2000-series Machine Controllers.

- Introduction to Ladder Programming
- Specifications
- Program Development Flow
- Programming
- Instructions
- MPE720 Engineering Tool
- Troubleshooting

■ This manual provides information on MP2000-series Machine Controllers and MPE720 version 6. For information on the MP900-series Machine Controllers and MPE720 version 5, refer to the appropriate manuals for them.

■ Read this manual carefully to ensure the proper use of the MP2000-series Machine Controllers. Keep this manual in a safe place so that it can be referred to whenever necessary.

# Using this Manual

## ■ Intended Audience

This manual is intended for the following users.

- Those responsible for designing the MP2000-series Machine Controller system
- Those responsible for writing the MP2000-series Machine Controller ladder programs

## ■ MPE720 Engineering Tool Version Number

In this manual, the operation of the MPE720 is described using screen captures of MPE720 version 6.
For this reason, the screen captures and some descriptions may differ for MPE720 version 5.

## ■ Abbreviations

The following abbreviation is used in this manual.

- MP2000: A generic term for the MP2100, MP2100M, MP2101, MP2101M, MP2101T, MP2101TM, MP2200, MP2300, MP2300S, MP2310, MP2400, MP2500/M/B/MB, and MPU-01.

# MP2000-series Manuals

■ The MP2000 Series includes the MP2100, MP2100M, MP2101, MP2101M, MP2101T, MP2101TM, MP2200, MP2300, MP2300S, MP2310, MP2400, MPU-01, MP2500/M/B/MB, and MPU-01. There are many manuals available for one or more of these Machine Controllers. A list of the related manuals is provided on the following page. Refer to these manuals as required.

## ■ Related Manuals

The following manuals are related to the MP2000 Series. Refer to these manuals as required.

| Manual Name | Manual Number | Description |
|---|---|---|
| Machine Controller MP210□/MP210□M User's Manual, Design and Maintenance | SIEP C880700 01 | Describes the functions, specifications, setup procedures, and operating methods of the MP2100/MP2100M. |
| Machine Controller MP2200 User's Manual | SIEP C880700 14 | Describes the functions, specifications, setup procedures, and operating methods of the MP2200. |
| Machine Controller MP2300 Basic Module User's Manual | SIEP C880700 03 | Describes the functions, specifications, setup procedures, and operating methods of the MP2300. |
| Machine Controller MP2300S Basic Module User's Manual | SIEP C880732 00 | Describes the functions, specifications, setup procedures, and operating methods of the MP2300S. |
| Machine Controller MP2310 Basic Module User's Manual | SIEP C880732 01 | Describes the functions, specifications, setup procedures, and operating methods of the MP2310. |
| Machine Controller MP2400 User's Manual | SIEP C880742 00 | Describes the functions, specifications, setup procedures, and operating methods of the MP2400. |
| Machine Controller MP2500/MP2500M/ MP2500D/MP2500MD User's Manual | SIEP C880752 00 | Describes how to use the MP2500, MP2500M, MP2500D, and MP2500MD Machine Controllers. |
| Machine Controller MP2000 Series Built-in SVB/SVB-01 Motion Module User's Manual | SIEP C880700 33 | Describes the SVB Module that is built into an MP2000-series Machine Controller and the SVB-1 Optional Module. |
| Machine Controller MP2000 Series SVC-01 Motion Module User's Manual | SIEP C880700 41 | Describes the SVC-01 SVA Motion Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series SVA-01 Motion Module User's Manual | SIEP C880700 32 | Describes the SVA-01 SVA Motion Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series Pulse Output Motion Module PO-01 User's Manual | SIEP C880700 28 | Describes the PO-01 Pulse Output Motion Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series Communication Module User's Manual | SIEP C880700 04 | Describes the Communications Modules that can be connected to MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series 262IF-01 FL-net Communication Module User's Manual | SIEP C880700 36 | Describes the 262IF-01 FL-net Communications Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series 263IF-01 EtherNet/IP Communication Module User's Manual | SIEP C880700 39 | Describes the 263IF-01 EtherNet/IP Communications Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series I/O Module User's Manual | SIEP C880700 34 | Describes the I/O Modules that can be connected to MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series Analog Input/Analog Output Module AI-01/AO-01 User's Manual | SIEP C880700 26 | Describes the AI-01 Analog Input Module and AO-01 Analog Output Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series Counter Module CNTR-01 User's Manual | SIEP C880700 27 | Describes the CNTR-01 Counter Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series MPU-01 Multiple-CPU Module User's Manual | SIEP C880781 05 | Describes the MPU-01 Multiple-CPU Module for MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series User's Manual for Motion Programming | SIEP C880700 38 | Describes the instructions that are used in motion programming for MP2000-series Machine Controllers. |
| Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual | SIEP C880700 30 | Describes how to install and operate the MPE720 version 6 Engineering Tool for MP2000-series Machine Controllers. |
| Machine Controller MP900/MP2000 Series MPE720 Software for Programming Device User's Manual | SIEP C880700 05 | Describes how to install and operate the MPE720 programming device software for MP900/MP2000-series Machine Controllers. |
| Machine Controller MP2000 Series Embedded C-Language Programming Package Development Guide | SIEP C880700 25 | Describes how to develop, design, and maintain embedded C-language application programs for MP2000-series Machine Controllers. |
| Machine Controller MP900/MP2000 Series New Ladder Editor User's Manual | SIEZ-C887-13.2 | Describes the operating methods of the New Ladder Editor, which assists MP900/MP2000-series design and maintenance. |

| Manual Name | Manual Number | Description |
|---|---|---|
| Machine Controller MP900/MP2000 Series Distributed I/O Module User's Manual, MECHATROLINK System | SIE-C887-5.1 | Describes MECHATROLINK distributed I/O for MP900/ MP2000-series Machine Controllers. |
| Machine Controller MP900/MP2000 Series User's Manual, For Linear Servomotors | SIEP C880700 06 | Describes the connection methods, setting methods, and other information for Linear Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Setup, Rotational Motor | SIEP S800000 43 | Describes the installation, wiring, connections, and trial operation of the Σ-V Series Servo Drives and Rotational Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Setup, Linear Motor | SIEP S800000 44 | Describes the installation, wiring, connections, and trial operation of the Σ-V Series Servo Drives and Linear Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Design and Maintenance, Analog-voltage, Pulse-string Reference, Rotational Motor | SIEP S800000 45 | Describes the design and maintenance of the Σ-V Series Analog Servo Drives and Rotational Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Design and Maintenance, Analog-voltage/ Pulse-string Reference, Linear Motor | SIEP S800000 47 | Describes the design and maintenance of the Σ-V Series Analog Servo Drives and Linear Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Design and Maintenance, MECHATROLINK-II Communications Reference, Rotational Motor | SIEP S800000 46 | Describes the design and maintenance of the Σ-V Series MECHA-TROLINK-II Communications-reference Servo Drives and Rota-tional Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, Design and Maintenance, MECHATROLINK-II Communications Reference, Linear Motor | SIEP S800000 48 | Describes the design and maintenance of the Σ-V Series MECHA-TROLINK-II Communications-reference Servo Drives and Linear Servomotors. |
| AC Servo Drives Σ-V Series User's Manual, MECHATROLINK-II Commands | SIEP S800000 54 | Describes the MECHATROLINK-II communications commands of the Σ-V Series Servo Drives with MECHATROLINK-II com-munications references. |
| AC Servo Drives Σ-V Series User's Manual, Operation of Digital Operator | SIEPS 800000 55 | Describes operating procedures of the Digital Operator for Σ-V Series Servo Drives. |

### ■ Visual Aids

The following visual aids are used to indicate certain types of information for easier reference. Use these to help you understand the different types of information.

IMPORTANT

- Indicates information that must be remembered.
  Also indicates alarm displays and other minor precautions that will not result in machine damage.

- Indicates supplemental information and convenient information to remember.

◀ EXAMPLE ▶

- Indicates concrete examples.

- Indicates definitions of difficult terms or terms that have not been previously explained in this manual.

### ■ Copyrights

- DeviceNet is a registered trademark of the ODVA (Open DeviceNet Venders Association).
- PROFIBUS is a trademark of the PROFIBUS User Organization.
- Ethernet is a registered trademark of the Xerox Corporation.
- MPLINK is a registered trademark of Yaskawa Electric Corporation.
- Microsoft, Windows, Windows NT, and Internet Explorer are trademarks or registered trademarks of the Microsoft Corporation.
- Pentium is a registered trademark of the Intel Corporation.
- MECHATROLINK is a trademark of the MECHATROLINK Members Association.
- Other product names and company names are the trademarks or registered trademarks of the respective company. “TM” and the ® mark do not appear with product or company names in this manual.

## Safety Information

The following signal words and marks are used to indicate safety precautions in this manual. Information marked as shown below is important for safety. Always read this information and heed the precautions that are provided.

⚠ WARNING

Indicates precautions that, if not heeded, could possibly result in loss of life or serious injury.

⚠ CAUTION

Indicates precautions that, if not heeded, could result in relatively serious or minor injury, or property damage.

If not heeded, even precautions classified as cautions ( ⚠ CAUTION ) can lead to serious results depending on circumstances.

⃠ PROHIBITED

Indicates prohibited actions. For example, indicates prohibition of open flame.

MANDATORY

Indicates mandatory actions. For example, indicates that grounding is required.

# Safety Precautions

This section provides important precautions that must be observed in ladder programming. Before you start to program, carefully read all of this manual and all other provided manuals and make sure that you program the MP2000-series Machine Controller correctly. You must be completely familiar with the MP2000-series Machine Controllers, safety information, and all safety precautions before you attempt to use the Machine Controller.

## ■ Storage and Transportation

| ⚠ CAUTION |
|---|
| • If disinfectants or insecticides must be used to treat packing materials such as wooden frames, pallets, or plywood, the packing materials must be treated before the product is packaged, and methods other than fumigation must be used.<br>Example: Heat treatment, where materials are kiln-dried to a core temperature of 56°C for 30 minutes or more.<br>If the electronic products, which include stand-alone products and products installed in machines, are packed with fumigated wooden materials, the electrical components may be greatly damaged by the gases or fumes resulting from the fumigation process. In particular, disinfectants containing halogen, which includes chlorine, fluorine, bromine, or iodine can contribute to the erosion of the capacitors. |

## ■ Other General Precautions

| Observe the following general precautions to ensure safe application. |
|---|
| • The MP2000-series Machine Controllers were not designed or manufactured for use in devices or systems directly related to human life.<br>Users who intend to use products that are described in this manual for special purposes such as devices or systems relating to transportation, medical, space aviation, atomic power control, or underwater use must contact Yaskawa Electric Corporation beforehand.<br>• The MP2000-series Machine Controllers have been manufactured under strict quality control guidelines. However, if an MP2000-series Machine Controller is to be installed in any location in which a failure of the MP2000-series Machine Controllers could involve a life and death situation or in a facility where failure may cause a serious accident, safety devices MUST be installed to minimize the likelihood of any serious accident.<br>• The products shown in illustrations in this manual are sometimes shown without covers or protective guards. Always replace the cover or protective guard as specified first, and then operate the products in accordance with the manual.<br>• The drawings that are presented in this manual are typical examples and may not match the product you received.<br>• If the manual must be ordered due to loss or damage, inform your nearest Yaskawa representative or one of the offices listed on the back of this manual.<br>• Contact your nearest Yaskawa representative or one of the offices listed on the back of this manual to order a new nameplate whenever a nameplate becomes worn or damaged. |

# Warranty

## ( 1 ) Details of Warranty

### ■ Warranty Period

The warranty period for a product that was purchased (hereinafter called “delivered product”) is one year from the time of delivery to the location specified by the customer or 18 months from the time of shipment from the Yaskawa factory, whichever is sooner.

### ■ Warranty Scope

Yaskawa shall replace or repair a defective product free of charge if a defect attributable to Yaskawa occurs during the warranty period above. This warranty does not cover defects caused by the delivered product reaching the end of its service life and replacement of parts that require replacement or that have a limited service life.
This warranty does not cover failures that result from any of the following causes.

1. Improper handling, abuse, or use in unsuitable conditions or in environments not described in product catalogs or manuals, or in any separately agreed-upon specifications
2. Causes not attributable to the delivered product itself
3. Modifications or repairs not performed by Yaskawa
4. Abuse of the delivered product in a manner in which it was not originally intended
5. Causes that were not foreseeable with the scientific and technological understanding at the time of shipment from Yaskawa
6. Events for which Yaskawa is not responsible, such as natural or human-made disasters

## ( 2 ) Limitations of Liability

1. Yaskawa shall in no event be responsible for any damage or loss of opportunity to the customer that arises due to failure of the delivered product.
2. Yaskawa shall not be responsible for any programs (including parameter settings) or the results of program execution of the programs provided by the user or by a third party for use with programmable Yaskawa products.
3. The information described in product catalogs or manuals is provided for the purpose of the customer purchasing the appropriate product for the intended application. The use thereof does not guarantee that there are no infringements of intellectual property rights or other proprietary rights of Yaskawa or third parties, nor does it construe a license.
4. Yaskawa shall not be responsible for any damage arising from infringements of intellectual property rights or other proprietary rights of third parties as a result of using the information described in catalogs or manuals.

## ( 3 ) Suitability for Use

1. It is the customer's responsibility to confirm conformity with any standards, codes, or regulations that apply if the Yaskawa product is used in combination with any other products.
2. The customer must confirm that the Yaskawa product is suitable for the systems, machines, and equipment used by the customer.
3. Consult with Yaskawa to determine whether use in the following applications is acceptable. If use in the application is acceptable, use the product with extra allowance in ratings and specifications, and provide safety measures to minimize hazards in the event of failure.
    - Outdoor use, use involving potential chemical contamination or electrical interference, or use in conditions or environments not described in product catalogs or manuals
    - Nuclear energy control systems, combustion systems, railroad systems, aviation systems, vehicle systems, medical equipment, amusement machines, and installations subject to separate industry or government regulations
    - Systems, machines, and equipment that may present a risk to life or property
    - Systems that require a high degree of reliability, such as systems that supply gas, water, or electricity, or systems that operate continuously 24 hours a day
    - Other systems that require a similar high degree of safety
4. Never use the product for an application involving serious risk to life or property without first ensuring that the system is designed to secure the required level of safety with risk warnings and redundancy, and that the Yaskawa product is properly rated and installed.
5. The circuit examples and other application examples described in product catalogs and manuals are for reference. Check the functionality and safety of the actual devices and equipment to be used before using the product.
6. Read and understand all use prohibitions and precautions, and operate the Yaskawa product correctly to prevent accidental harm to third parties.

## ( 4 ) Specifications Change

The names, specifications, appearance, and accessories of products in product catalogs and manuals may be changed at any time based on improvements and other reasons. The next editions of the revised catalogs or manuals will be published with updated code numbers. Consult with your Yaskawa representative to confirm the actual specifications before purchasing a product.

# Contents


About this Manual - - - - - iii
Using this Manual - - - - - iii
MP2000-series Manuals - - - - - iii
Safety Information - - - - - vi
Safety Precautions - - - - - vii
Warranty - - - - - viii

1 Introduction to Ladder Programming - - - - - 1-1

1.1 What Is a Ladder Program? - - - - - 1-2

1.2 Features of Ladder Programming for MP2000-series Machine Controllers - - - - - 1-3
1.2.1 Types of Ladder Drawings and Their Different Execution Timing - - - - - 1-3
1.2.2 Program Modules - - - - - 1-4
1.2.3 Programming Complicated Numeric Operations - - - - - 1-4
1.2.4 Communications Control with External Devices - - - - - 1-5
1.2.5 Complete Synchronization with Motion Control - - - - - 1-5

2 Specifications for Ladder Programs - - - - - 2-1

2.1 MP2000-series Machine Controller Specifications - - - - - 2-2
2.1.1 Applicable Machine Controllers - - - - - 2-2
2.1.2 Machine Controller Program Specifications - - - - - 2-3

2.2 Engineering Tool Specifications - - - - - 2-4
2.2.1 Applicable Engineering Tool - - - - - 2-4
2.2.2 MPE720 Version 6 Engineering Tool Specifications - - - - - 2-4

2.3 Ladder Programming Instructions - - - - - 2-5

3 Ladder Program Development Flow - - - - - 3-1

3.1 Ladder Program Design Procedures - - - - - 3-2
3.1.1 Connecting the Hardware - - - - - 3-3
3.1.2 Installing MPE720 Version 6 - - - - - 3-4
3.1.3 Communications Settings - - - - - 3-4
3.1.4 System Startup - - - - - 3-4
3.1.5 Creating a Project - - - - - 3-5
3.1.6 Creating Ladder Programs - - - - - 3-6
3.1.7 Transferring Ladder Programs - - - - - 3-9
3.1.8 Checking the Operation of the Ladder Programs - - - - - 3-11
3.1.9 Saving the Ladder Programs to Flash Memory - - - - - 3-14

4 Programming - - - - - 4-1

4.1 Ladder Program Editor - - - - - 4-2

4.2 Ladder Drawings - - - - - 4-3
4.2.1 Types of Ladder Drawings - - - - - 4-3
4.2.2 Controlling the Execution of Drawings - - - - - 4-5

4.3 User Functions - - - - - 4-7
4.3.1 What Is a User Function? - - - - - 4-7
4.3.2 Creating User Functions - - - - - 4-9
4.3.3 Calling a User Function - - - - - 4-12

4.4 Registers (Variables) - - - 4-13
4.4.1 What Are Registers? - - - 4-13
4.4.2 Register Types - - - 4-14
4.4.3 Data Types - - - 4-17
4.4.4 Index Registers (i, j) - - - 4-19

4.5 Table Data - - - 4-21
4.5.1 What Is Table Data? - - - 4-21
4.5.2 Creating Table Data - - - 4-21

4.6 Transferring Data - - - 4-23

4.7 Setting the High-speed/Low-speed Scan Times - - - 4-24

4.8 Advanced Programming - - - 4-25
4.8.1 Motion Programs - - - 4-25
4.8.2 C-language Programs - - - 4-26
4.8.3 Security - - - 4-27
4.8.4 Tracing - - - 4-28

5 Instructions - - - 5-1

5.1 How to Read the Instructions - - - 5-4

5.2 Relay Circuit Instructions - - - 5-5
5.2.1 NO Contact (NOC) - - - 5-5
5.2.2 NC Contact (NCC) - - - 5-6
5.2.3 10-ms ON-Delay Timer (TON[10ms]) - - - 5-7
5.2.4 10-ms OFF-Delay Timer (TOFF[10ms]) - - - 5-9
5.2.5 1-s ON-Delay Timer (TON[1s]) - - - 5-11
5.2.6 1-s OFF-Delay Timer (TOFF[1s]) - - - 5-13
5.2.7 Rising-edge Pulses (ON-PLS) - - - 5-15
5.2.8 Falling-edge Pulses (OFF-PLS) - - - 5-17
5.2.9 Coil (COIL) - - - 5-19
5.2.10 Set Coil (S-COIL) - - - 5-20
5.2.11 Reset Coil (R-COIL) - - - 5-21

5.3 Numeric Operation Instructions - - - 5-22
5.3.1 Store (STORE) - - - 5-22
5.3.2 Add (ADD (+)) - - - 5-24
5.3.3 Extended Add (ADDX (++)) - - - 5-26
5.3.4 Subtract (SUB (–)) - - - 5-28
5.3.5 Extended Subtract (SUBX (– –)) - - - 5-30
5.3.6 Multiply (MUL (x)) - - - 5-32
5.3.7 Divide (DIV (÷)) - - - 5-34
5.3.8 Integer Remainder (MOD) - - - 5-36
5.3.9 Real Remainder (REM) - - - 5-38
5.3.10 Increment (INC) - - - 5-40
5.3.11 Decrement (DEC) - - - 5-42
5.3.12 Add Time (TMADD) - - - 5-44
5.3.13 Subtract Time (TMSUB) - - - 5-46
5.3.14 Spend Time (SPEND) - - - 5-48
5.3.15 Invert Sign (INV) - - - 5-51
5.3.16 One's Complement (COM) - - - 5-52
5.3.17 Absolute Value (ABS) - - - 5-53
5.3.18 Binary Conversion (BIN) - - - 5-54
5.3.19 BCD Conversion (BCD) - - - 5-55
5.3.20 Parity Conversion (PARITY) - - - 5-56
5.3.21 ASCII Conversion 1 (ASCII) - - - 5-57
5.3.22 ASCII Conversion 2 (BINASC) - - - 5-59
5.3.23 ASCII Conversion 3 (ASCBIN) - - - 5-61

5.4 Logic Operations and Comparison Instructions - - - 5-63
5.4.1 Inclusive AND (AND) - - - 5-63
5.4.2 Inclusive OR (OR) - - - 5-65
5.4.3 Exclusive OR (XOR) - - - 5-67
5.4.4 Less Than (<) - - - 5-69
5.4.5 Less Than or Equal (≤) - - - 5-70
5.4.6 Equal (=) - - - 5-71
5.4.7 Not Equal (≠) - - - 5-72
5.4.8 Greater Than or Equal (≥) - - - 5-73
5.4.9 Greater Than (>) - - - 5-74
5.4.10 Range Check (RCHK) - - - 5-75

5.5 Program Control Instructions - - - 5-77
5.5.1 Call Sequence Program (SEE) - - - 5-77
5.5.2 Call Motion Program (MSEE) - - - 5-78
5.5.3 Call User Function (FUNC) - - - 5-80
5.5.4 Direct Input String (INS) - - - 5-81
5.5.5 Direct Output String (OUTS) - - - 5-84
5.5.6 Call Extended Program (XCALL) - - - 5-87
5.5.7 WHILE Construct (WHILE, END_WHILE) - - - 5-88
5.5.8 FOR Construct (FOR, END_FOR) - - - 5-91
5.5.9 IF Construct (IF, END_IF) - - - 5-93
5.5.10 IF-ELSE Construct (IF, ELSE, END_IF) - - - 5-95
5.5.11 Expression (EXPRESSION) - - - 5-97

5.6 Basic Function Instructions - - - 5-99
5.6.1 Square Root (SQRT) - - - 5-99
5.6.2 Sine (SIN) - - - 5-101
5.6.3 Cosine (COS) - - - 5-103
5.6.4 Tangent (TAN) - - - 5-105
5.6.5 Arc Sine (ASIN) - - - 5-106
5.6.6 Arc Cosine (ACOS) - - - 5-107
5.6.7 Arc Tangent (ATAN) - - - 5-108
5.6.8 Exponential (EXP) - - - 5-109
5.6.9 Natural Logarithm (LN) - - - 5-110
5.6.10 Common Logarithm (LOG) - - - 5-111

5.7 Data Shift Instructions - - - 5-112
5.7.1 Bit Rotate Left (ROTL) - - - 5-112
5.7.2 Bit Rotate Right (ROTR) - - - 5-114
5.7.3 Move Bit (MOVB) - - - 5-116
5.7.4 Move Word (MOVW) - - - 5-118
5.7.5 Exchange (XCHG) - - - 5-120
5.7.6 Table Initialization (SETW) - - - 5-122
5.7.7 Byte-to-word Expansion (BEXTD) - - - 5-124
5.7.8 Word-to-byte Compression (BPRESS) - - - 5-126
5.7.9 Binary Search (BSRCH) - - - 5-128
5.7.10 Sort (SORT) - - - 5-130
5.7.11 Bit Shift Left (SHFTL) - - - 5-132
5.7.12 Bit Shift Right (SHFTR) - - - 5-134
5.7.13 Copy Word (COPYW) - - - 5-136
5.7.14 Byte Swap (BSWAP) - - - 5-138

5.8 DDC Instructions - - - 5-139
5.8.1 Dead Zone A (DZA) - - - 5-139
5.8.2 Dead Zone B (DZB) - - - 5-141
5.8.3 Upper/Lower Limit (LIMIT) - - - 5-143
5.8.4 PI Control (PI) - - - 5-145
5.8.5 PD Control (PD) - - - 5-150
5.8.6 PID Control (PID) - - - 5-156

5.8.7 First-order Lag (LAG) - - - - - 5-161
5.8.8 Phase Lead Lag (LLAG) - - - - - 5-164
5.8.9 Function Generator (FGN) - - - - - 5-167
5.8.10 Inverse Function Generator (IFGN) - - - - - 5-172
5.8.11 Linear Accelerator/Decelerator 1 (LAU) - - - - - 5-177
5.8.12 Linear Accelerator/Decelerator 2 (SLAU) - - - - - 5-184
5.8.13 Pulse Width Modulation (PWM) - - - - - 5-194

5.9 Table Manipulation Instructions - - - - - 5-197
5.9.1 Read Table Block (TBLBR) - - - - - 5-197
5.9.2 Write Table Block (TBLBW) - - - - - 5-200
5.9.3 Search for Table Row (TBLSRL) - - - - - 5-203
5.9.4 Search for Table Column (TBLSRC) - - - - - 5-206
5.9.5 Clear Table Block (TBLCL) - - - - - 5-209
5.9.6 Move Table Block (TBLMV) - - - - - 5-212
5.9.7 Read Queue Table (QTBLR and QTBLRI) - - - - - 5-215
5.9.8 Write Queue Table (QTBLW and QTBLWI) - - - - - 5-219
5.9.9 Clear Queue Table Pointers (QTBLCL) - - - - - 5-223

5.10 System Function Instructions - - - - - 5-225
5.10.1 Counter (COUNTER) - - - - - 5-225
5.10.2 First-in First-out (FINFOUT) - - - - - 5-228
5.10.3 Trace (TRACE) - - - - - 5-232
5.10.4 Read Data Trace (DTRC-RD) - - - - - 5-234
5.10.5 Read Inverter Trace (ITRC-RD) - - - - - 5-238
5.10.6 Send Message (MSG-SND) - - - - - 5-241
5.10.7 Receive Message (MSG-RCV) - - - - - 5-253
5.10.8 Write Inverter Parameter (ICNS-WR) - - - - - 5-261
5.10.9 Read Inverter Parameter (ICNS-RD) - - - - - 5-266
5.10.10 Write SERVOPACK Parameter (MLNK-SVW) - - - - - 5-270
5.10.11 Write Motion Register (MOTREG-W) - - - - - 5-275
5.10.12 Read Motion Register (MOTREG-R) - - - - - 5-278

5.11 C-language Control Instructions - - - - - 5-281
5.11.1 Call C-language Function (C-FUNC) - - - - - 5-281
5.11.2 C-language Task Control (TSK-CTRL) - - - - - 5-283

6 Features of the MPE720 Engineering Tool - - - - - 6-1
6.1 Ladder Program Runtime Monitoring - - - - - 6-2
6.2 Searching/Replacing - - - - - 6-3
6.3 Cross References - - - - - 6-4
6.4 Checking for Multiple Coils - - - - - 6-5
6.5 Forcing Coils ON and OFF - - - - - 6-5
6.6 Viewing Called Programs - - - - - 6-6
6.7 Register Lists - - - - - 6-6
6.8 Tuning Panel - - - - - 6-7
6.9 Enabling and Disabling Ladder Programs - - - - - 6-8
6.10 Compiling for MPE720 Version 5 - - - - - 6-9

7 Troubleshooting - - - - - 7-1
7.1 Basic Flow of Troubleshooting - - - - - 7-2

7.2 Indicator Status - - - - - - - - - - 7-3

7.3 Problem Classifications - - - - - - - - - - 7-4

7.3.1 Overview - - - - - - - - - - 7-4
7.3.2 Error Checking Flowchart for MP2000-series Machine Controllers - - - - - - - - - - 7-5

7.4 Detailed Troubleshooting - - - - - - - - - - 7-6

7.4.1 Operation Errors - - - - - - - - - - 7-6
7.4.2 I/O Errors - - - - - - - - - - 7-9
7.4.3 Watchdog Timer Errors - - - - - - - - - - 7-10
7.4.4 Module Synchronization Errors - - - - - - - - - - 7-10
7.4.5 System Errors - - - - - - - - - - 7-11

Appendix A System Registers - - - - - - - - - - A-1

A.1 System Service Registers - - - - - - - - - - A-2

A.2 System Status - - - - - - - - - - A-6

A.3 System Error Status - - - - - - - - - - A-7

A.4 Overview of User Operation Error Status - - - - - - - - - - A-9

A.5 System Service Execution Status - - - - - - - - - - A-11

A.6 Detailed User Operation Error Status - - - - - - - - - - A-11

A.7 System I/O Error Status - - - - - - - - - - A-12

A.8 CF Card-related System Registers
(MP2200-series CPU-02 and CPU-03 only) - - - - - - - - - - A-13

A.9 Interrupt Status - - - - - - - - - - A-14

A.9.1 Interrupt Status List - - - - - - - - - - A-14
A.9.2 Details on Interrupting Module - - - - - - - - - - A-14

A.10 Module Information - - - - - - - - - - A-15

A.11 MPU-01 System Status - - - - - - - - - - A-16

A.12 Motion Program Information - - - - - - - - - - A-17

Appendix B CP (Previous) Ladder Instructions and New Ladder Instructions - - - B-1

B.1 Correspondence between CP (Previous) Ladder Instructions and New Ladder
Instructions - - - - - - - - - - B-2

B.2 Converting CP (Previous) Ladder Programs to New Ladder Programs - - - - - - B-3

Appendix C Sample Programming - - - - - - - - - - C-1

C.1 Jogging from the Control Panel - - - - - - - - - - C-2

C.2 Motion Program Control - - - - - - - - - - C-3

C.3 Simple Synchronized Operation of Two Axes with a Virtual Axis - - - - - - - - - - C-4

C.4 Transferring Project Files to Different Models - - - - - - - - - - C-6

Appendix D Format for EXPRESSION Instruction - - - - - - - - - - D-1

D.1 Elements That You Can Use in Numeric Expressions - - - - - - - - - - D-2

D.2 National Limitations ---------- D-5
D.2.1 Arithmetic Operators ---------- D-5
D.2.2 Comparison Operators ---------- D-5
D.2.3 Logic Operators ---------- D-5
D.2.4 Substitution Operator ---------- D-6
D.2.5 Functions ---------- D-6
D.2.6 Others ---------- D-6

Appendix E Precautions ---------- E-1

E.1 General Precautions ---------- E-2

E.2 Precautions on Motion Parameters ---------- E-2

Index ---------- Index-1

Revision History

# 1

# Introduction to Ladder Programming

This chapter gives an overview of ladder programming and describes its features.

1.1 What Is a Ladder Program? - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-2

1.2 Features of Ladder Programming for MP2000-series Machine Controllers - 1-3

1.2.1 Types of Ladder Drawings and Their Different Execution Timing - - - - - - - - - - - - - - - - 1-3

1.2.2 Program Modules - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-4

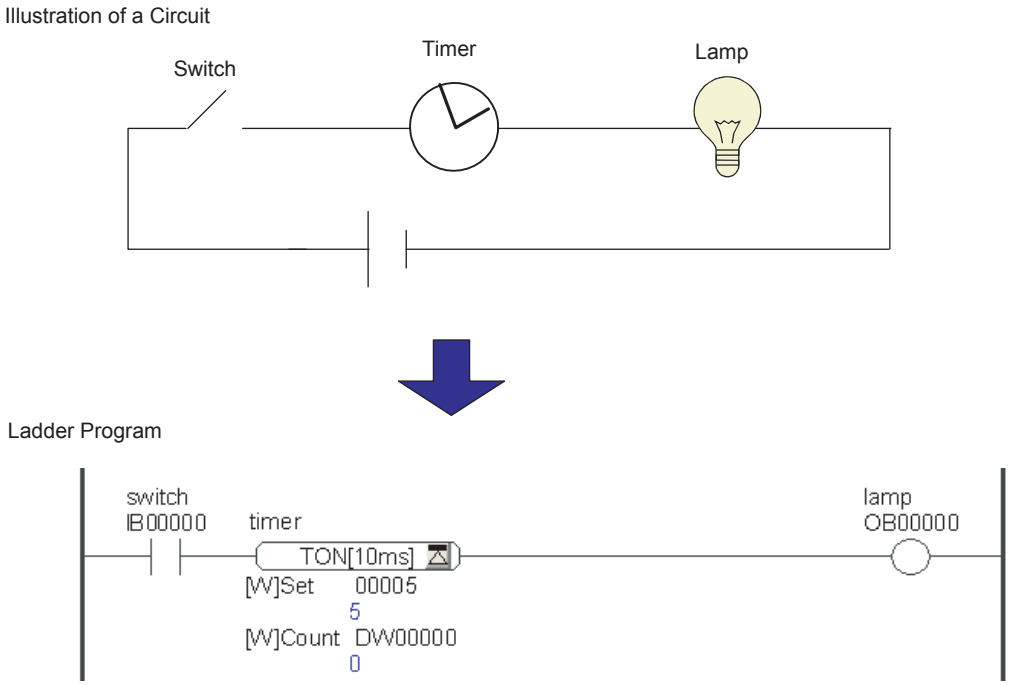
1.2.3 Programming Complicated Numeric Operations - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-4

1.2.4 Communications Control with External Devices - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-5

1.2.5 Complete Synchronization with Motion Control - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-5

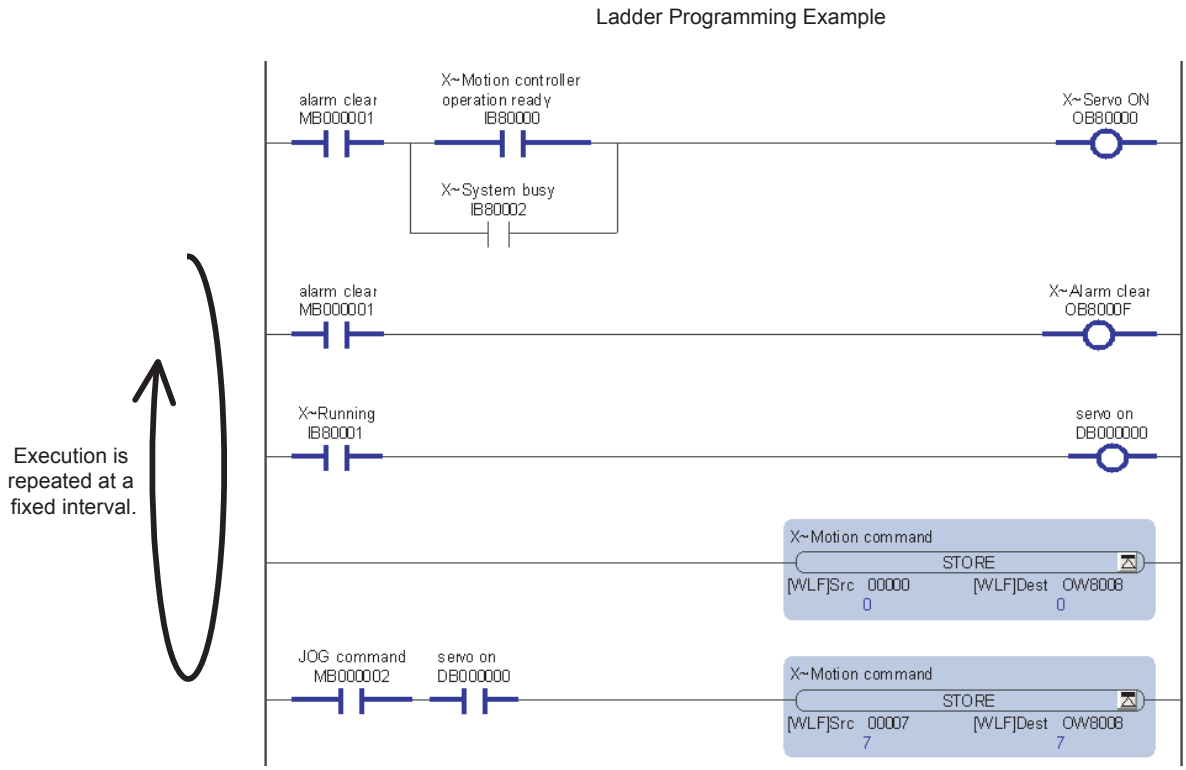
# 1.1 What Is a Ladder Program?

A ladder program uses ladder instructions and registers to symbolically represent electrical circuits that consist of switches, timers, lamps, and other devices.




Ladder programming allows you to easily program large, complex circuits.

Each of the ladder programs that you create is executed in a single scan and then executed repeatedly at fixed intervals.

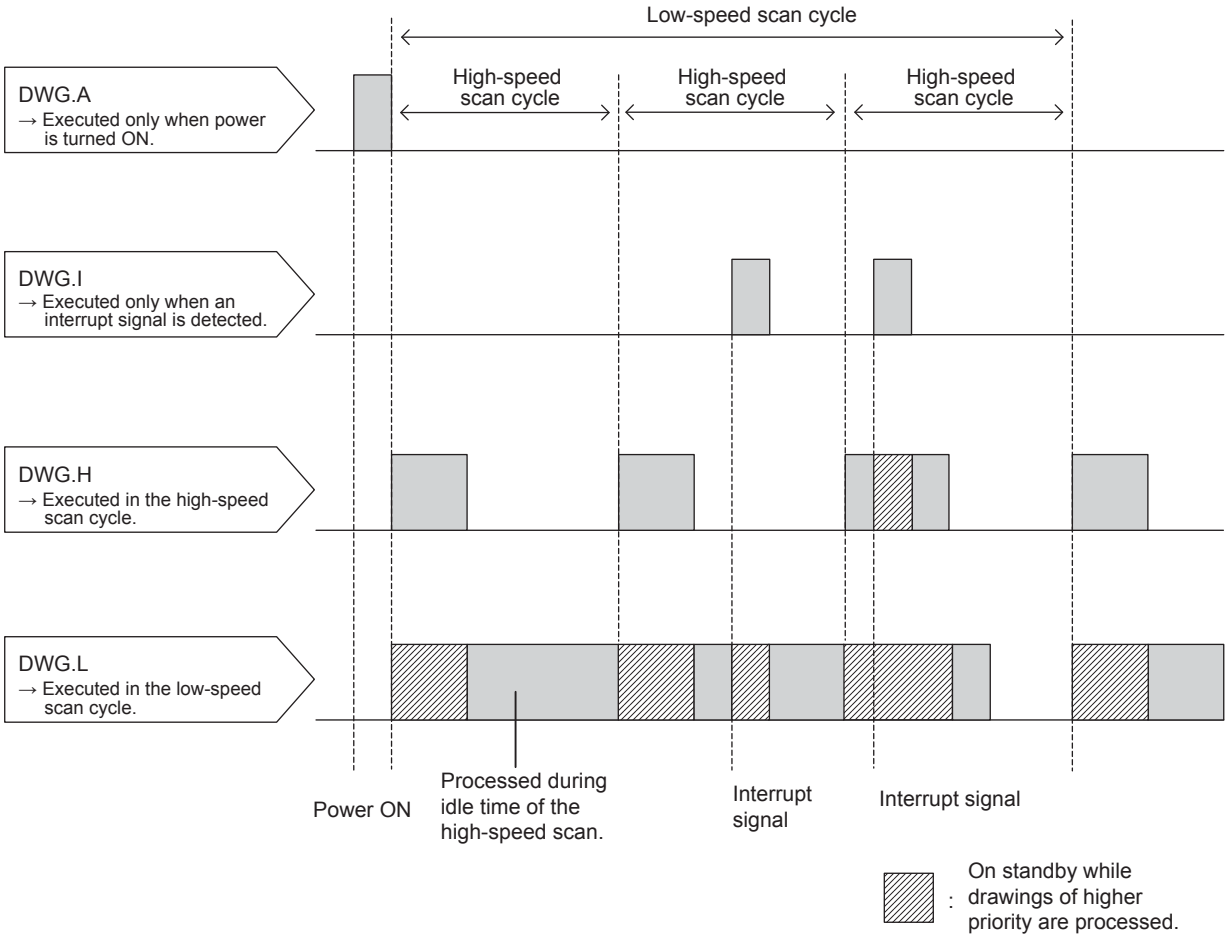


# 1.2 Features of Ladder Programming for MP2000-series Machine Controllers

This section describes the features of ladder programming.

## 1.2.1 Types of Ladder Drawings and Their Different Execution Timing

Ladder programs are managed in units of drawings (DWG). These are called ladder drawings.
In the MP2000-series Machine Controllers, ladder drawings are executed at various times, as illustrated in the following figure.
Processing can be executed at the appropriate time by programming it in the appropriate ladder drawing.



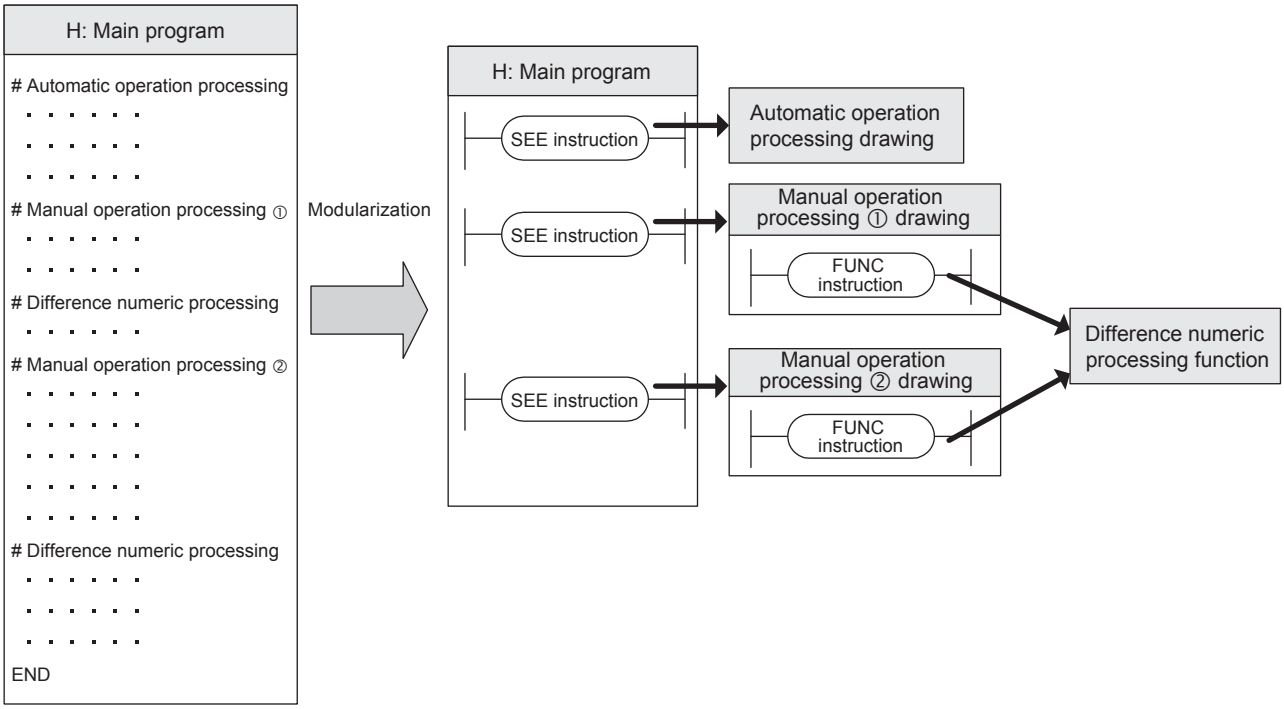

### ■ Drawing Execution Timing

| Priority | Ladder Drawing | Execution Timing (Processing Example) |
|---|---|---|
| 1 (High) | DWG.A | This drawing is executed only once when the power supply is turned ON (e.g., for data initialization). |
| 2 (↑) | DWG.I | This drawing is executed when an interrupt signal is detected (e.g., for interrupt processing for external signals). |
| 3 (↑) | DWG.H | This drawing is executed every high-speed scan cycle (e.g., for motion control). |
| 4 (Low) | DWG.L | This drawing is executed every low-speed scan cycle (e.g., for touch panel display processing). |

- The drawings with lower numbers have higher execution priority.

## 1.2.2 Program Modules

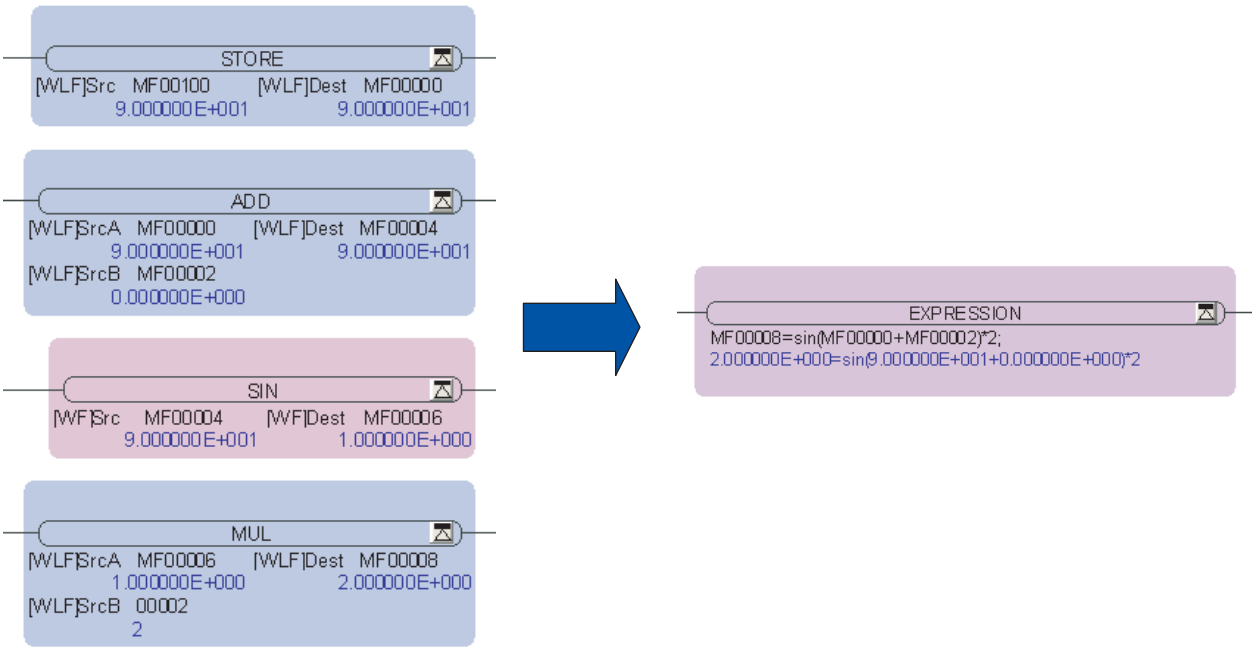
The main program can be separated into modular units to suit different processing requirements, such as child drawings, grandchild drawings, and functions, to make the program easier to read.




## 1.2.3 Programming Complicated Numeric Operations

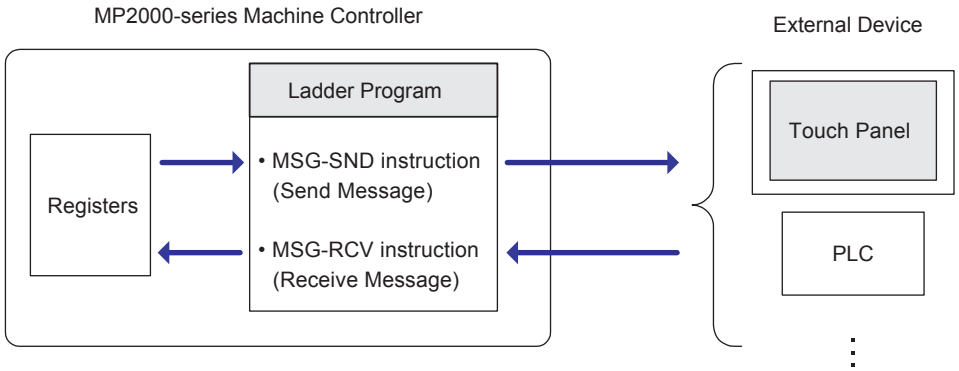
Complicated calculations written over several lines can be written easily within a single EXPRESSION instruction. Variables, structures, and basic functions, such as those for sine and cosine calculations, can be programmed using familiar C-like expressions.

You can display the current value inside expressions in the same way as you can for other ladder language instructions.



## 1.2.4 Communications Control with External Devices

The MSG-SND and MSG-RCV ladder instructions support various protocols and can be used to control communications with many external devices, such as a touch panels or host PLCs. This allows external devices to access registers in the Machine Controller.

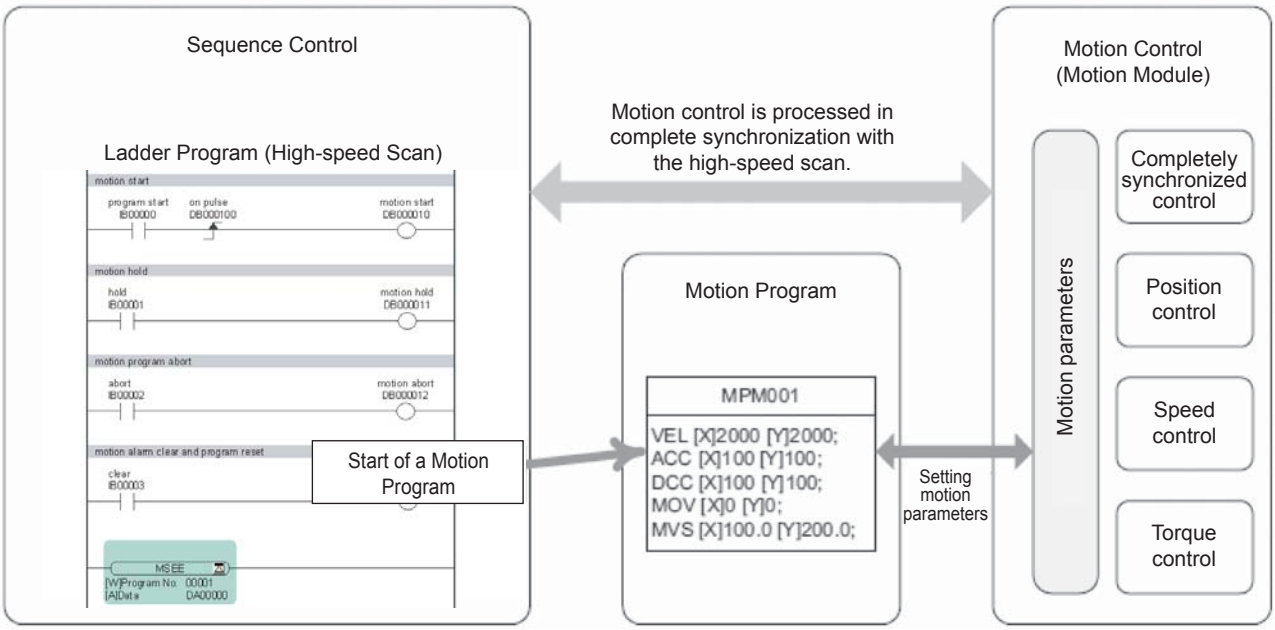



INFO

Instead of using a ladder program, the Machine Controller can also communicate with external devices by using I/O message communications or automatic reception.
Refer to *Chapter 6 Ethernet Communications* in the *Machine Controller MP2310 Basic Module User's Manual* (Manual No.: SIEP C880732 01) for details.

## 1.2.5 Complete Synchronization with Motion Control

Ladder programs that are started in the high-speed scan are processed in complete synchronization with motion control processing. This allows you to call and process a motion program that performs complicated motion control synchronously with a ladder program.



# 2

# Specifications for Ladder Programs

This chapter gives the specifications for ladder programs.


2.1 MP2000-series Machine Controller Specifications ----------2-2
2.1.1 Applicable Machine Controllers ---------- 2-2
2.1.2 Machine Controller Program Specifications ---------- 2-3
2.2 Engineering Tool Specifications ----------2-4
2.2.1 Applicable Engineering Tool ---------- 2-4
2.2.2 MPE720 Version 6 Engineering Tool Specifications ---------- 2-4
2.3 Ladder Programming Instructions ----------2-5

# 2.1 MP2000-series Machine Controller Specifications

## 2.1.1 Applicable Machine Controllers

You can use ladder programs with the following MP2000-series Machine Controllers.

- MP2100
- MP2100M
- MP2101
- MP2101M
- MP2101T
- MP2101TM
- MP2200 with CPU-01
- MP2200 with CPU-02
- MP2200 with CPU-03
- MP2200 with CPU-04
- MP2300
- MP2300S
- MP2310
- MP2500
- MP2500B
- MP2500M
- MP2500MB
- MPU-01

The MP2400 supports only motion programs and sequence programs.
You cannot use ladder programs with it.

## 2.1.2 Machine Controller Program Specifications

| Machine Controller | | MP2100 and MP2100M | MP2300 | MP2300S | MP2200 with CPU-01 | MP2310 | MP2200 with CPU-02 | MP2101, MP2101M, MP2101T, MP2101TM, MP2200 with CPU-03, MP2200 with CPU-04, and MPU-01 |
|---|---|---|---|---|---|---|---|---|
| Program Capacity*1 | | 5.5 MB | | | 7.5 MB | | 11.5 MB | |
| Ladder Programs | Applicable Models | Applicable*2 | NA | Applicable | | | | |
| | Startup Processing | 64 drawings max. including parent drawings, operation error drawings, child drawings, and grandchild drawings | | | | | | |
| | Interrupt Processing | 64 drawings max. including parent drawings, operation error drawings, child drawings, and grandchild drawings | | | | | | |
| | High-speed Scan Processing | 200 drawings max. including parent drawings, operation error drawings, child drawings, and grandchild drawings | | | | | | |
| | Low-speed Scan Processing | 500 drawings max. including parent drawings, operation error drawings, child drawings, and grandchild drawings | | | | | | |
| | User Functions | 500 drawings max. | | | | | | |
| | Maximum Number of Steps | 1,000 steps per drawing | | | | | | |
| Motion Programs | Applicable Models | Applicable | | | | | | |
| | Number of Programs | 256 programs max. including motion programs and sequence programs | | | | | | |
| | Number of Groups | 8 groups (Up to 16 axes can be set in one group.) | | | | | | |
| | Number of Tasks | 16 tasks max. (This is the number of simultaneously executable motion programs.) | | | | | | |
| | Number of Parallel Forks per Task | 8 (4 main program forks × 2 subprogram forks) | | | | | | |
| Sequence Programs | Applicable Models | NA | Applicable | Applicable | NA | Applicable | NA | Applicable |
| | Number of Programs | 256 programs max. including motion programs and sequence programs | | | | | | |
| | Number of Tasks | 16 tasks max. (This is the number of simultaneously executable sequence programs.) | | | | | | |
| Accessible Registers | M Registers | Applicable (65,535 words)<br>These registers are backed up with a battery.*3 | | | | | | |
| | S Registers | Applicable (8,192 words)<br>These registers are backed up with a battery.*3 | | | | | | |
| | I Registers | Applicable (32,768 words + motion monitor parameters) | | | | | | |
| | O Registers | Applicable (32,768 words + motion setting parameters) | | | | | | |
| | C Registers | Applicable (16,384 words) | | | | | | |
| | D Registers | Applicable (Can be specified to between 0 and 16,384 words.)<br>These registers are unique to each drawing (DWG). They can be used within each drawing. | | | | | | |
| | # Registers | Applicable (Can be specified to between 0 and 16,384 words.)<br>These are internal registers that are unique to each drawing (DWG). They can be referenced within each drawing. | | | | | | |
| Capacity of Table Data Backed Up by a Battery | | None | | | | 1 MB | | 3 MB |

∗ 1. This is the total capacity for ladder programs and motion programs.
∗ 2. This is supported only for version 2.66 or higher.
∗ 3. The # registers can be used only when ladder programs are used.

# 2.2 Engineering Tool Specifications

This section gives the specifications for programs for the Engineering Tool.

## 2.2.1 Applicable Engineering Tool

You can create ladder programs with the following Engineering Tool.

- MPE720 version 5 for all MP2000-series Machine Controllers except for the MP2400
- MPE720 version 6 for all MP2000-series Machine Controllers

In addition to the Engineering Tool, you can also use the following Support Tools to monitor Machine Controller information and transfer data.

- MPLOGGER (Control Information Monitoring Tool)
- MPLoader (Data Transfer Tool)
- MPLoadMaker (Automatic Transfer Data Creation Tool)

You can install the Engineering Tool and Support Tools in one PC to use them.

## 2.2.2 MPE720 Version 6 Engineering Tool Specifications

The following table shows the relationship between the Engineering Tool and the Machine Controller.

| Machine Controller | MPE720 Version 6 (CPMC-MPE770) | Remarks |
|---|---|---|
| MP2100 | Applicable | – |
| MP2100M | Applicable | – |
| MP2101 | Applicable | Applicable with MPE720 version 6.24 or higher |
| MP2101M | Applicable | Applicable with MPE720 version 6.24 or higher |
| MP2101T | Applicable | Applicable with MPE720 version 6.24 or higher |
| MP2101TM | Applicable | Applicable with MPE720 version 6.24 or higher |
| MP2200 with CPU-01 | Applicable | – |
| MP2200 with CPU-02 | Applicable | – |
| MP2200 with CPU-03 | Applicable | Applicable with MPE720 version 6.20 or higher |
| MP2200 with CPU-04 | Applicable | Applicable with MPE720 version 6.22 or higher |
| MP2300 | Applicable | – |
| MP2300S | Applicable | Applicable with MPE720 version 6.04 or higher |
| MP2310 | Applicable | Applicable with MPE720 version 6.04 or higher |
| MPU-01 | Applicable | Applicable with MPE720 version 6.23 or higher |

The following table shows the relationship between the Engineering Tool and the programs.

| Program | MPE720 Version 6 (CPMC-MPE770) | Remarks |
|---|---|---|
| Ladder Programs | Applicable | – |
| Motion Programs | Applicable | – |
| Sequence Programs | Applicable | – |

# 2.3 Ladder Programming Instructions

The following table lists the ladder programming instructions.

Refer to the reference sections for details on individual instructions.

| Type | Symbol | Function | Reference |
|---|---|---|---|
| Relay Circuit Instructions | NOC | NO Contact | 5.2.1 |
| | NCC | NC Contact | 5.2.2 |
| | TON[10 ms] | 10-ms ON-Delay Timer | 5.2.3 |
| | TOFF[10 ms] | 10-ms OFF-Delay Timer | 5.2.4 |
| | TON[1 s] | 1-s ON-Delay Timer | 5.2.5 |
| | TOFF[1 s] | 1-s OFF-Delay Timer | 5.2.6 |
| | ON-PLS | Rising-edge Pulses | 5.2.7 |
| | OFF-PLS | Falling-edge Pulses | 5.2.8 |
| | COIL | Coil | 5.2.9 |
| | S-COIL | Set Coil | 5.2.10 |
| | R-COIL | Reset Coil | 5.2.11 |
| Numeric Operation Instructions | STORE | Store | 5.3.1 |
| | ADD | Add | 5.3.2 |
| | ADDX | Extended Add | 5.3.3 |
| | SUB | Subtract | 5.3.4 |
| | SUBX | Extended Subtract | 5.3.5 |
| | MUL | Multiply | 5.3.6 |
| | DIV | Divide | 5.3.7 |
| | MOD | Integer Remainder | 5.3.8 |
| | REM | Real Remainder | 5.3.9 |
| | INC | Increment | 5.3.10 |
| | DEC | Decrement | 5.3.11 |
| | TMADD | Add Time | 5.3.12 |
| | TMSUB | Subtract Time | 5.3.13 |
| | SPEND | Spend Time | 5.3.14 |
| | INV | Invert Sign | 5.3.15 |
| | COM | One’s Complement | 5.3.16 |
| | ABS | Absolute Value | 5.3.17 |
| | BIN | Binary Conversion | 5.3.18 |
| | BCD | BCD Conversion | 5.3.19 |
| | PARITY | Parity Conversion | 5.3.20 |
| | ASCII | ASCII Conversion 1 | 5.3.21 |
| | BINASC | ASCII Conversion 2 | 5.3.22 |
| | ASCBIN | ASCII Conversion 3 | 5.3.23 |
| Logic Operation Instructions | AND | Inclusive AND | 5.4.1 |
| | OR | Inclusive OR | 5.4.2 |
| | XOR | Exclusive OR | 5.4.3 |
| | < | Less Than | 5.4.4 |
| | ≤ | Less Than or Equal | 5.4.5 |
| | = | Equal | 5.4.6 |
| | ≠ | Not Equal | 5.4.7 |
| | ≥ | Greater Than or Equal | 5.4.8 |
| | > | Greater Than | 5.4.9 |
| | RCHK | Range Check | 5.4.10 |

| Type | Symbol | Function | Reference |
|---|---|---|---|
| Program Control Instructions | SEE | Call Sequence Subprogram | 5.5.1 |
| | MSEE | Call Motion Program | 5.5.2 |
| | FUNC | Call User Function | 5.5.3 |
| | INS | Direct Input String | 5.5.4 |
| | OUTS | Direct Output String | 5.5.5 |
| | XCALL | Call Extended Program | 5.5.6 |
| | WHILE<br>END_WHILE | WHILE construct | 5.5.7 |
| | FOR<br>END_FOR | FOR construct | 5.5.8 |
| | IF<br>END_IF | IF construct | 5.5.9 |
| | IF<br>ELSE<br>END_IF | IF ELSE construct | 5.5.10 |
| | EXPRESSION | Expression | 5.5.11 |
| Basic Function Instructions | SQRT | Square Root | 5.6.1 |
| | SIN | Sine | 5.6.2 |
| | COS | Cosine | 5.6.3 |
| | TAN | Tangent | 5.6.4 |
| | ASIN | Arc Sine | 5.6.5 |
| | ACOS | Arc Cosine | 5.6.6 |
| | ATAN | Arc Tangent | 5.6.7 |
| | EXP | Exponential | 5.6.8 |
| | LN | Natural Logarithm | 5.6.9 |
| | LOG | Common Logarithm | 5.6.10 |
| Data Manipulation Instructions | ROTL | Bit Rotate Left | 5.7.1 |
| | ROTR | Bit Rotate Right | 5.7.2 |
| | MOVB | Move Bit | 5.7.3 |
| | MOVW | Move Word | 5.7.4 |
| | XCHG | Exchange | 5.7.5 |
| | SETW | Table Initialization | 5.7.6 |
| | BEXTD | Byte-to-word Expansion | 5.7.7 |
| | BPRESS | Word-to-byte Compression | 5.7.8 |
| | BSRCH | Binary Search | 5.7.9 |
| | SORT | Sort | 5.7.10 |
| | SHFTL | Bit Shift Left | 5.7.11 |
| | SHFTR | Bit Shift Right | 5.7.12 |
| | COPYW | Copy Word | 5.7.13 |
| | BSWAP | Byte Swap | 5.7.14 |
| DDC Instructions | DZA | Dead Zone A | 5.8.1 |
| | DZB | Dead Zone B | 5.8.2 |
| | LIMIT | Upper/Lower Limit | 5.8.3 |
| | PI | PI Control | 5.8.4 |
| | PD | PD Control | 5.8.5 |
| | PID | PID Control | 5.8.6 |
| | LAG | First Order Lag | 5.8.7 |
| | LLAG | Phase Lead Lag | 5.8.8 |
| | FGN | Function Generator | 5.8.9 |
| | IFGN | Inverse Function Generator | 5.8.10 |
| | LAU | Linear Accelerator/Decelerator 1 | 5.8.11 |
| | SLAU | Linear Accelerator/Decelerator 2 | 5.8.12 |
| | PWM | Pulse Width Modulation | 5.8.13 |

| Type | Symbol | Function | Reference |
|---|---|---|---|
| Table Manipulation Instructions | TBLBR | Read Table Block | 5.9.1 |
| | TBLBW | Write Table Block | 5.9.2 |
| | TBLSRL | Search Table Row | 5.9.3 |
| | TBLSRC | Search Table Column | 5.9.4 |
| | TBLCL | Clear Table Block | 5.9.5 |
| | TBLMV | Move Table Block | 5.9.6 |
| | QTBLR | Read Queue Table | 5.9.7 |
| | QTBLRI | Read Queue Table with Pointer Increment | 5.9.7 |
| | QTBLW | Write Queue Table | 5.9.8 |
| | QTBLWI | Write Queue Table with Pointer Increment | 5.9.8 |
| | QTBLCL | Clear Queue Table Pointer | 5.9.9 |
| Standard System Function Instructions | COUNTER | Counter | 5.10.1 |
| | FINFOUT | First-in First-out | 5.10.2 |
| | TRACE | Trace | 5.10.3 |
| | DTRC-RD | Read Data Trace | 5.10.4 |
| | ITRC-RD | Read Inverter Trace | 5.10.5 |
| | MSG-SND | Send Message | 5.10.6 |
| | MSG-RCV | Receive Message | 5.10.7 |
| | ICNS-WR | Write Inverter Parameters | 5.10.8 |
| | ICNS-RD | Read Inverter Parameters | 5.10.9 |
| | MLNK-SVW | Write SERVOPACK Parameters | 5.10.10 |
| | MOTREG-W | Write Motion Register | 5.10.11 |
| | MOTREG-R | Read Motion Register | 5.10.12 |
| C-language Control Instructions | C-FUNC | Call User C-language Function | 5.11.1 |
| | TSK-CTRL | Control User C-language Task | 5.11.2 |

# 3

# Ladder Program Development Flow

This chapter describes the development flow for ladder programs.


3.1 Ladder Program Design Procedures - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-2
3.1.1 Connecting the Hardware - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-3
3.1.2 Installing MPE720 Version 6 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-4
3.1.3 Communications Settings - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-4
3.1.4 System Startup - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-4
3.1.5 Creating a Project - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-5
3.1.6 Creating Ladder Programs - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-6
3.1.7 Transferring Ladder Programs - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-9
3.1.8 Checking the Operation of the Ladder Programs - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-11
3.1.9 Saving the Ladder Programs to Flash Memory - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 3-14

# 3.1 Ladder Program Design Procedures

This section describes the design procedures for ladder programs as outlined below.

**① Preparation for Devices to Be Connected**

Assemble and wire all devices to be connected.
Install MPE720 on a PC.

Refer to *3.1.1 Connecting the Hardware*.
Refer to *3.1.2 Installing MPE720 Version 6*.
Refer to *3.1.3 Communications Settings*.

**② System Startup**

Perform self configuration and start the system.

Refer to *3.1.4 System Startup*.

**③ Creating a Project**

Create a project before you start ladder program development.

Refer to *3.1.5 Creating a Project*.

**④ Creating Ladder Programs**

Enter the ladder programs in the Ladder Editor.

Refer to *3.1.6 Creating Ladder Programs*.

**⑤ Transferring Ladder Programs**

Transfer the ladder programs that you created to the MP2000-series Machine Controllers.

Refer to *3.1.7 Transferring Ladder Programs*.

**⑥ Checking the Operation of the Ladder Programs**

Check the operation of the ladder programs.

Refer to *3.1.8 Checking the Operation of the Ladder Programs*.

**⑦ Saving the Ladder Programs to Flash Memory**

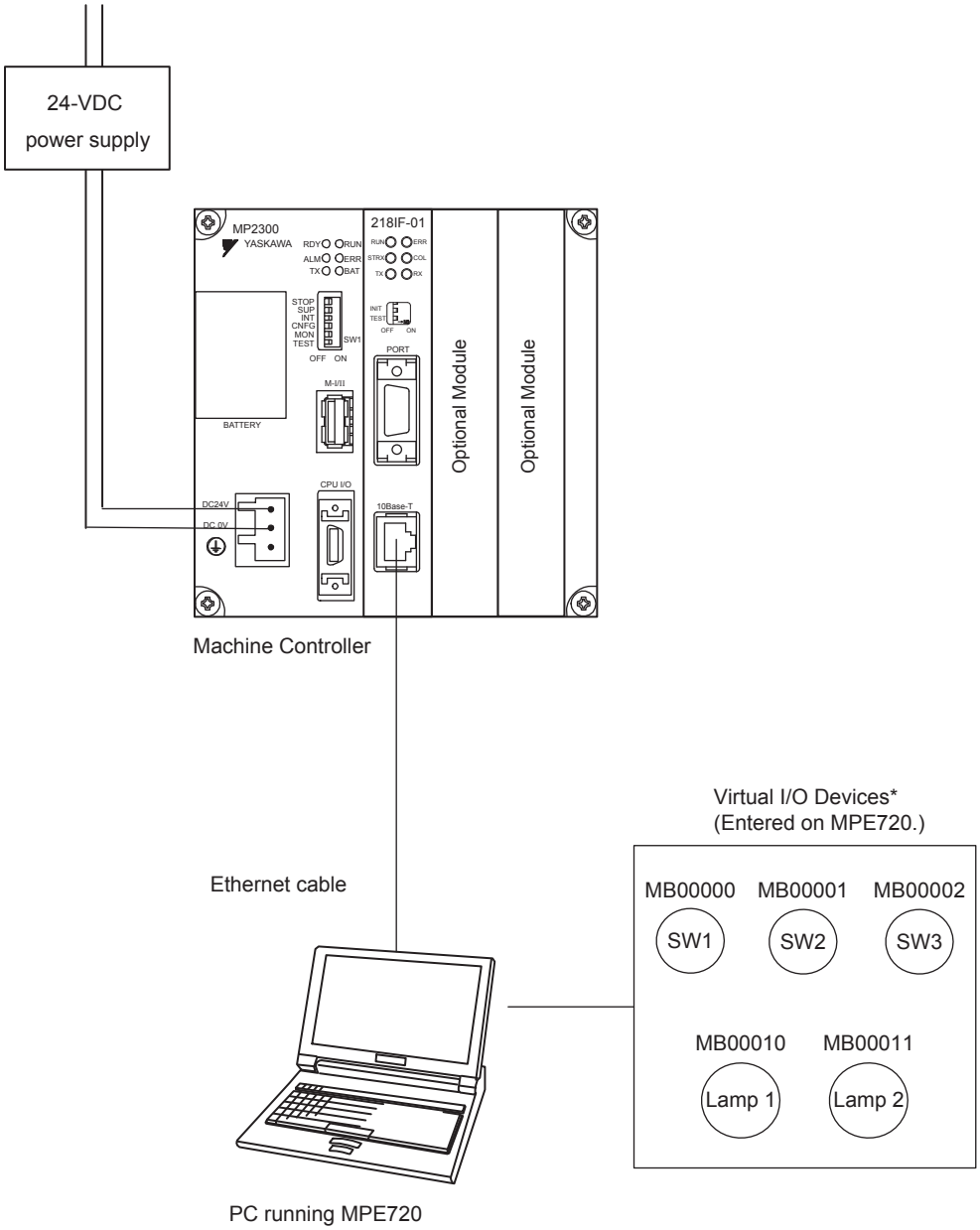
Save the debugged ladder programs to flash memory.

Refer to *3.1.9 Saving the Ladder Programs to Flash Memory*.

- The above flowchart is an example of the ladder program design process. Settings to interface the external devices must be completed to use programs on the actual system.

## 3.1.1 Connecting the Hardware

The flow of ladder program development that is described in this chapter is based on the following system configuration.




* In this chapter, M registers in the Machine Controller are used to simulate virtual I/O devices in the example system.
In practice, the input and output signals would be connected to I/O Modules on the Machine Controller, and the ladder program would be created using I and O registers.

## 3.1.2 Installing MPE720 Version 6

Install MPE720 version 6 on a PC.
Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for the installation procedure.

## 3.1.3 Communications Settings

After you install MPE720 version 6 on the PC, set up communications between the MP2000-series Machine Controller and the PC.
Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for the communications setup procedure.

## 3.1.4 System Startup

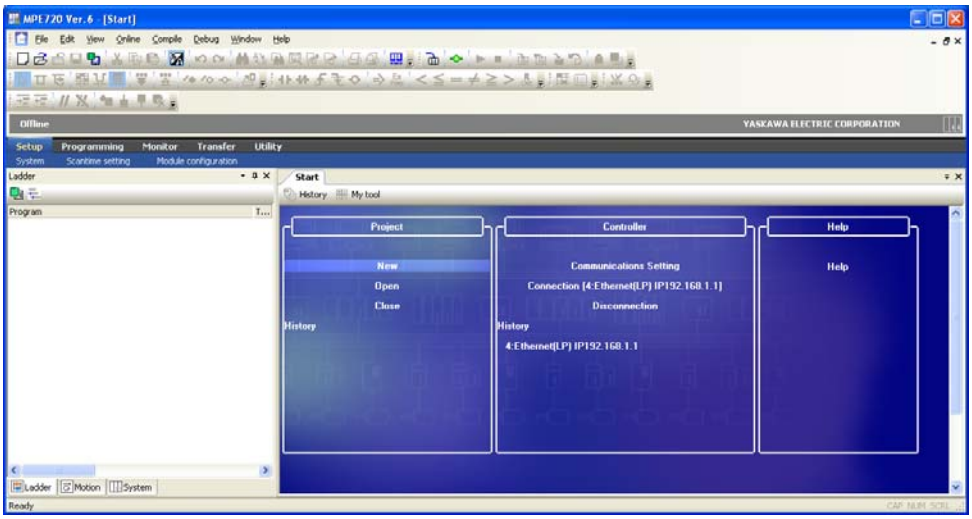
Set up the system by performing self configuration. Self configuration automatically recognizes the Modules that are installed in the MP2000-series Machine Controller and the devices that are connected through the MECHATROLINK connector. This allows you to quickly and easily set up the system. You can perform self configuration by using the DIP switch on the Machine Controller or by using the MPE720.
Refer to the user's manual for your Machine Controller for details on self configuration.

## 3.1.5 Creating a Project

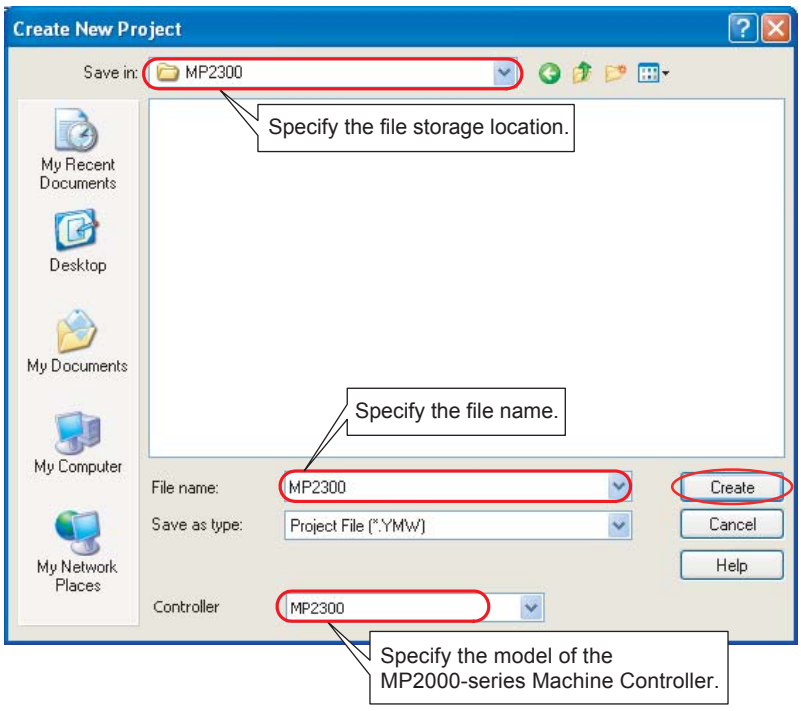
Use the following procedure to create a project.

**1.** Double-click the following icon on the PC desktop to start MPE720 version 6.

**2.** When MPE720 version 6 starts, select ***New*** on the Start Tab Page.



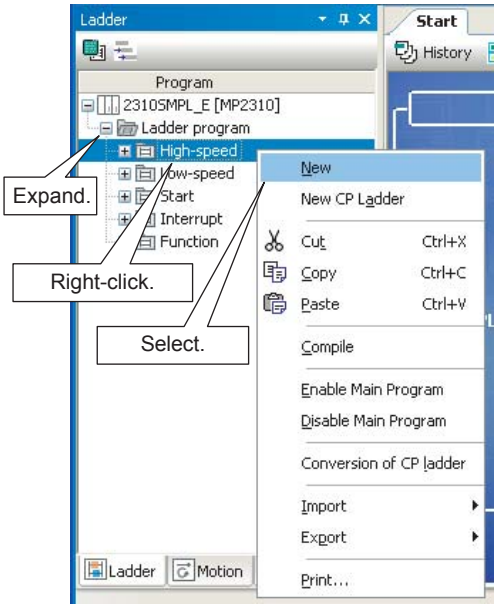

**3.** Specify the file name, file storage location, and Machine Controller model, and then click the **Create** Button.



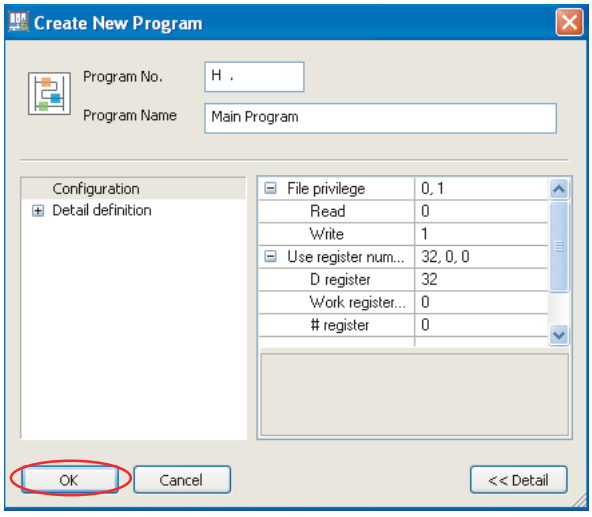
## 3.1.6 Creating Ladder Programs

Start the Ladder Editor and use the following procedure to create a ladder program.

**1.** In the pane on the left, expand the tree under **Ladder program**. Right-click **High-speed** and select ***New*** from the menu.



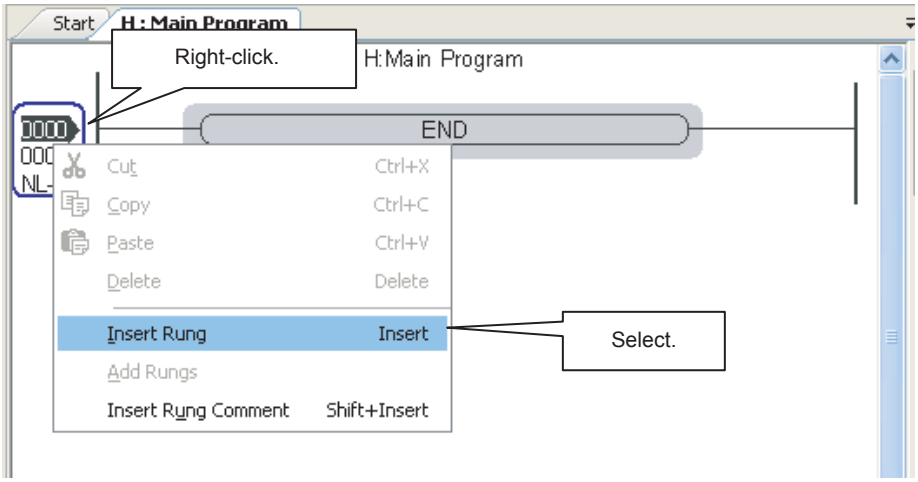

**2.** Click the **OK** Button.



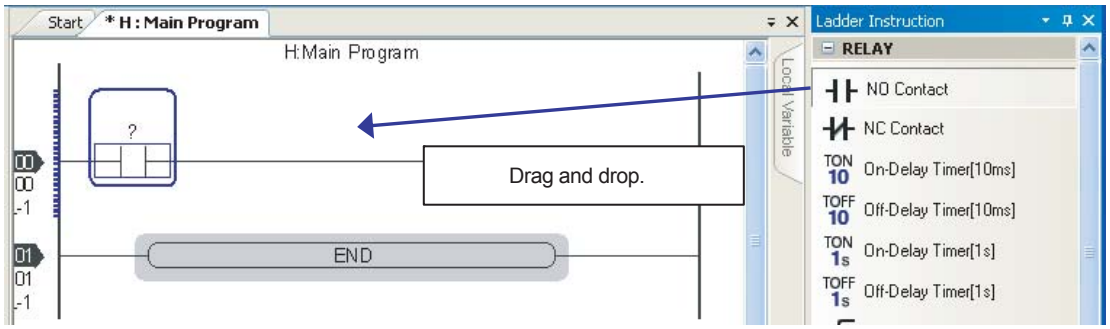
**3.** Create the ladder program in the Ladder Editor that you started.

Ladder programs are entered by inserting rungs, then instructions, and finally parameters for the instructions. The following example shows how to insert an NO Contact instruction.

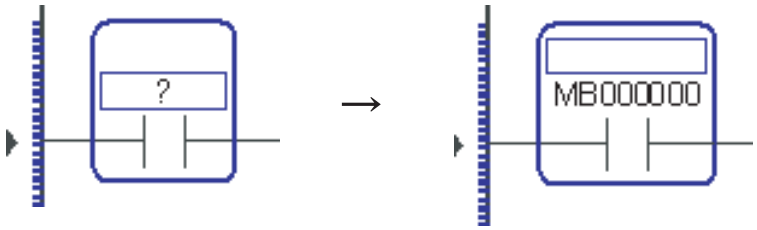
① Right-click the tab with the row number, and select ***Insert Rung***.




② Drag the instruction to insert (here, the NO Contact instruction under the relay instructions) from the Ladder Instructions Pane to the inserted rung.

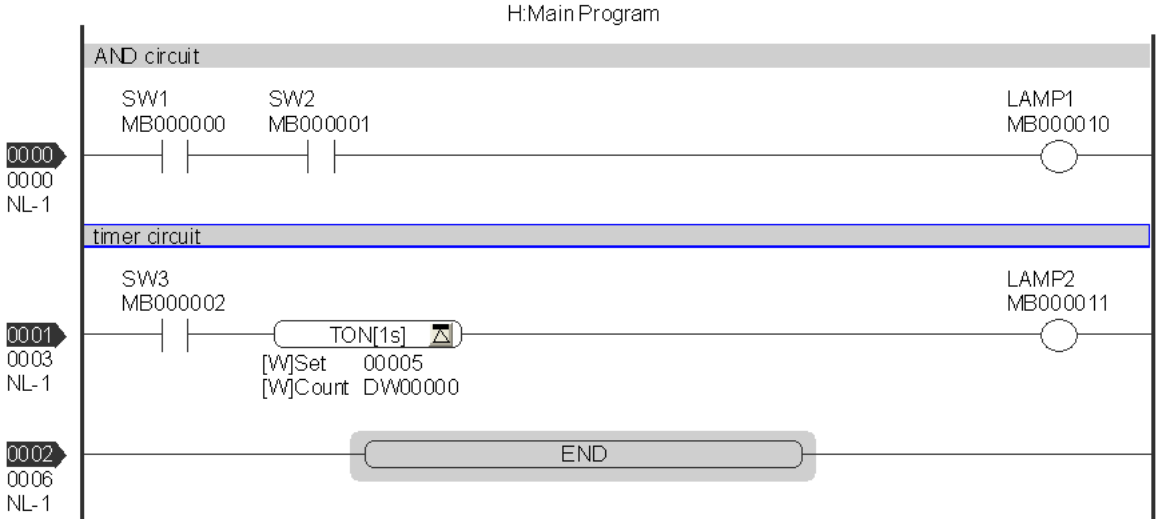



③ Click the portion of the instruction with a question mark and enter the parameter (MB00000) from the keyboard.



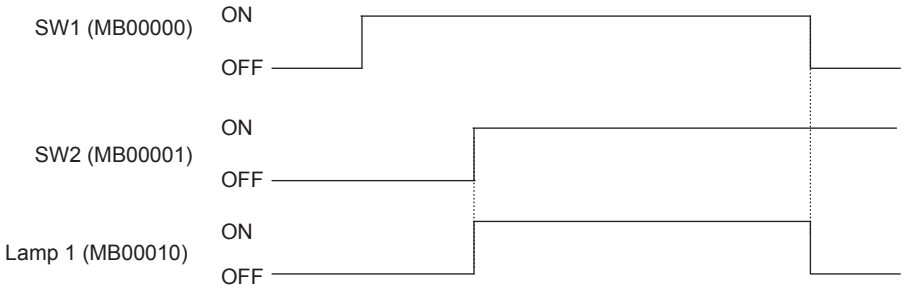

- The types and number of instruction parameters depend on the instruction. Refer to *Chapter 5 Instructions* for details on individual instructions.

④ Repeat steps 1 to 3 until you have created the entire ladder program. The following figures show examples of a ladder program and its timing chart.

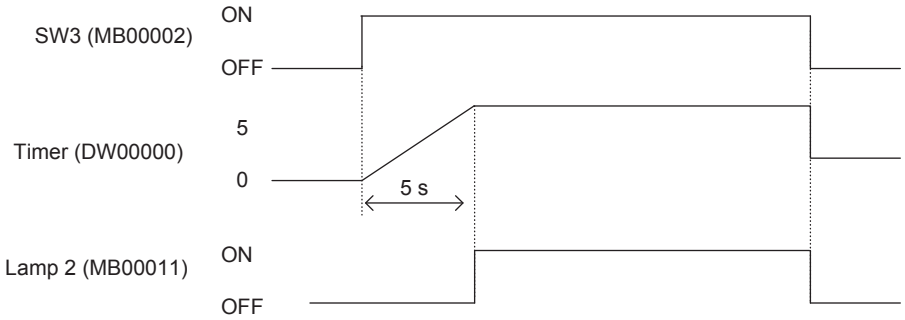



- The ladder program example that is shown above uses M registers for switches and lamps. When you enter a ladder program for an actual system, use the appropriate I and O registers.

AND Circuit Operation




Timer Circuit Operation




**4.** While displaying the ladder program, select ***Compile - Compile*** from the menu bar to compile the program. When the compilation is finished, the ladder program will be saved automatically.

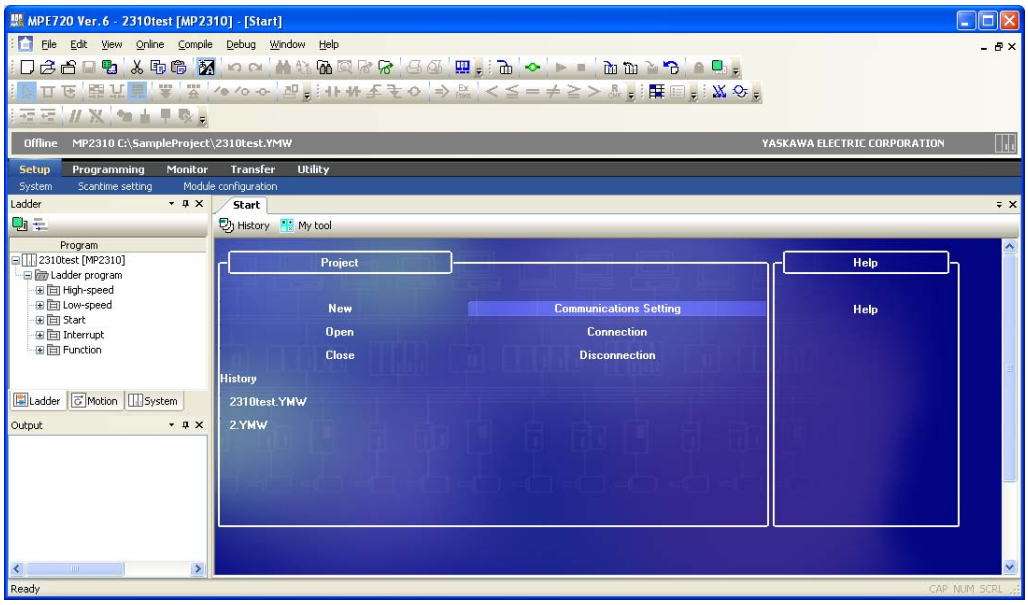
IMPORTANT

If an error is displayed in the Output Pane during compilation, the ladder program will not be saved.

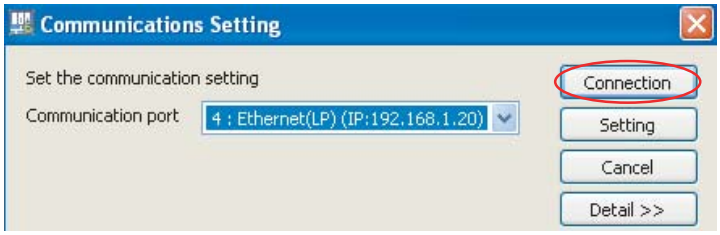
## 3.1.7 Transferring Ladder Programs

Use the following procedure to transfer the ladder program to the MP2000-series Machine Controller. This procedure is not necessary if you created the ladder program online.

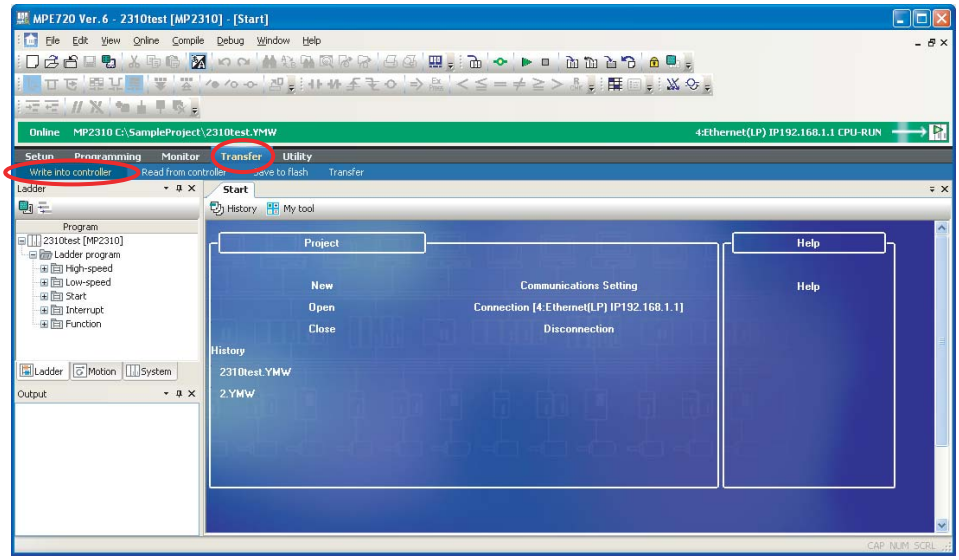
**1.** Select ***Communications Setting*** on the Start Tab Page.



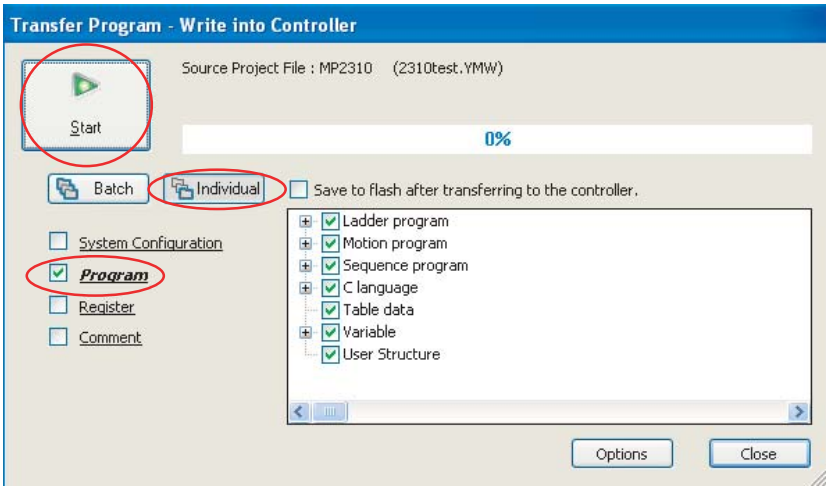

**2.** Select the desired communications port in the **Communications Setting** Box, and then click the **Connection** Button.




**3.** Wait for the MPE720 to go online, and then select ***Transfer*** – ***Write into controller***.



**4.** Click the **Individual** Button, then select the **Program** Check Box. Click the **Start** Button.




INFO

- When an individual transfer is selected, the same file in the Machine Controller will be overwritten with the selected project file data.
- When a batch transfer is selected, the RAM in the MP2000-series Machine Controller will be cleared before the transfer, and all project file data will be written in the RAM.

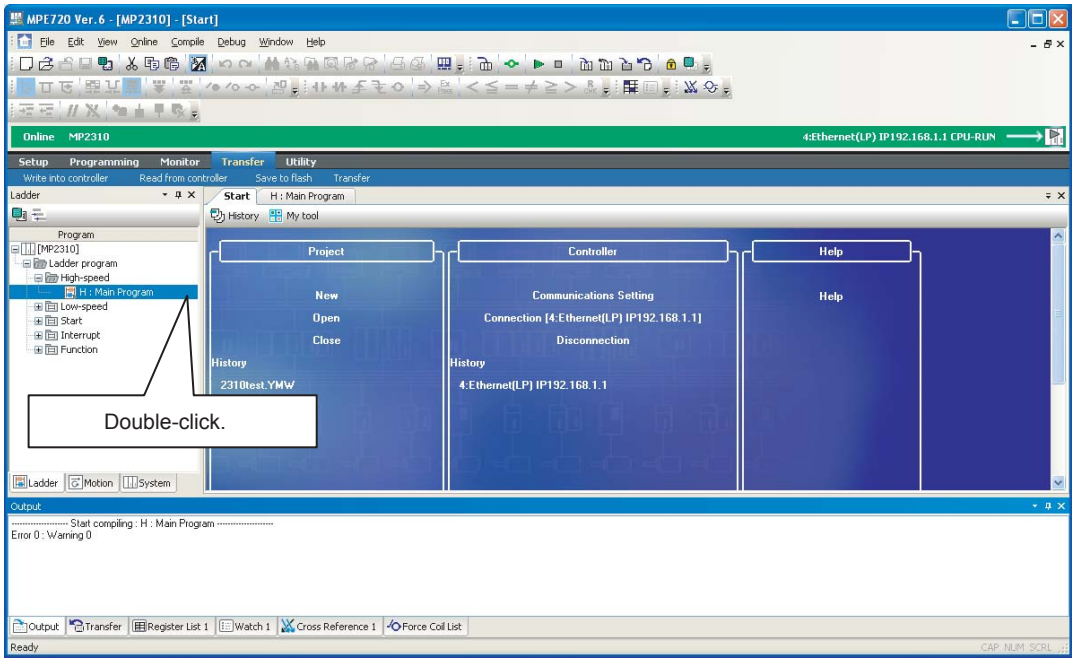
**5.** Click the **CPU STOP** Button. The transfer will start.

## 3.1.8 Checking the Operation of the Ladder Programs

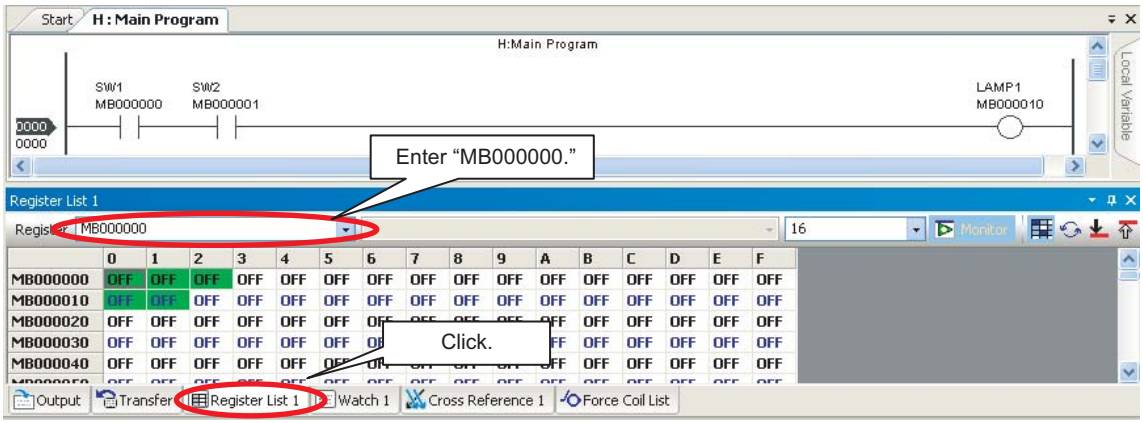
This section provides procedures to check the ladder program that was created in *3.1.6 Creating Ladder Programs*. Confirm that your program operates correctly by manipulating registers with the Register List, and by checking the runtime monitor in the Register List and Ladder Editor.

### ( 1 ) Preparations for Checking Operation

**1.** Open the ladder program that was transferred.



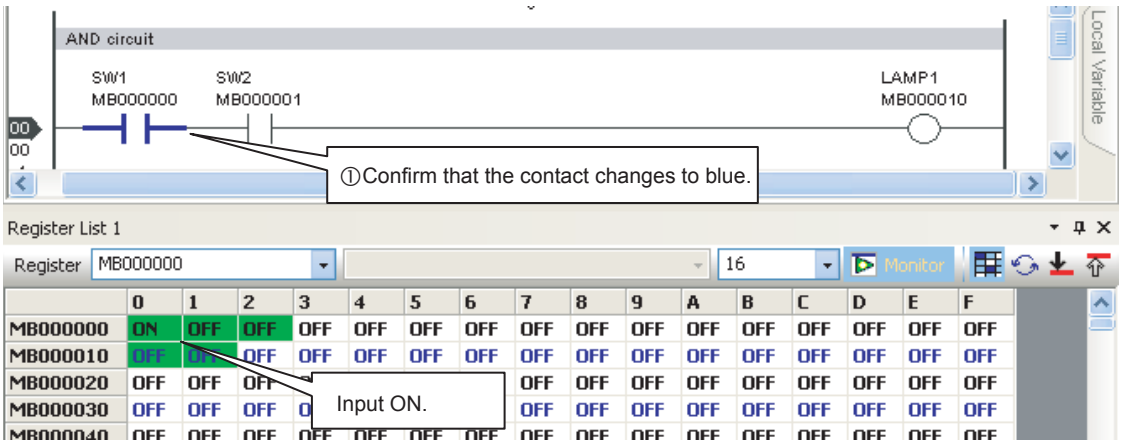

**2.** Click the **Register List 1** Tab, and then enter “MB000000” in the **Register** Box.
If the **Register List 1** Tab is not visible, select ***View*** – ***Register List*** – ***Register List 1*** from the menu bar. The tab will be displayed and the register list will be opened.



## ( 2 ) Confirming the Operation of the 0000th Line (AND Circuit)

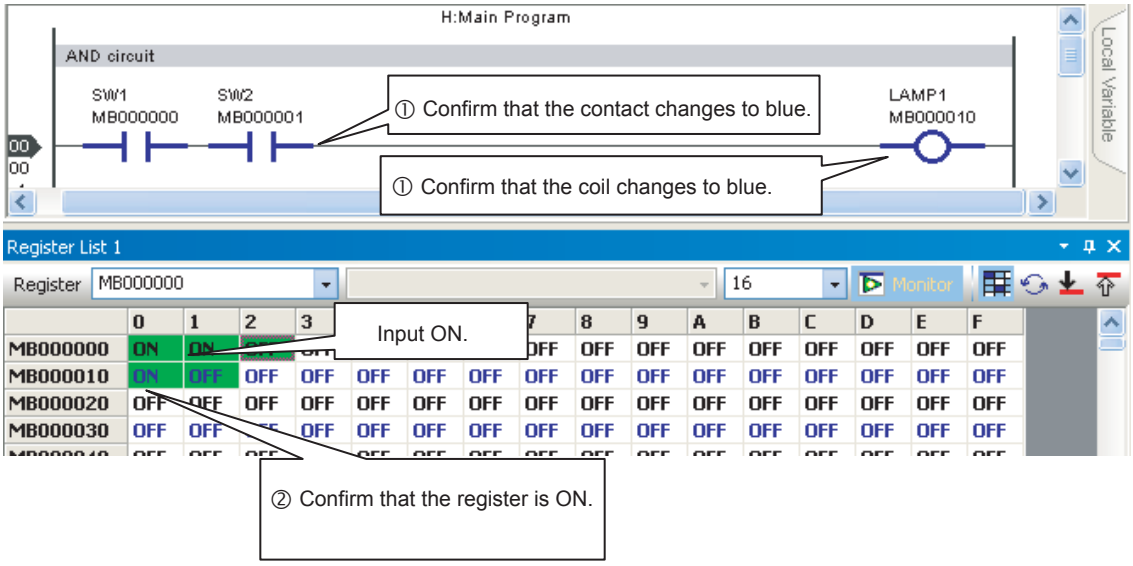
**1.** Set MB000000 to ON in the Register List. Confirm that the NO contact for MB000000 in the Ladder Editor changes to blue.

- When a coil or contact is highlighted in blue, it means that it is ON.




**2.** Set MB000001 to ON in the Register List. Confirm the following points.

- In the Ladder Editor, the NO contact for MB000001 and coil for MB000010 must be blue.
- In the Register List, MB000010 must be ON.

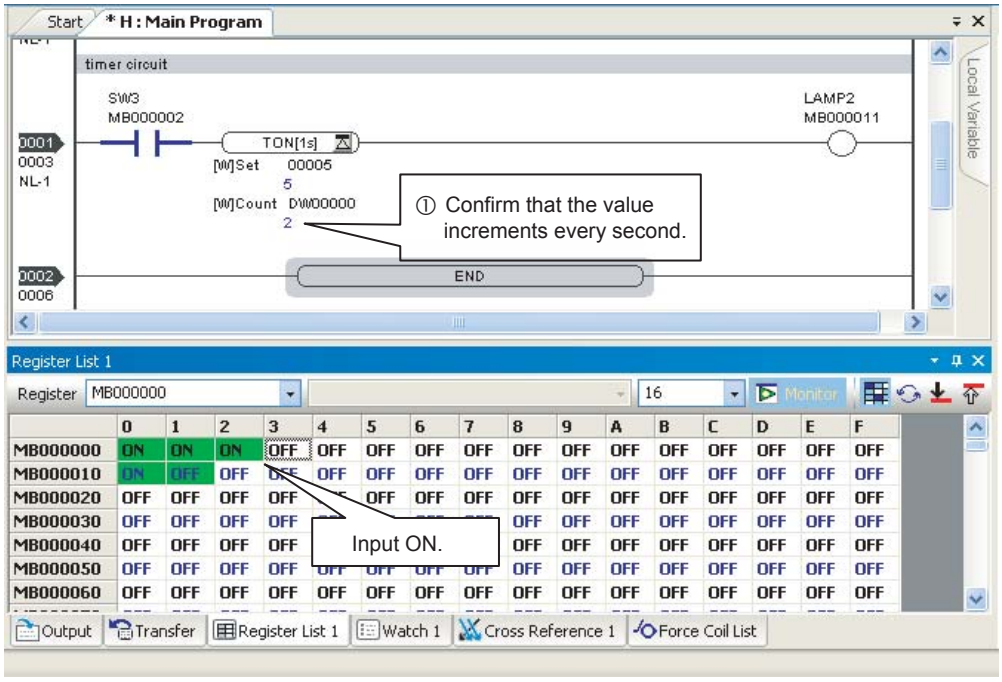



If no problems occur in the above procedure, then this concludes checking the operation of the 0000th line.

## ( 3 ) Confirming the Operation of the 0001st Line (Timer Circuit)

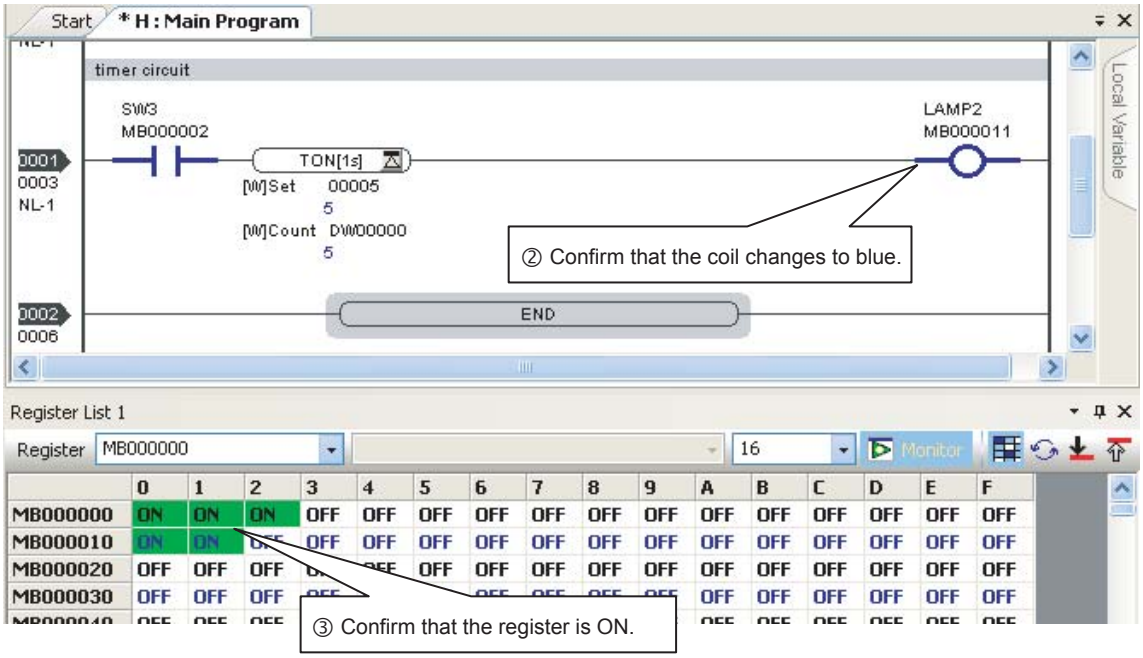
Set MB000002 to ON in the Register List. Confirm the following points.

① The DW00000 timer must increment every second.




② After five seconds, the coil for MB000011 must turn blue in the Ladder Editor.

③ In the Register List, MB000011 must be ON for step ②.

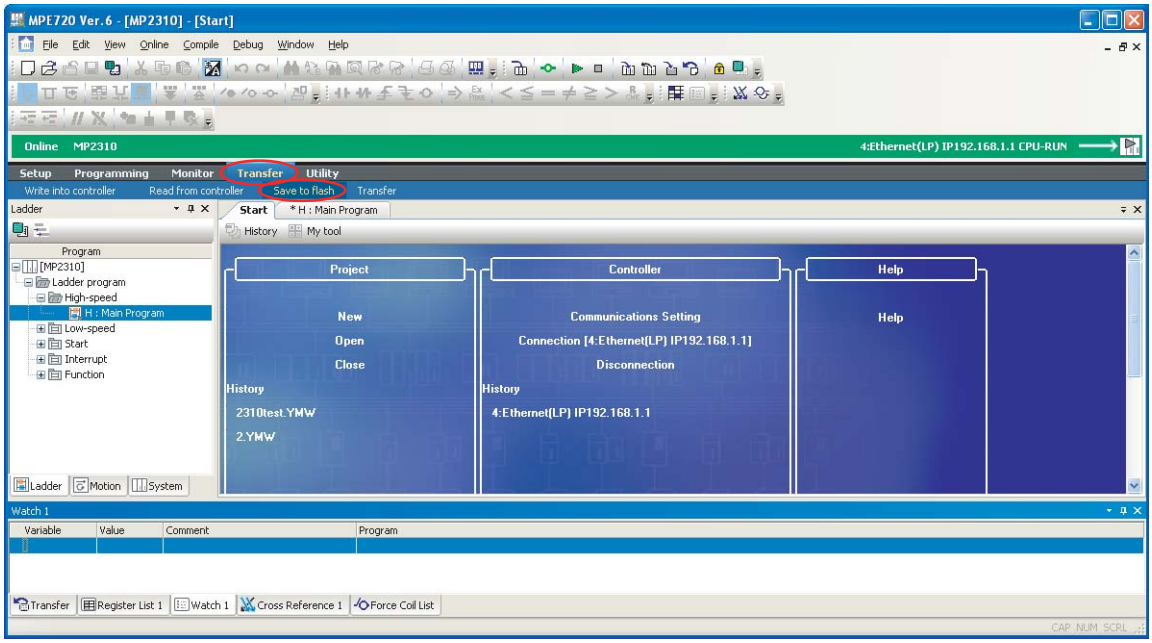



If no problems occur in the above procedure, then this concludes checking the operation of the 0001st line.

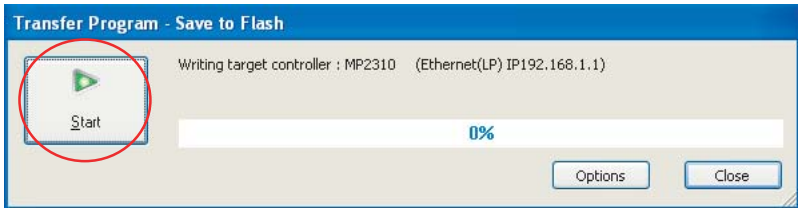
## 3.1.9 Saving the Ladder Programs to Flash Memory

Use the following procedure to save the data from the RAM in the MP2000-series Machine Controller to the flash memory in the MP2000-series Machine Controller.

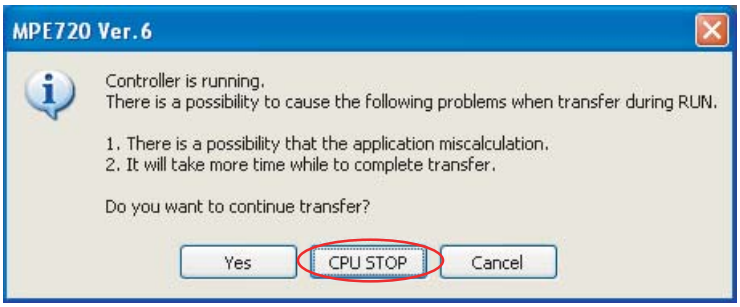
**1.** Select ***Transfer*** – ***Save to flash*** from the following window.




**2.** Click the **Start** Button.




**3.** Click the **CPU STOP** Button. The transfer will start.




**4.** Click the **Yes** Button in the following dialog box. The Machine Controller will switch to RUN Mode.

# 4

# Programming

This chapter describes ladder programming methods and the elements that are necessary for ladder programming.


4.1 Ladder Program Editor ----- 4-2

4.2 Ladder Drawings ----- 4-3

4.2.1 Types of Ladder Drawings ----- 4-3

4.2.2 Controlling the Execution of Drawings ----- 4-5

4.3 User Functions ----- 4-7

4.3.1 What Is a User Function? ----- 4-7

4.3.2 Creating User Functions ----- 4-9

4.3.3 Calling a User Function ----- 4-12

4.4 Registers (Variables) ----- 4-13

4.4.1 What Are Registers? ----- 4-13

4.4.2 Register Types ----- 4-14

4.4.3 Data Types ----- 4-17

4.4.4 Index Registers (i, j) ----- 4-19

4.5 Table Data ----- 4-21

4.5.1 What Is Table Data? ----- 4-21

4.5.2 Creating Table Data ----- 4-21

4.6 Transferring Data ----- 4-23

4.7 Setting the High-speed/Low-speed Scan Times ----- 4-24

4.8 Advanced Programming ----- 4-25

4.8.1 Motion Programs ----- 4-25

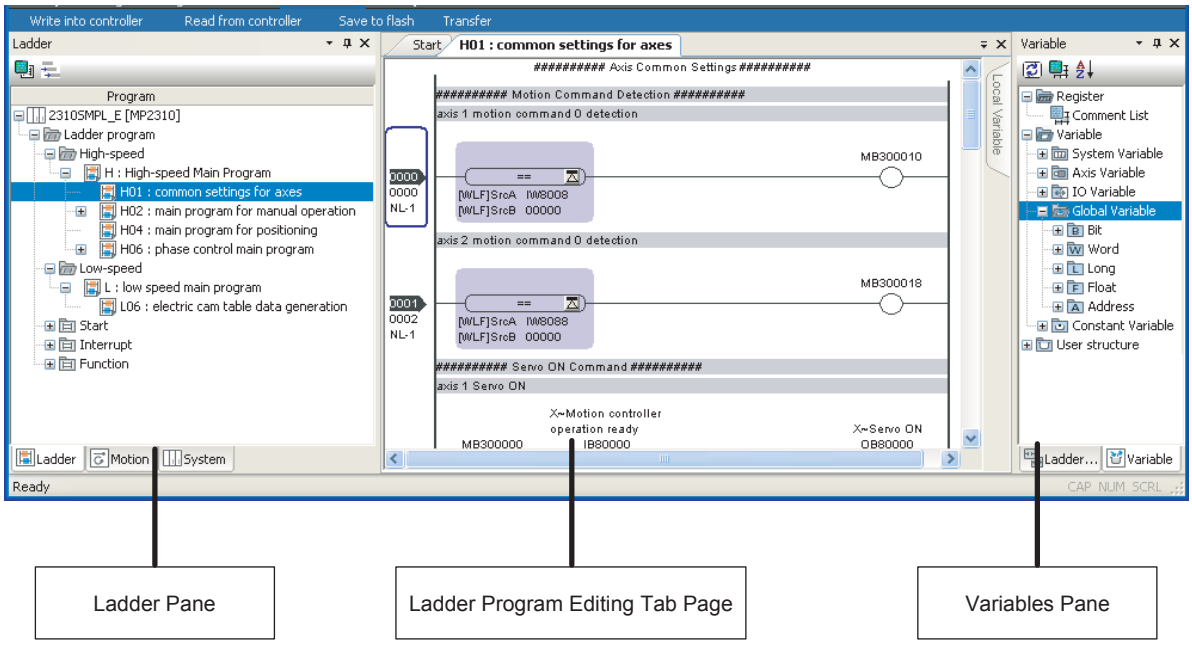
4.8.2 C-language Programs ----- 4-26

4.8.3 Security ----- 4-27

4.8.4 Tracing ----- 4-28

# 4.1 Ladder Program Editor

On the MPE720 version 6 Engineering Tool, the following panes are displayed to edit a ladder program. These panes are used to create and edit ladder programs.




## ■ Ladder Pane

Ladder programs are displayed by drawing.
Refer to *4.2 Ladder Drawings* for details on drawings.

## ■ Ladder Program Editing Tab Page

This tab page is used to edit ladder programs.

## ■ Variables Pane

This pane displays variables. Refer to *4.4 Registers (Variables)* for details on variables.
In addition to the panes and tab page that were just described, various other panes, tab pages, and tool bars also exist.
Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for details on MPE720 version 6.

# 4.2 Ladder Drawings

Ladder programs are managed as drawings (ladder drawings) that are identified by their drawing numbers (DWG numbers).
The ladder drawings form the basis of the ladder programs.

## 4.2.1 Types of Ladder Drawings

### ( 1 ) Types and Priorities of Drawings

There are the following types of ladder drawings: parent drawings, child drawings, grandchild drawings, and operation error drawings.

- Parent Drawings
  These drawings are automatically executed by the system when the execution conditions that are listed in the following table are met.
- Child Drawings
  These drawings are executed when they are called from a parent drawing with a Call Program (SEE) instruction.
- Grandchild Drawings
  These drawings are executed when they are called from a child drawing with a Call Program (SEE) instruction.
- Operation Error Drawings
  These drawings are automatically executed by the system when an operation error occurs.

There are also five different types of drawings based on their role.
The following table gives the priority and parent drawing execution conditions for each type of drawing.

| Priority* | Drawing Type | Role | Parent Drawing Execution Condition | Maximum Number of Drawings |
|---|---|---|---|---|
| 1 | DWG.A | Startup processing | Power ON<br>(Processed once when the power supply is turned ON.) | 64 |
| 2 | DWG.I | Interrupt processing | External interrupt<br>(Executed when a DI interrupt or counter match interrupt is received from an Optional Module.) | 64 |
| 3 | DWG.H | High-speed scan processing | Started at fixed intervals.<br>(Executed every high-speed scan.) | 200 |
| 4 | DWG.L | Low-speed scan processing | Started at fixed intervals.<br>(Executed every low-speed scan.) | 500 |
| – | Functions | User functions | Function call<br>(Executed when called with a FUNC instruction from a drawing.) | 500 |

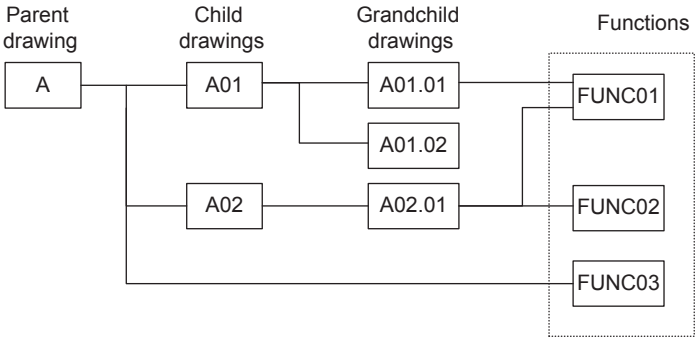
* Drawings with lower numbers have higher priority.

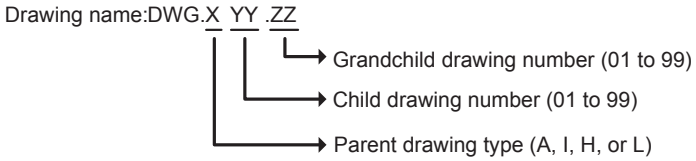
The breakdown of the number of drawings in each category is given in the following table.

| Drawing | Number of Drawings | | | |
|---|---|---|---|---|
| | DWG.A | DWG.I | DWG.H | DWG.L |
| Parent drawings | 1 drawing | 1 drawing | 1 drawing | 1 drawing |
| Operation error drawings | 1 drawing | 1 drawing | 1 drawing | 1 drawing |
| Child drawings | Total of 62 drawings max. | Total of 62 drawings max. | Total of 198 drawings max. | Total of 498 drawings max. |
| Grandchild drawings | | | | |

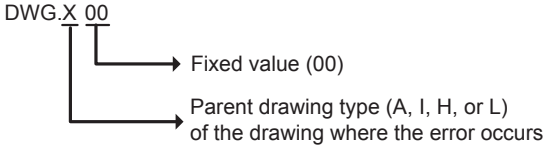
## ( 2 ) Hierarchical Configuration of Drawings

Each process program is organized in a parent-child-grandchild hierarchy.
The parent drawing first must call a child drawing, and then the child drawing must call a grandchild drawing. This is called the hierarchical configuration of drawings.
A parent drawing cannot call a child drawing with a different drawing type. Similarly, a child drawing cannot call a grandchild drawing from a different drawing type. A parent drawing cannot call a grandchild drawing directly.
You can call functions from any drawing regardless of the drawing type or hierarchy.
The hierarchy of drawings is shown below using DWG.A drawings as an example.



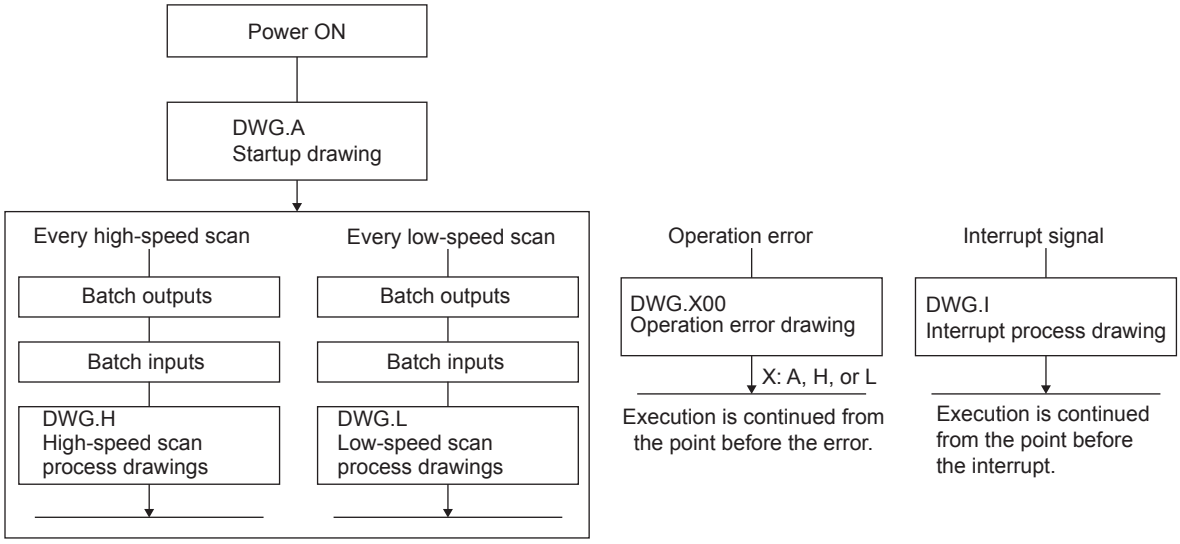

The following notation is used for operation error drawings.



## 4.2.2 Controlling the Execution of Drawings

### ( 1 ) Controlling the Execution of Drawings

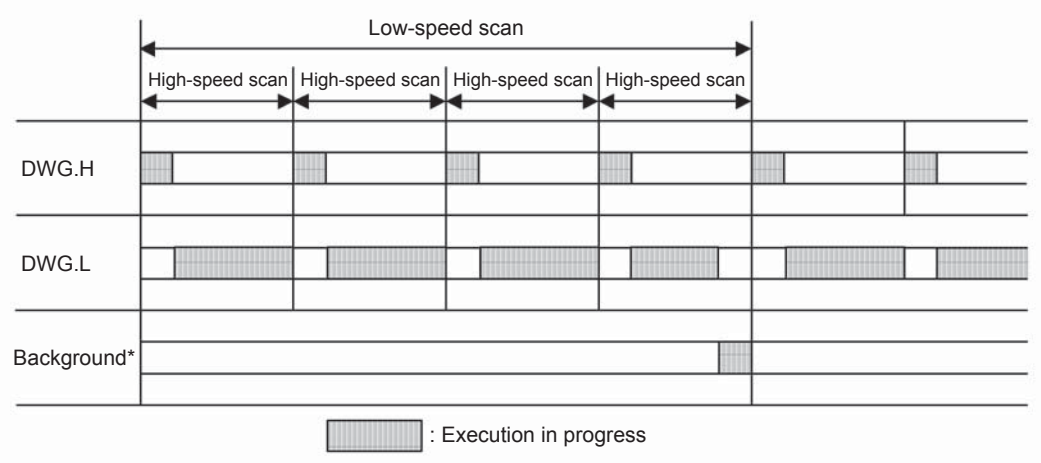
Drawings are executed based on their priorities, as shown in the following figure.




- The parent drawing of each drawing is automatically called and executed by the system.

### ( 2 ) Scheduling the Execution of Scan Process Drawings

All scan process drawings are not executed at the same time. The following figure shows how execution time is allocated to them based on their priority levels.

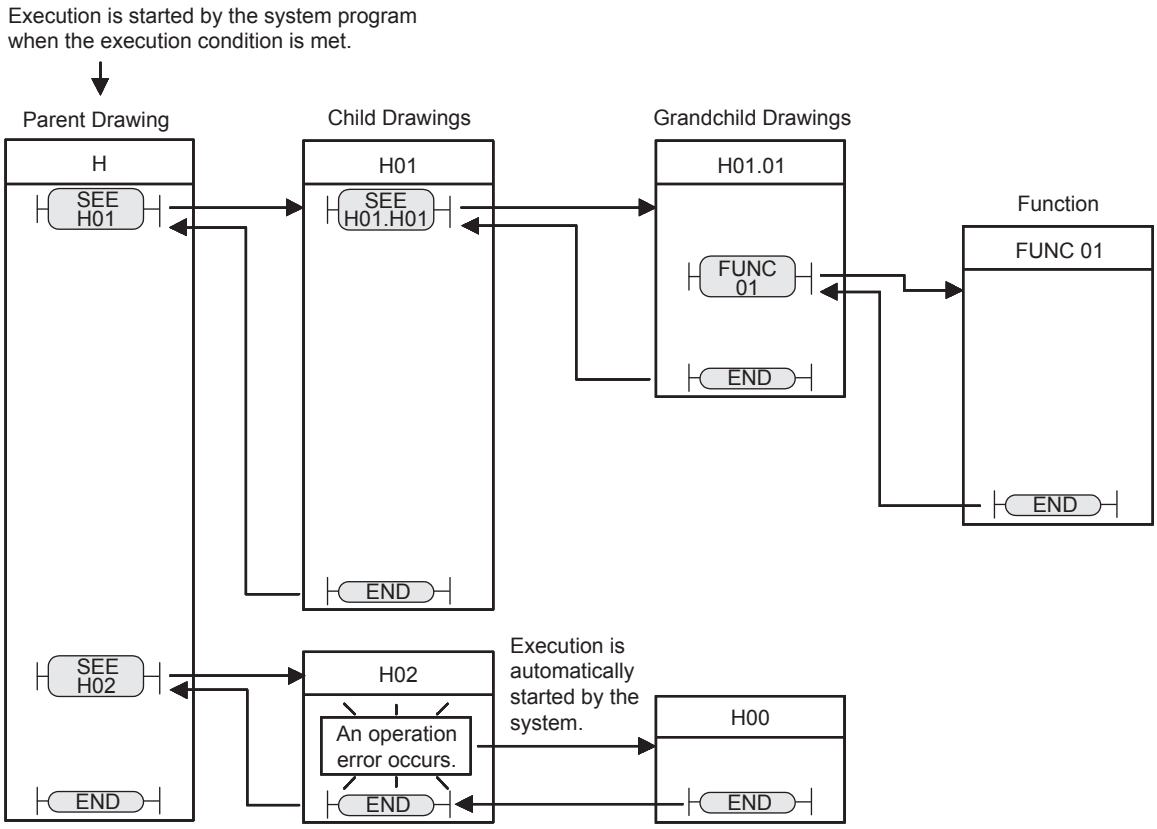



* This time is used to execute internal system processing, such as self-diagnosis.

The low-speed scan is executed during the time that is not used by the high-speed scans. Set the time of the high-speed scan to approximately twice the total execution time of the high-speed drawings (DWG.H).

## ( 3 ) Execution Processing of Drawings

The execution processing for drawings is executed by calling the drawings from the top to the bottom, following the hierarchy of the drawings. The hierarchy of drawings is shown below using DWG.A drawings as an example.



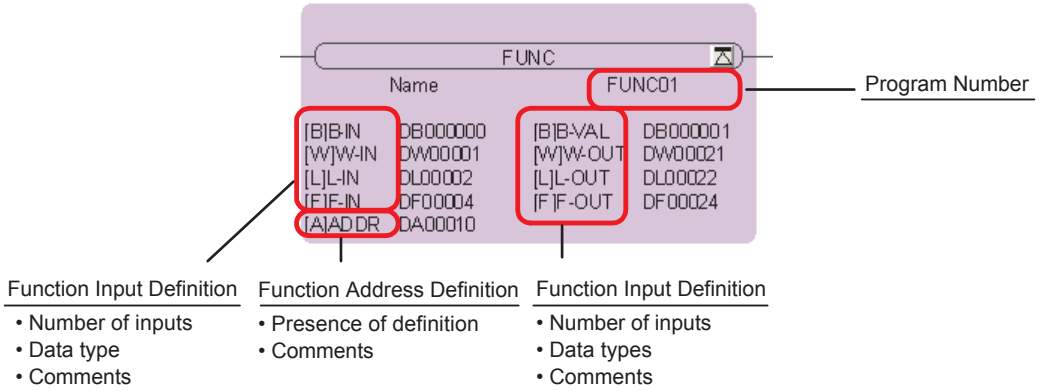

- The parent drawing is automatically called and executed by the system.
  Child drawings and grandchild drawings are executed by calling them from a parent drawing or a child drawing using the Call Program (SEE) instruction.
- You can call functions from any drawing. You can also call functions from other functions.
- If an operation error occurs, the operation error drawing for the drawing type will be started automatically.

# 4.3 User Functions

## 4.3.1 What Is a User Function?

### ( 1 ) Overview of User Functions

A user function contains a function definition (program number and I/O definitions) and processing instructions that are defined by the user.
The following figure shows an example of a function definition.




The processing to be performed by a function is created using a ladder program.
Functions are executed when they are called from a parent, child, or grandchild drawing with the FUNC instruction.
You can call a user function freely from any drawing. You can also simultaneously call the same function from different types or different levels of drawings. You can also call user functions from other user functions.
The use of functions provides the following advantages.

- Easy user program modularization
- Easy user programming and program maintenance

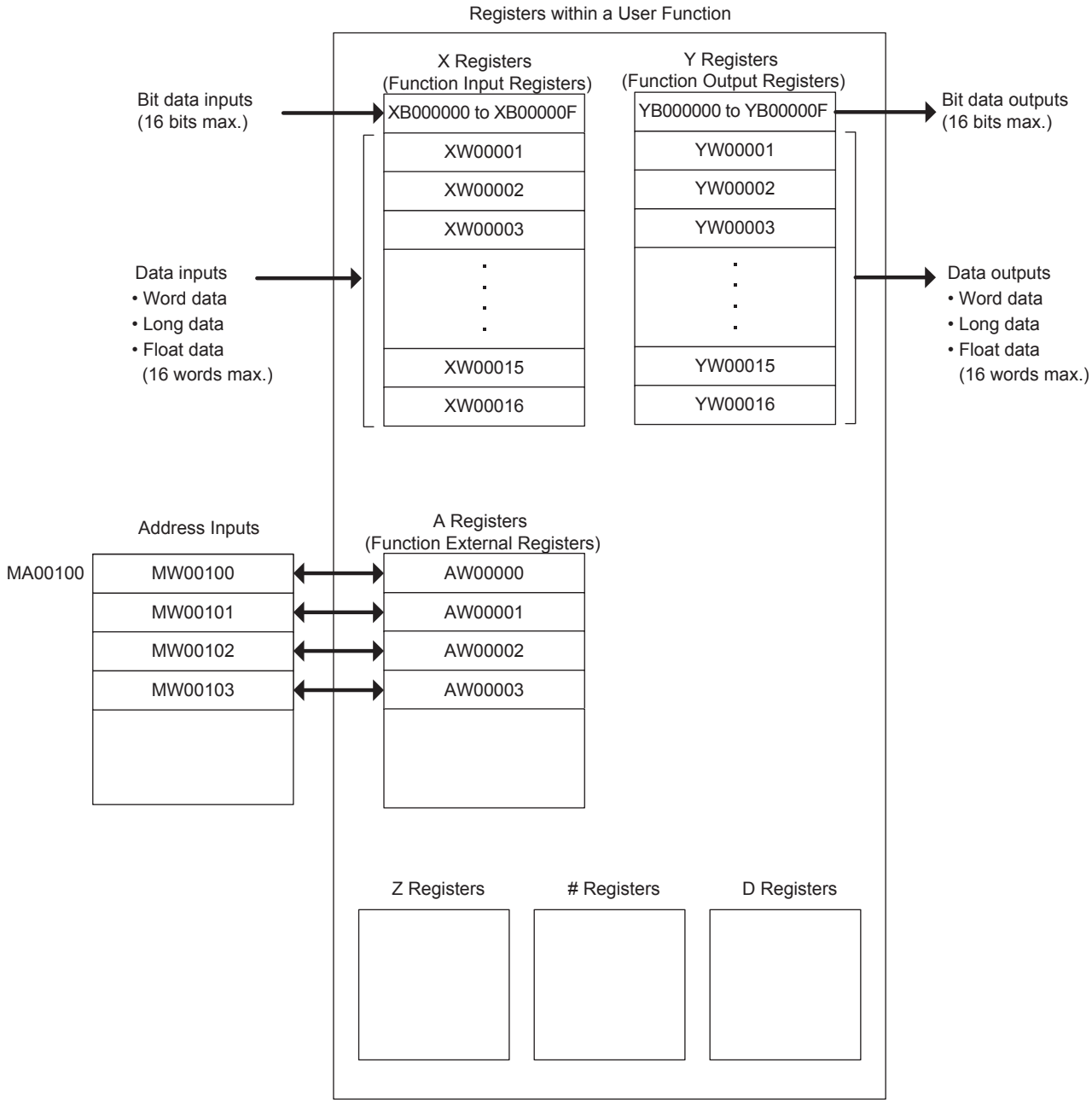
IMPORTANT

User functions can be called from any programs, any number of times.

When you call a user function, consider what values could be in the variables in each function, and perform initialization as needed.

Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

## ( 2 ) Relationship between I/O Data for a Function and Registers in the Function




IMPORTANT

The X, Y, Z, and D registers are initialized to different values when a function is called.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

INFO

The S, M, I, O, and C registers can also be accessed from within a function.

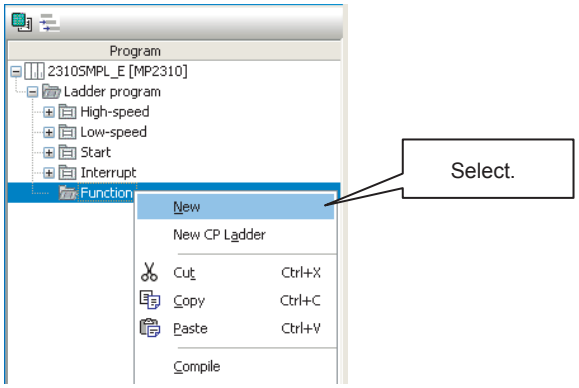
## 4.3.2 Creating User Functions

This section describes how to create a user function that has, as an example, the following specifications.

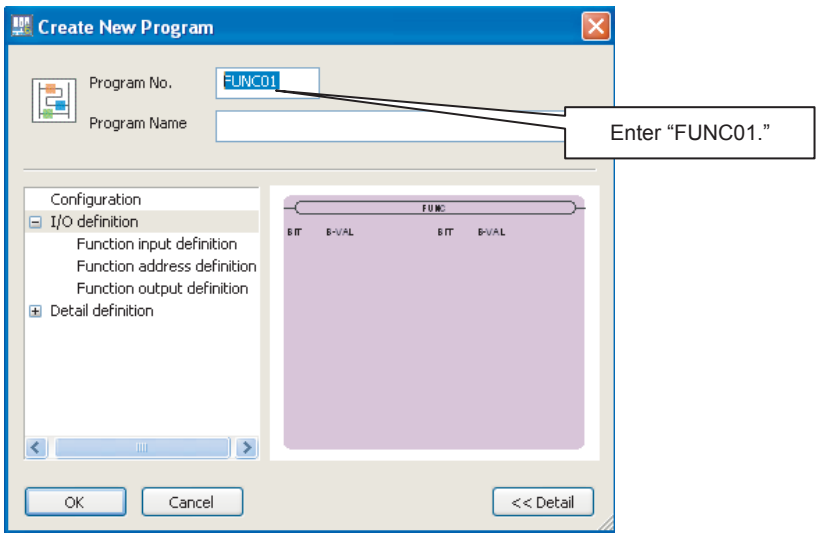
| Function Definition Item | Name | Remarks |
|---|---|---|
| Program Number | FUNC01 | |
| Function Input Value | IN | Integer data |
| Function Output Value 1 | OUT1 | Integer data |
| Function Output Value 2 | OUT2 | Integer data |
| Processing Details | | |
| Multiply the function input value (IN) by 2 and output it to function output value 1 (OUT1).<br>Multiply the function input value (IN) by 3 and output it to function output value 2 (OUT2). | | |

### ■ Procedure to Create a User Function

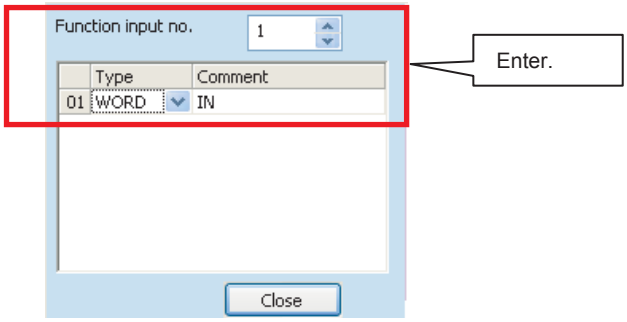
**1.** In the pane on the left, expand the tree under **Ladder program**. Right-click **Function** and select ***New*** from the menu.



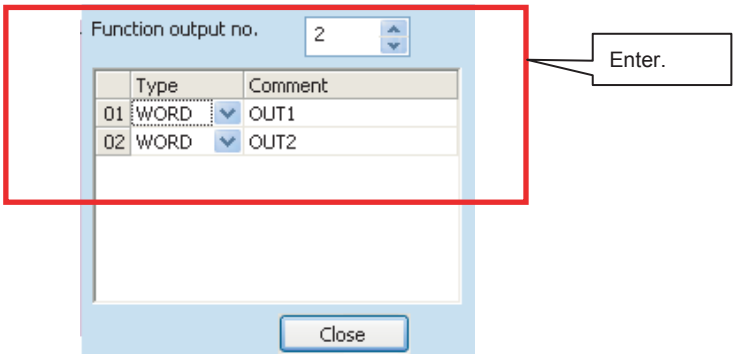

**2.** Enter “FUNC01” in the **Program No.** Box.



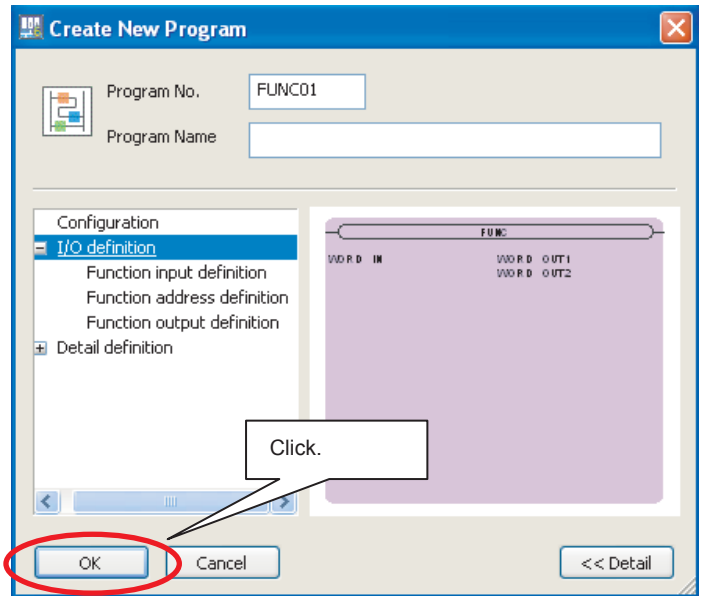
**3.** Select **Function input definition** under **I/O definition** and enter the following information.



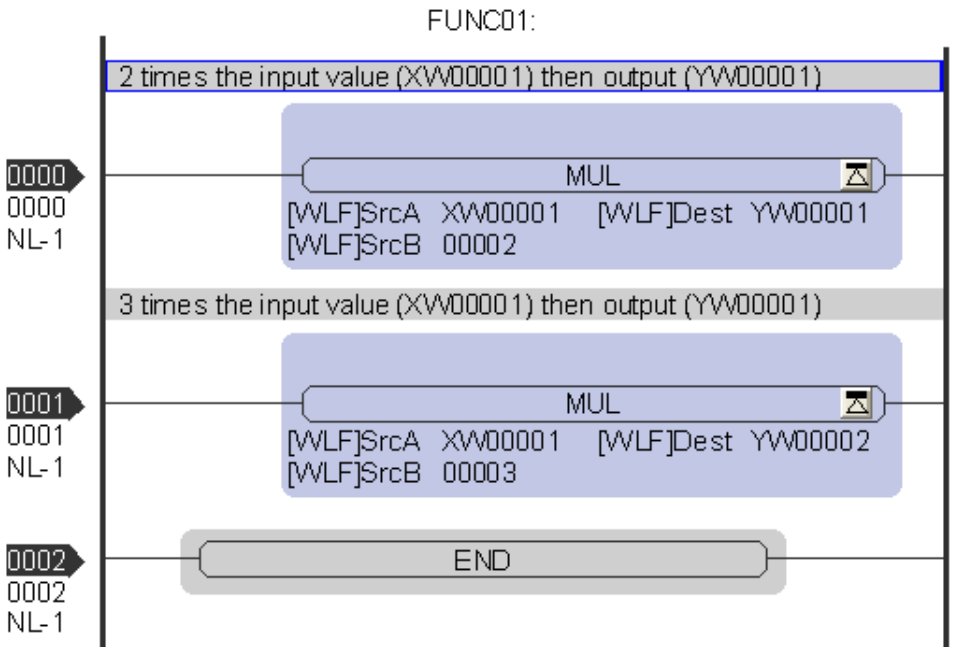

**4.** Select **Function output definition** under **I/O definition** and enter the following information.




**5.** Click the **OK** Button. This concludes setting the function definition.



**6.** Create the following ladder program in the drawing of the FUNC01 user function that was created in step 5.



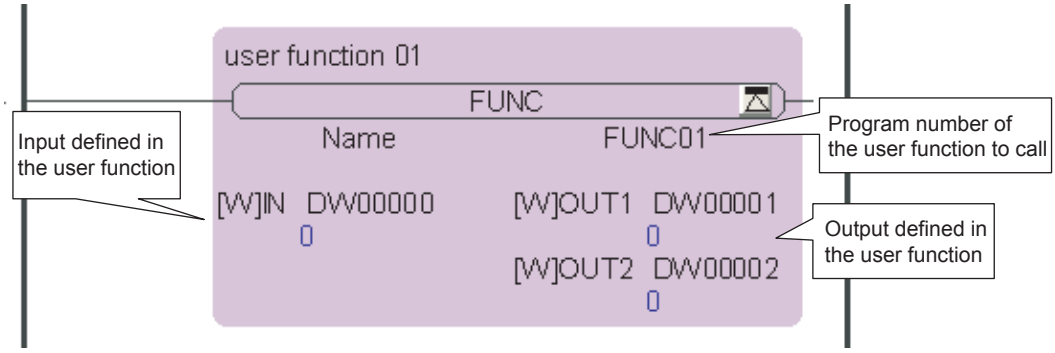

**7.** Compile the user function to conclude the creation of the user function.

## 4.3.3 Calling a User Function

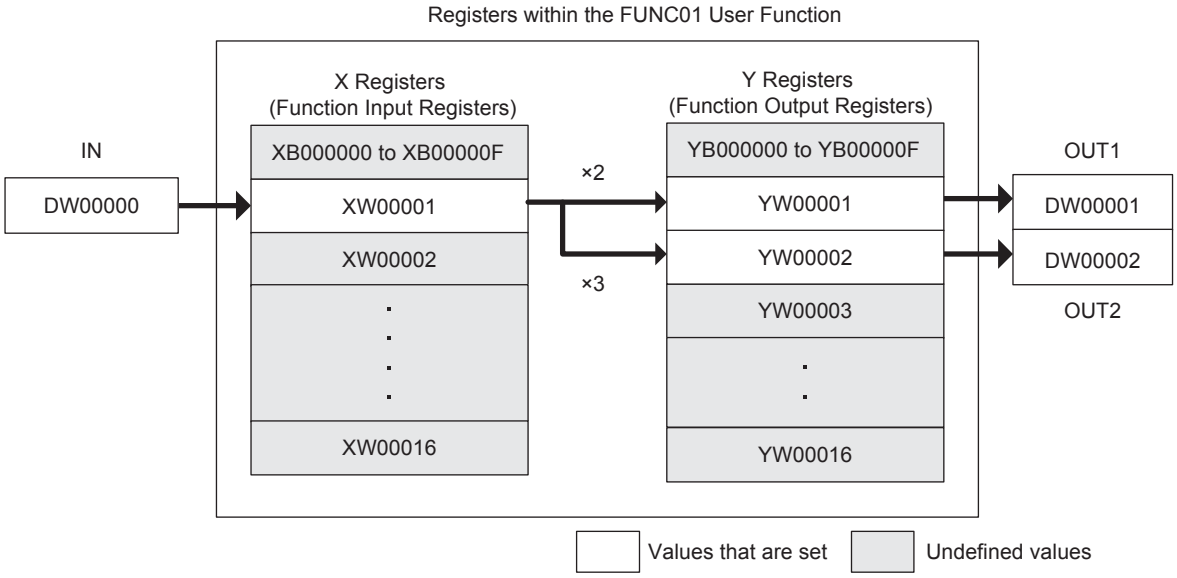
You can call a user function by using a FUNC instruction in the ladder drawing.
This section describes how to call the user function that was created in the previous section from the high-speed drawing (DWG.H).

### ■ Example for Calling the FUNC01 User Function from DWG.H

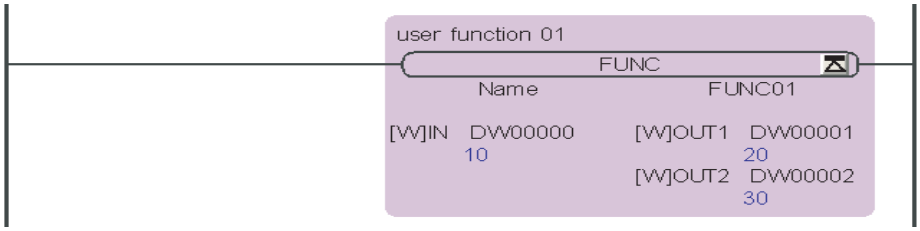
Program a FUNC instruction in DWG.H as shown below.




This diagram shows a conceptual image of what the programming shown above accomplishes.




In this example, when DW00000 in DWG.H is set to 10, DW00001 becomes 20 and DW00002 becomes 30, demonstrating that the user function was called correctly.



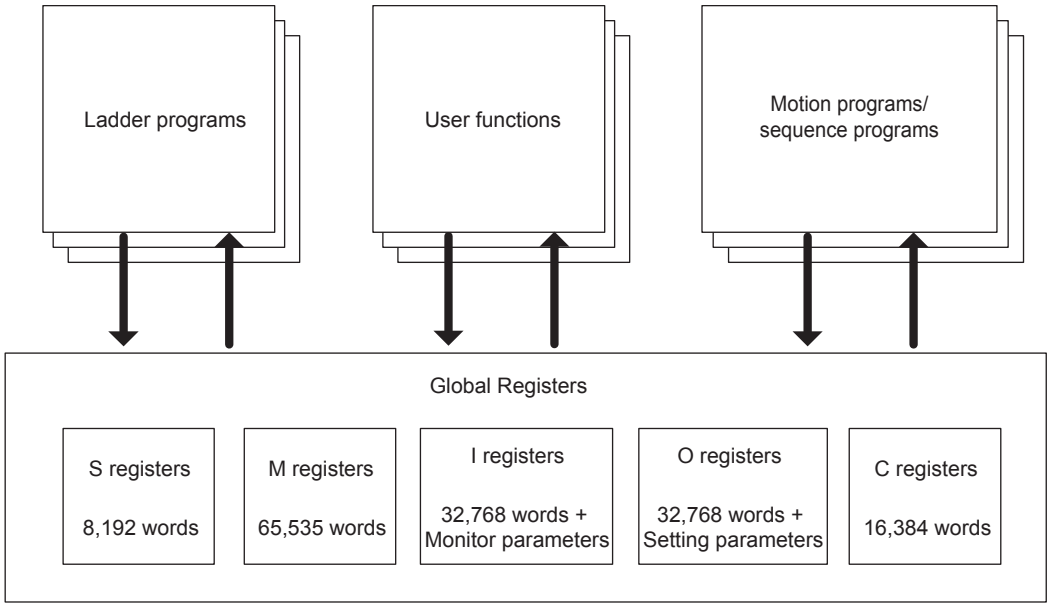
# 4.4 Registers (Variables)

## 4.4.1 What Are Registers?

Registers are areas that store data within the Machine Controller. Variables are registers with labels (variable names). There are two kinds of registers: global registers that are shared between all programs, and local registers that are used only by a specific program.

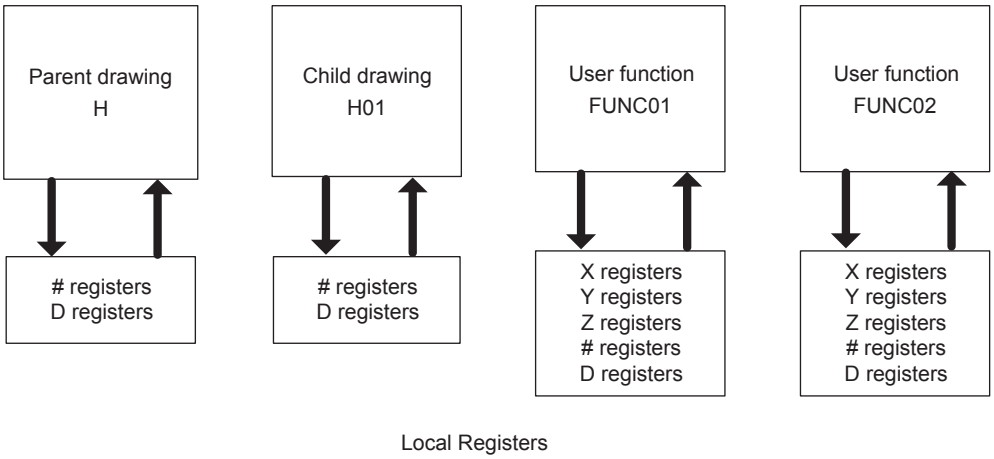
### ( 1 ) Global Registers

Global registers are variables that are shared by ladder programs, user functions, motion programs, and sequence programs. Memory space for global registers is reserved by the system for each register type.




### ( 2 ) Local Registers

Local registers can be used within a specific drawing. They cannot be used in other drawings.



## 4.4.2 Register Types

### ( 1 ) Global Registers

Global registers are variables that are shared by ladder programs, user functions, motion programs, and sequence programs. In other words, the operation results of a ladder program can be used by other user functions, motion programs, or sequence programs.

| Type | Name | Designation Method | Usable Range | Description |
|---|---|---|---|---|
| S | System registers (S registers) | SBnnnnnh, SWnnnnn, SLnnnnn, SFnnnnn, SAnnnnn | SW00000 to SW08191 | These registers are prepared by the system. They report the status of the Machine Controller and other information. The system clears the registers from SW00000 to SW00049 to 0 at startup. They have a battery backup. |
| M | Data registers (M registers) | MBnnnnnh, MWnnnnn, MLnnnnn, MFnnnnn, MAnnnnn | MW00000 to MW65534 | These registers are used as interfaces between programs. They have a battery backup. |
| I | Input registers (I registers) | IBhhhhh, IWhhhh, ILhhhh, IFhhhh | IW0000 to IW7FFF | These registers are used for input data. |
| | | | IW8000 to IWFFFF | These registers store the motion monitor parameters. These registers are used for Motion Modules. |
| O | Output registers (O registers) | OBhhhhh, OWhhhh, OLhhhh, OFhhhh | OW0000 to OW0FFF | These registers are used for output data. |
| | | | OW8000 to OWFFFF | These registers store the motion setting parameters. These registers are used for Motion Modules. |
| C | Constant registers (C registers) | CBnnnnnh, CWnnnnn, CLnnnnn, CFnnnnn, CAnnnnn | CW00000 to CW16383 | These registers can be read in programs but they cannot be written. The values are set from the MPE720. |

- n: decimal digit, h: hexadecimal digit

### ( 2 ) Local Registers

Local registers are valid within only one specific program. The local registers in other programs cannot be accessed. You specify the usable range of local registers from the MPE720.

| Type | Name | Designation Method | Description |
|---|---|---|---|
| # | # registers | #Bnnnnnh, #Wnnnn, #Lnnnnn, #Fnnnnn, #Annnnn | These registers can be read in programs but they cannot be written. The values are set from the MPE720. |
| D | D registers | DBnnnnnh, DWnnnn, DLnnnnn, DFnnnnn, DAnnnnn | These registers can be used for general purposes within a program. By default, 32 words are reserved for each program. The default values after startup depend on the setting of the **D Register Clear when Start** Option. For details, refer to ■ *Setting the D Register Clear When Start Option.* |

- n: decimal digit, h: hexadecimal digit

#### ■ Local Registers within a User Function

In addition to the # registers and D registers, there are local registers that can be used only within user functions.

| Type | Name | Designation Method | Description |
|---|---|---|---|
| X | Function input registers | XBnnnnnh, XWnnnnn, XLnnnnn, XFnnnnn | These registers are used for inputs to functions.<br>Bit inputs: XB000000 to XB00000F<br>Integer inputs: XW00001 to XW00016<br>Double-length integers: XL00001 to XL00015<br>Real numbers: XF00001 to XF00015 |
| Y | Function output registers | YBnnnnnh, YWnnnnn, YLnnnnn, YFnnnnn | These registers are used for outputs from functions.<br>Bit outputs: YB000000 to YB00000F<br>Integer outputs: YW00001 to YW00016<br>Double-length integers: YL00001 to YL00015<br>Real numbers: YF00001 to YF00015 |
| Z | Function internal registers | ZBnnnnnh, ZWnnnnn, ZLnnnnn, ZFnnnnn | These are internal registers that are unique within each function. You can use them for internal processing in functions. |
| A | Function external registers | ABnnnnnh, AWnnnnn, ALnnnnn, AFnnnnn | These are external registers that use the address input values as the base addresses. When the address input value of an M or D register is provided by the source of the function call, then the registers of the source of the function call can be accessed from inside the function by using that address as the base. |

- n: decimal digit, h: hexadecimal digit

IMPORTANT

User functions can be called from any programs, any number of times.

When you call a user function, consider what values could be in the local registers, and perform initialization as needed.

Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

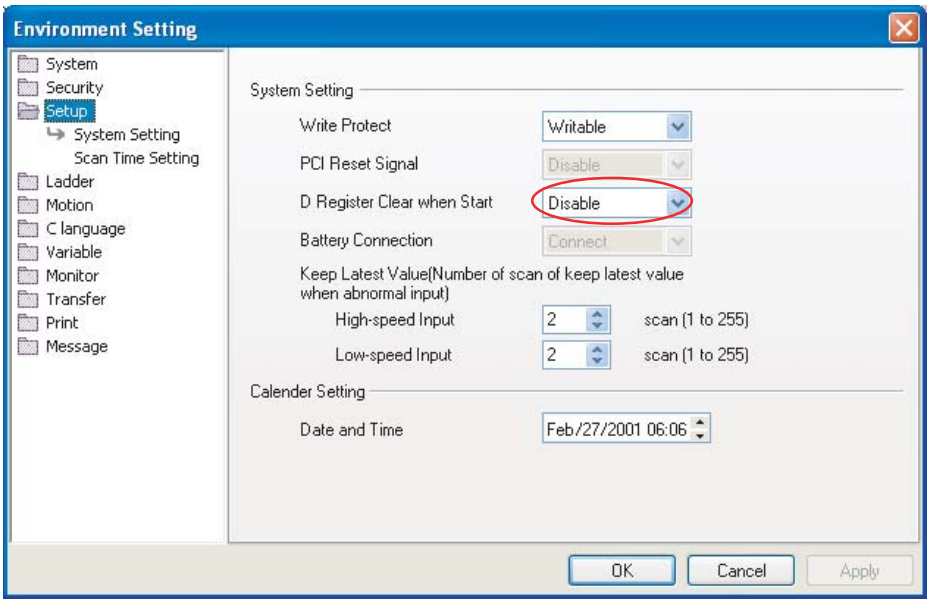
## ( 3 ) Precautions When Using Local Registers within a User Function

When you call a user function, consider what values should be in the local registers, and perform initialization as needed.

| Name | Precaution |
|---|---|
| X registers (function input registers) | If input values are not set, the values will be uncertain.<br>Do not use X registers that are outside of the range that is specified in the input definitions. |
| Y registers (function output registers) | If output values are not set, the values will be uncertain.<br>Always set the values of the range of Y registers that is specified in the output definitions. |
| Z registers (function internal registers) | When the function is called, the previously set values will be lost and the values will be uncertain.<br>These registers are not appropriate for instructions if the previous value must be retained.<br>Use them only after initializing them within the function. |
| # registers | These are constant registers. Their values cannot be changed. |
| D registers | When the function is called, the previously set values are preserved.<br>If a previous value is not necessary, initialize the value or use a Z register instead. D registers retain the data until the power is turned OFF.<br>The default values after startup depend on the setting of the **D Register Clear when Start** Option. For details, refer to ■ *Setting the D Register Clear When Start Option.* |

### ■ Setting the D Register Clear When Start Option

1. Select ***File*** – ***Environment Setting*** from the MPE720 Version 6 Window.
2. Select ***Setup*** – ***System Setting.***
3. Select **Enable** or **Disable** for the **D Register Clear when Start** Option.




Set Values
Disable: The initial values will be uncertain.
Enable: The initial values will be 0.

## 4.4.3 Data Types

### ( 1 ) List of Data Types

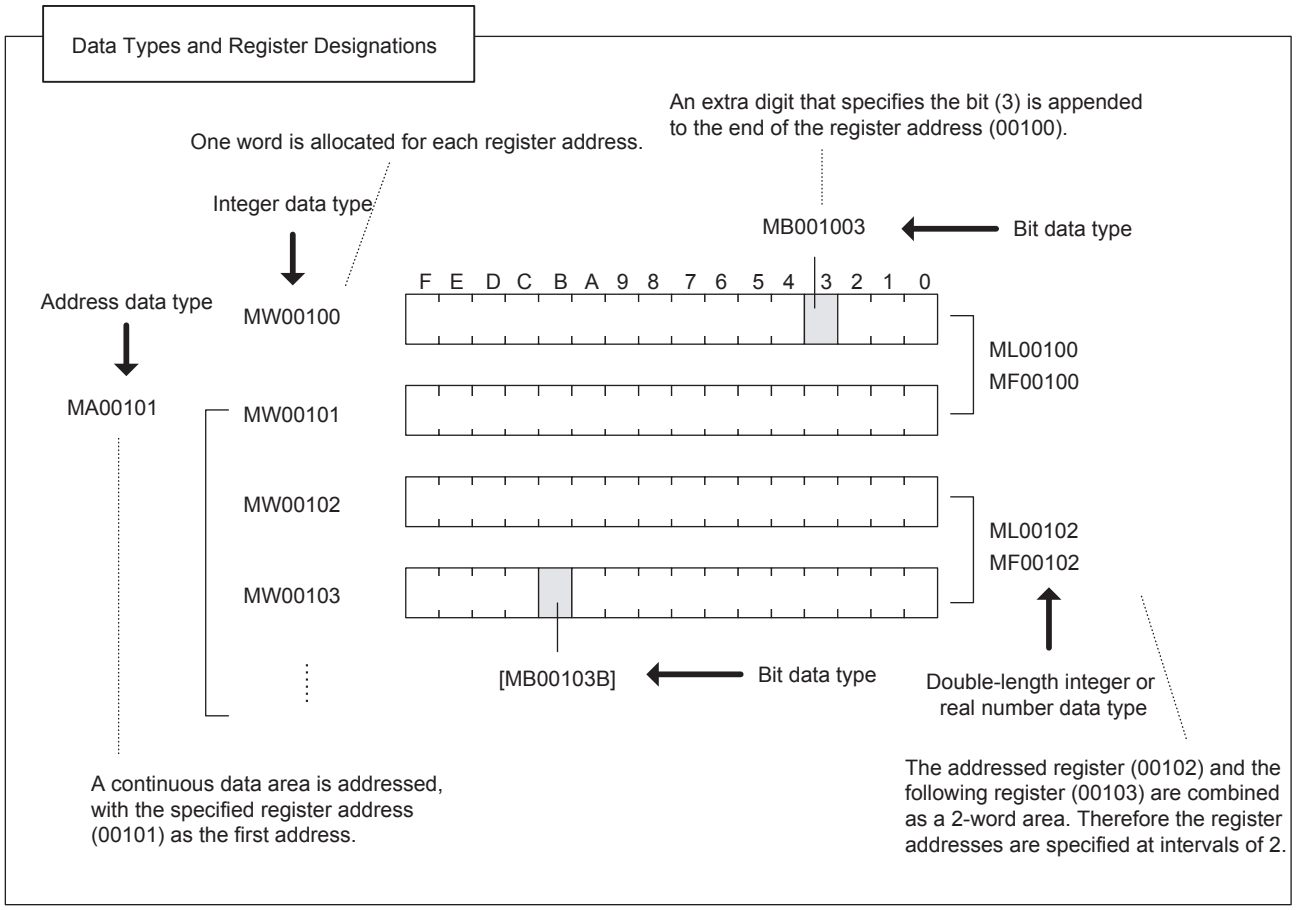
There are various data types that you can use depending on the purpose of the application: bit, integer, double-length integer, real number, and address.

| Symbol | Data Type | Range of Values | Remarks |
|---|---|---|---|
| B | Bit | 1 (ON) or 0 (OFF) | Used in relay circuits and to determine ON/OFF status. |
| W | Integer | -32,768 to 32,767 (8000 to 7FFF hex) | Used for numeric operations. The values in parentheses on the left are for logical operations. |
| L | Double-length integer | -2,147,483,648 to 2,147,483,647 (80000000 to 7FFFFFFF hex) | Used for numeric operations. The values in parentheses on the left are for logical operations. |
| F | Single-precision real number | ±(1.175E-38 to 3.402E+38) or 0 | Used for numeric operations. |
| A | Address | 0 to 32,767 | Used only as pointers for addressing. |

**IMPORTANT**

The MP3000-series Machine Controllers do not have separate registers for each data type. As shown in the following figure, the same address will access the same register even if the data type is different.

For example, MB001003, a bit address, and the MW00100, an integer address, have different data types, but they both access the same register, MW00100.



## ( 2 ) Precautions for Operations Using Different Data Types

If you perform an operation using different data types, the results will be different depending on the data type of the storage register, as described below.

### [ a ] Storing Real Number Data in an Integer Register

MW00100 = MF00200: The real number data is converted to integer data and stored in the destination register.
(00001) (1.234)

- There may be rounding error due to storing a real number in an integer register.
  Whether numbers are rounded or truncated when converting a real number to an integer can be set in the properties of the drawing. (See below.)

MW00100 = MF00200 + MF00202:
(0124) (123.48) (0.02) The result of the operation may be different depending on the value of the variable.
(0123) (123.49) (0.01)

### [ b ] Storing Real Number Data in a Double-length Integer Register

ML00100 = MF00200: The real number data is converted to integer data and stored in the destination register.
(65432) (65432.1)

### [ c ] Storing Double-length Integer Data in an Integer Register

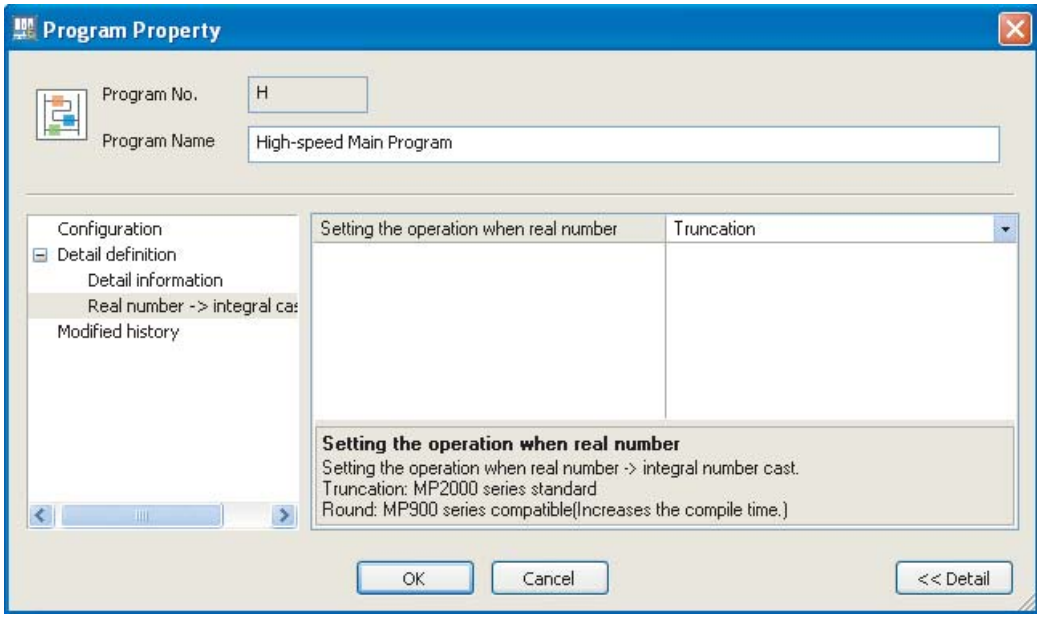
MW00100 = ML00200: The lower 16 bits of the double-length integer data are stored without change.
(-00001) (65535)

### [ d ] Storing Integer Data in a Double-length Integer Register

ML00100 = MW00200: The integer data is converted to double-length integer data and stored in the destination register.
(0001234) (1234)

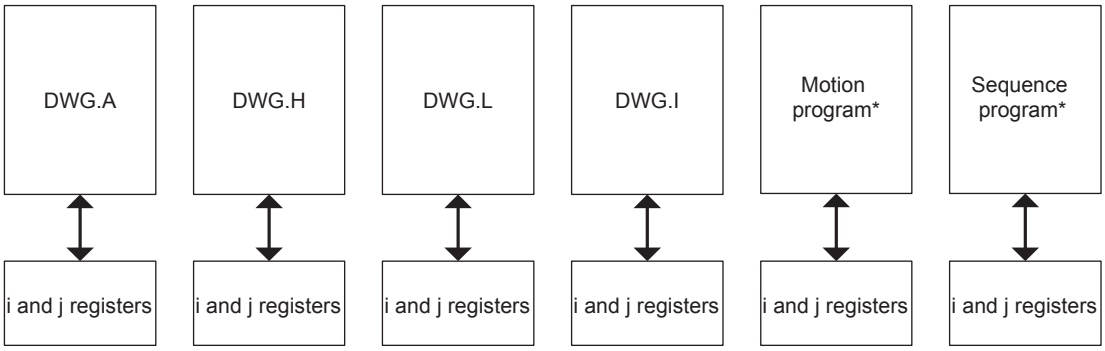
#### ■ Setting for Real Number Casting

The casting method (truncating or rounding) can be set in the detailed definitions in the Program Property Dialog Box. The method to use for real number casting is set for each drawing.



## 4.4.4 Index Registers (i, j)

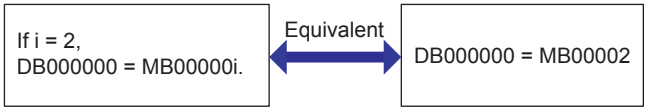
There are two index registers, i and j, that are used to modify relay and register addresses. The functions of i and j are identical.
There are index registers for each program type, as shown in the following figure.




* Motion programs and sequence programs have separate i and j registers for each task.
- Functions reference the i and j registers that belong to the calling drawing.
  For example, a function called by DWG.H will reference the i and j registers for DWG.H.

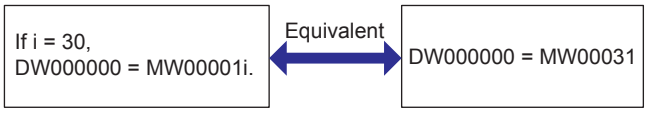
The operation for each register data type is described next.

### [ a ] Attaching an Index to a Bit Register



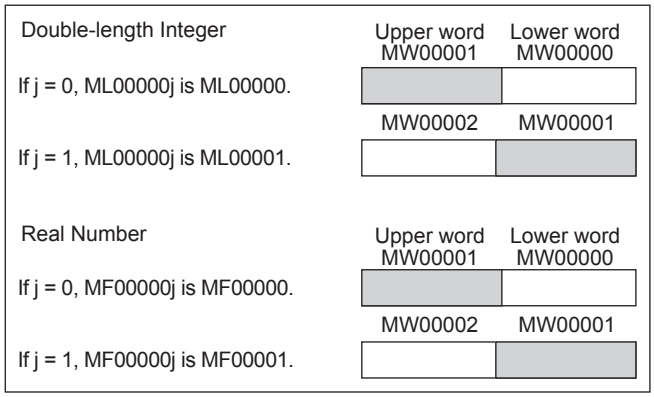

Using an index is the same as adding the value of i or j to the register address.
For example, if i = 2, MB00000i is the same as MB00002.

### [ b ] Attaching an Index to an Integer Register



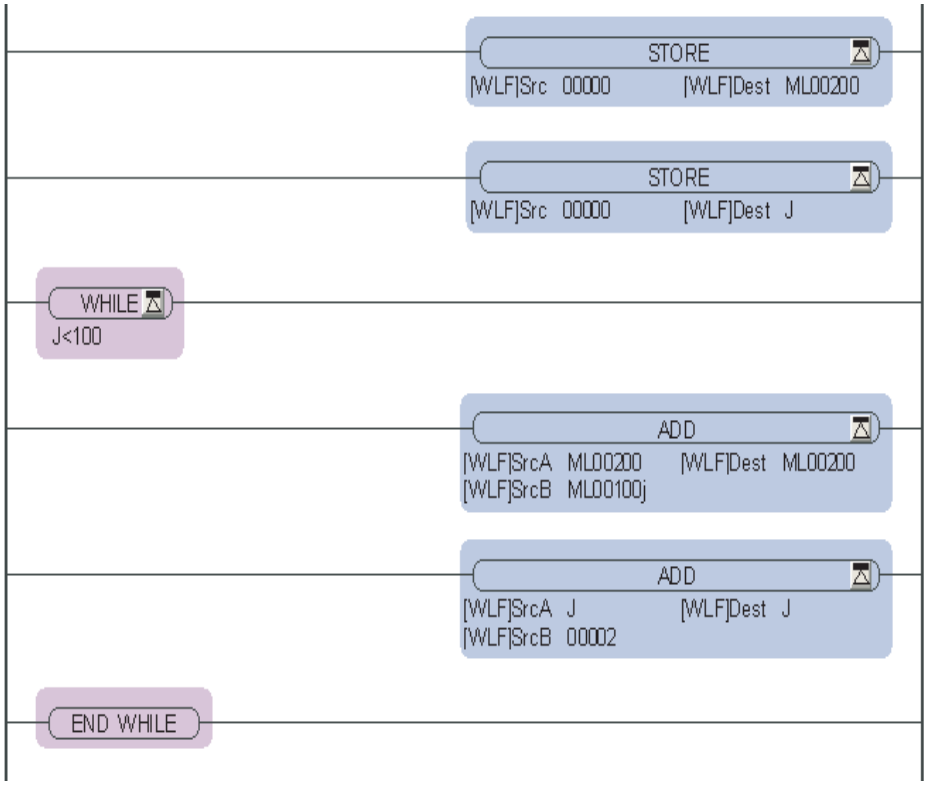

Using an index is the same as adding the value of i or j to the register address.
For example, if i = 30, MW00001i is the same as MW00031.

### [ c ] Attaching an Index to a Double-length Integer or a Real Number Register




Using an index is the same as adding the value of i or j to the register address.
For example, if j = 1, ML00000j is the same as ML00001.
Similarly, if j = 1, MF00000j is the same as MF00001.
In the case of double-length integers and real numbers, the one-word area of the register address and the one-word area of the register address + 1 are used together. Be careful of overlapping areas when indexing double-length integer or real number register addresses. For example, when using ML00000j with both j = 0 and j = 1, the one-word area of MW00001 will overlap.

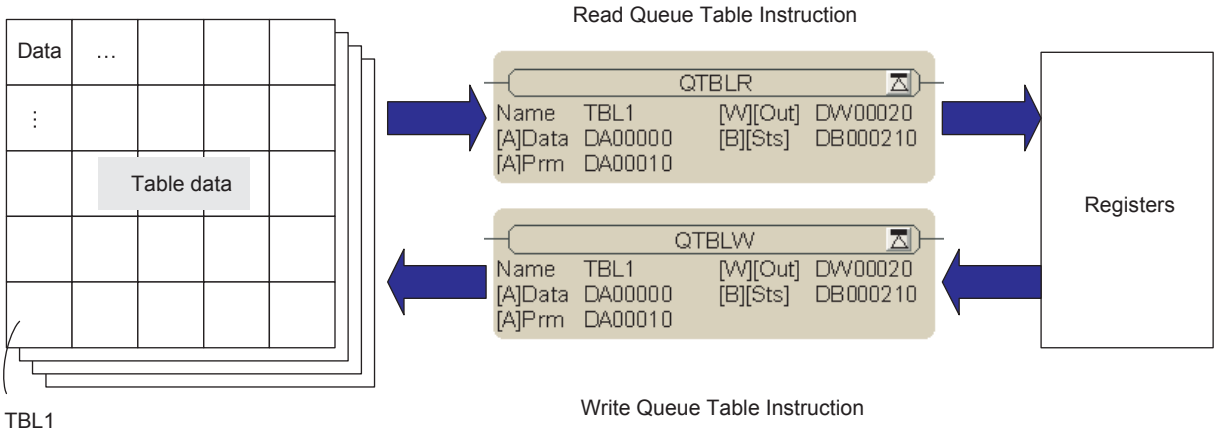
A programming example that uses indexed registers is shown below.
This example uses index j to find the total of the values in 50 registers from ML00100 to ML00198.



# 4.5 Table Data

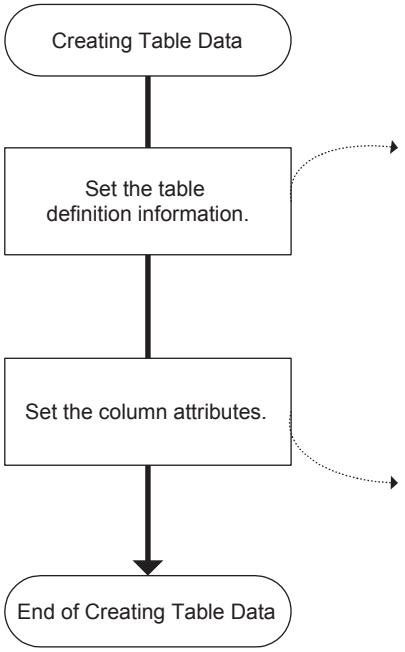
## 4.5.1 What Is Table Data?

Table data is data that is managed in tabular form. The data is stored separately from the registers.
Data can be copied from a table to registers or from registers to a table by executing table data manipulation instructions in the ladder program. Tables can also be used to hold data when there is not a sufficient range of registers.




## 4.5.2 Creating Table Data

Use the following procedure to create table data. The table definition information and column attributes that are set for table data are listed in the following table.




| Table Definition Information | Description |
|---|---|
| Table Name | This is the name of the table. |
| Table Type | Select an array-type or record-type table. |
| Number of Columns | This is the number of columns in the table. (10,000 columns max.) |
| Number of Rows | This is the number of rows in the table. (10,000 rows max.) |
| Table Comment | This is a comment for the table. |
| Table Data Storage Location | Select normal or battery backup. Refer to *2.1.2 Machine Controller Program Specifications* for details on the maximum size of tables and which models have battery backup storage. |

| Column Attribute | Description |
|---|---|
| Column Name | This is the name of the column. |
| Data Type | The data type can be integer, double-length integer, real number, or text string. |
| Size | This is the length of the data type. |
| Display Type | The display type can be binary, decimal, hexadecimal, real number, or text string. |
| Column Comment | This is a comment for the column. |

INFO

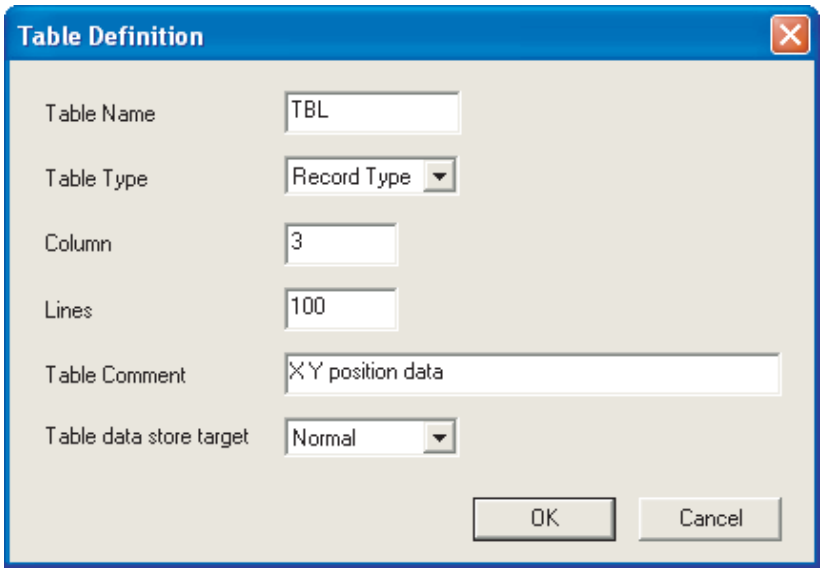
You can select one of the following table types when you create table data.
- Array type: Specifies a table where all columns have the same attributes.
- Record type: Specifies a table where each column has a different attribute.

You can select one of the following table data storage locations.
- Normal: Refer to *2.1.2 Machine Controller Program Specifications* for the maximum program size. The maximum size per table is 5 MB.
- Battery backup: Refer to *2.1.2 Machine Controller Program Specifications* for the maximum size of table data that can be backed up with the battery. The maximum size per table is 3 MB.

## ■ Procedure to Create Table Data

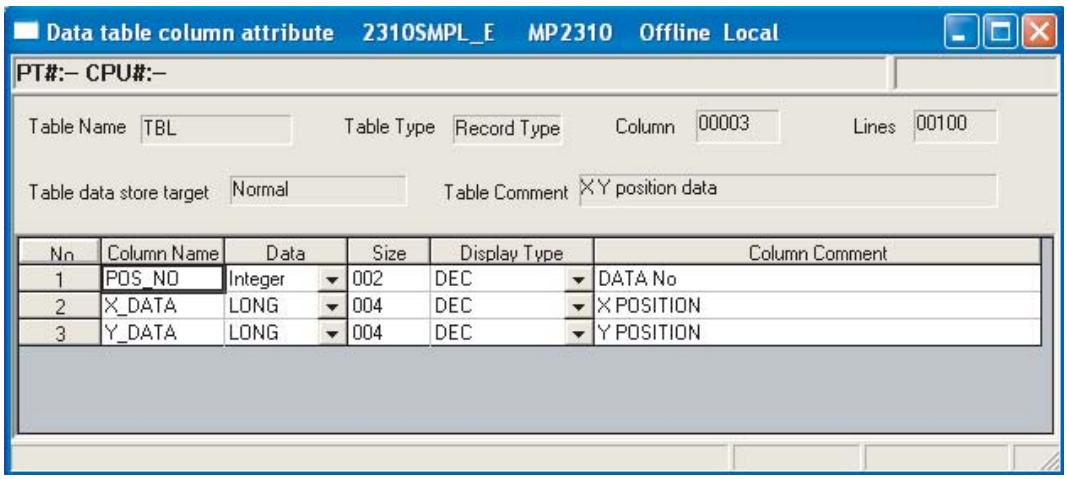
**1.** Select ***File - Open*** – ***Define Data Table*** – ***Data Table Map*** in the Module Configuration Definitions Window. The Table Data Store Target Dialog Box will be displayed.

**2.** Select ***File*** – ***Create New*** from the menu bar. The Table Definition Dialog Box will be displayed. Set the table definition information and click the **OK** Button.




**3.** The Data Table Column Attribute Dialog Box will be displayed. Set the table data column attributes, and then save them.

- If the table is set to an array-type table, set only one row of column attributes.



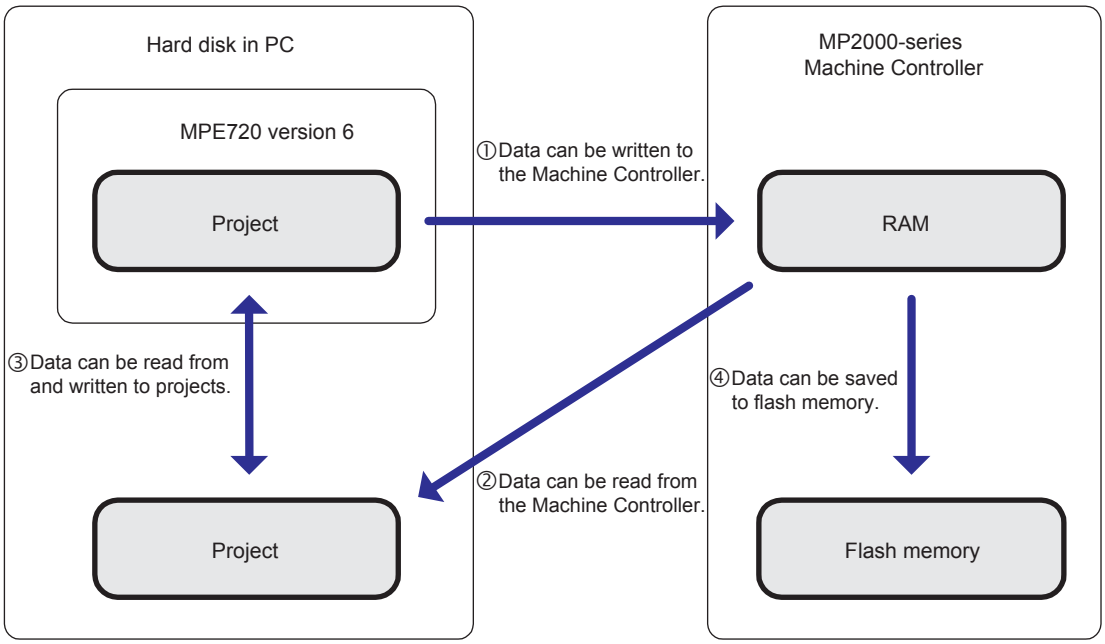

The Table Data Store Target Dialog Box that was displayed in step 1 will show the table that you created. This concludes the creation of the data table.

INFO

When a table is created, the contents are initialized to 0.
Select the table that was created in the Table Data Store Target Dialog Box, and click the **Table Data** Button to read or write table data.
Use the table instructions to perform operations on the table data from a ladder program.

# 4.6 Transferring Data

You can perform one of the four operations that are shown in the following figure to transfer data.




### ① Writing Data to a Machine Controller

You can transfer the project data that was created offline to RAM in the Machine Controller.

### ② Reading Data from the Machine Controller.

You can transfer data from the Machine Controller to a project on the hard disk of the PC.

### ③ Reading Data from and Writing Data to Projects

You can transfer data between projects on the hard disk of the PC.

### ④ Saving Data to Flash Memory

You can transfer the data in RAM in the Machine Controller to flash memory.

IMPORTANT

Always save the data to flash memory after you transfer it to the MP2000-series Machine Controller.
Failure to save the data to flash memory will result in losing the data that was transferred when the power is turned OFF and ON again, causing the Machine Controller to run on the data that was last saved in the flash memory.

# 4.7 Setting the High-speed/Low-speed Scan Times

## ( 1 ) What Are the Scan Times?

With an MP2000-series Machine Controller, both the high-speed scan and low-speed scan can be set. The high-speed scan time is the cycle at which high-speed drawings are executed. The low-speed scan time is the cycle at which low-speed drawings are executed. The following table shows the possible set values and default values for each scan time.

| Item | Possible Set Values | Default |
|---|---|---|
| High-speed Scan Time | 0.5 to 32 ms (in 0.5-ms increments) | 10.0 ms |
| Low-speed Scan Time | 2.0 to 300.0 ms (in 0.5-ms increments) | 200.0 ms |

- The possible set values and default values depend on the model. Refer to the user's manual for the Module you are using for details.

## ( 2 ) Scan Time Set Value Precautions

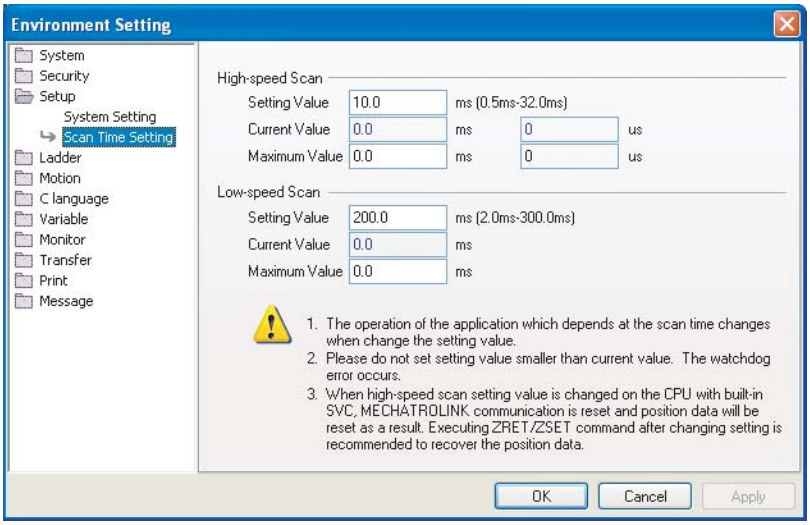
Observe the following precautions when setting the high-speed scan time and low-speed scan time.

- Set the scan set value so that it is 1.25 times greater than the maximum value.
  - If the scan set value is too close to the maximum value, the refresh rate of the MPE720 window will noticeably drop and can cause communications timeout errors to occur. If the maximum value exceeds the scan set value, a watchdog error may occur and cause the Machine Controller system to shut down.
- If you are using MECHATROLINK-II or MECHATROLINK-III, set values that are an integral multiple of the communications cycle. If you change the communications cycle, check the scan time set values.
- Do not change the scan set value while the Servo is ON. Never change the scan set values while an axis is in motion (i.e., while the motor is rotating). Doing so may cause the motor to rotate out of control.
- After changing or setting the scan times, make sure to save the data to flash memory.

## ( 3 ) Checking and Setting the Scan Times

You can check the current and maximum values of the scan times and the set values of the scan times, and you can set the scan times in the following dialog box of MPE720 Version 6.0.

Select ***File – Environment Setting – Setup – Scan Time Setting.***

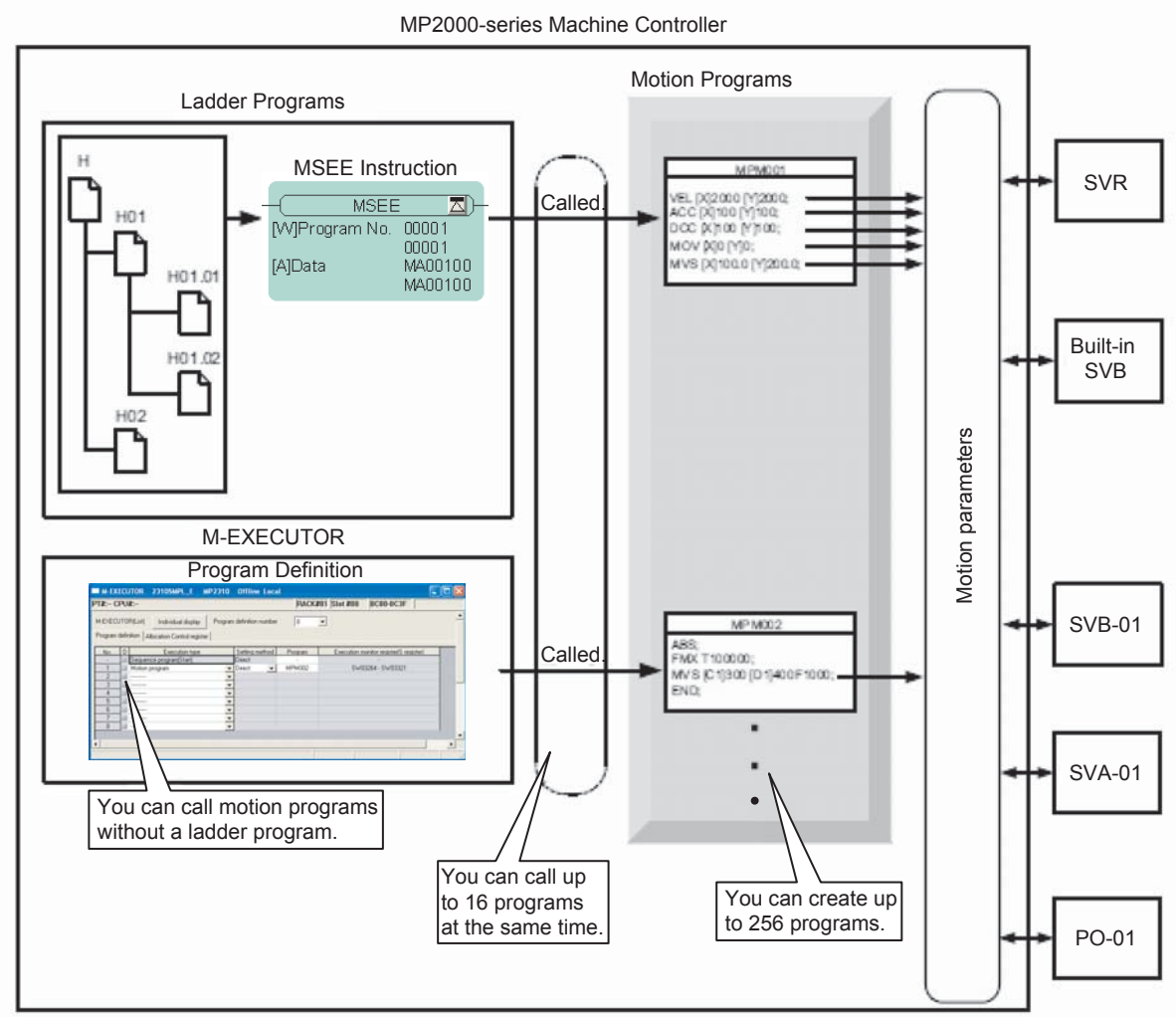


# 4.8 Advanced Programming

## 4.8.1 Motion Programs

A motion program is written in a text-based motion language. In addition to basic motion control and operations, motion programs can also be used to easily program complex movements, such as linear interpolation and circular interpolation.

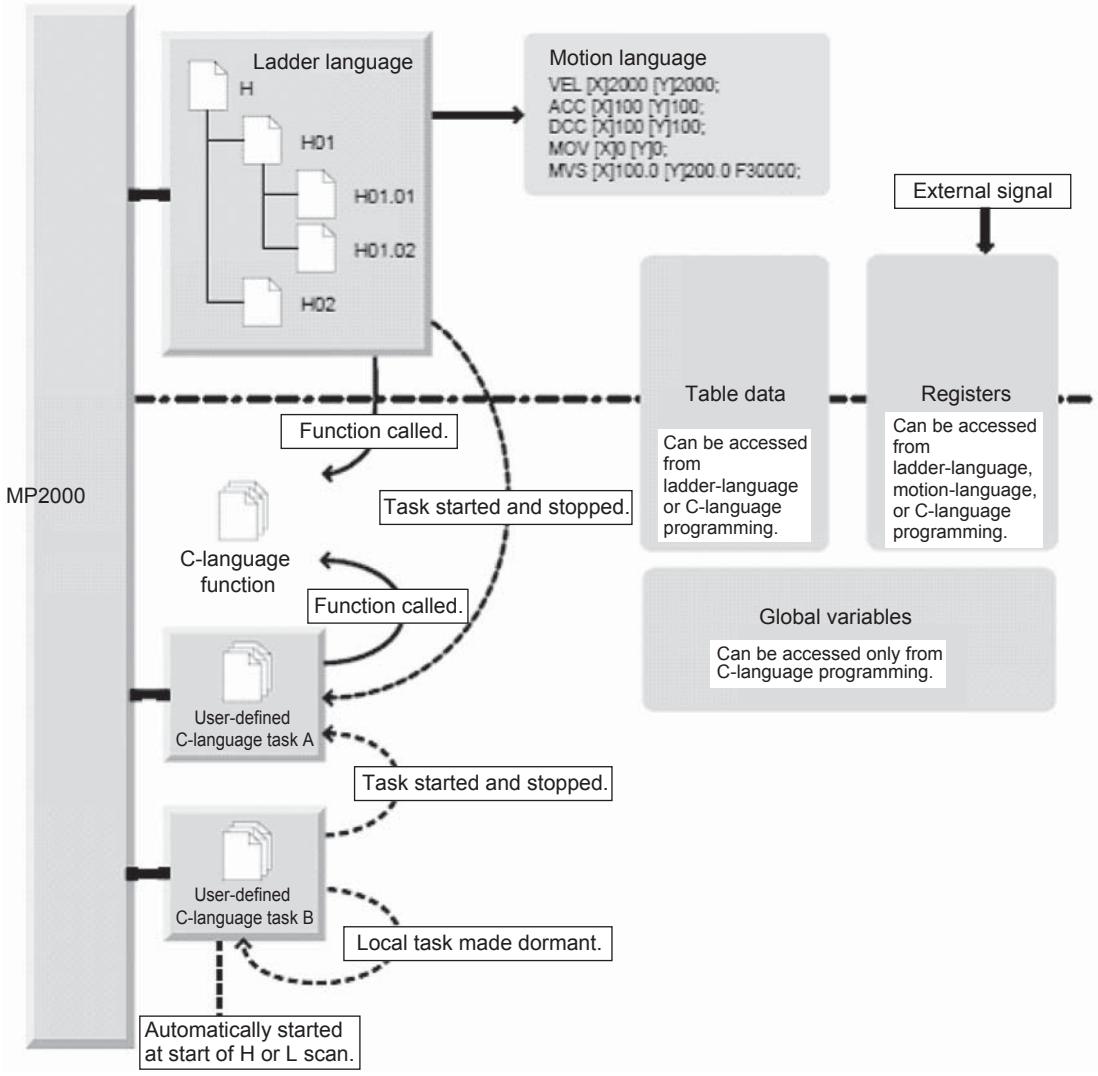
You can execute motion programs either by placing MSEE instructions in ladder programming in high-speed drawings, or by registering the motion programs in the Program Definition Tab Page for the M-EXECUTOR.




For details on motion programs, refer to the *Machine Controller MP2000 Series User's Manual for Motion Programming* (Manual No.: SIEP C880700 38).

## 4.8.2 C-language Programs

You can use the MP2000-series Machine Controller Embedded C-language Programming Package to use C-language functions and C-language tasks in addition to ladder programs and motion programs.
You can call C-language functions and start and stop C-language tasks from the ladder programs.
The following configuration is for using C-language programming.




For details on C-language programming, refer to the *Machine Controller MP2000 Series Embedded C-Language Programming Package Development Guide* (Manual No.: SIEP C880700 25).

## 4.8.3 Security

MPE720 version 6 has the following security features. You can use these security features for data protection by specifying access privileges for individual projects and program drawings.

### ■ User Administration (User Name and Password Setting)

You can register and change the name of the users who can open projects.
If the setting is performed while the Machine Controller is online, the setting will provide access privileges to the Machine Controller.

### ■ Project Password Setting

You can set a password for opening a project file.

### ■ Program Password Setting

You can set a password for opening ladder programs and motion programs. A password can be set for each program.

### ■ Online Security Setting

You can set a security key (i.e., a password) and privilege levels for reading data from a Machine Controller. This allows you to restrict the ability to read the program data from the Machine Controller or the ability to open the programs to users who have the specified level of privilege or a higher privilege.

Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for detailed setting procedures for security.

# 4.8.4 Tracing

MPE720 version 6 has three trace modes.

### ■ Realtime Tracing

You can monitor specified registers on a graph in real time.

### ■ Data Tracing

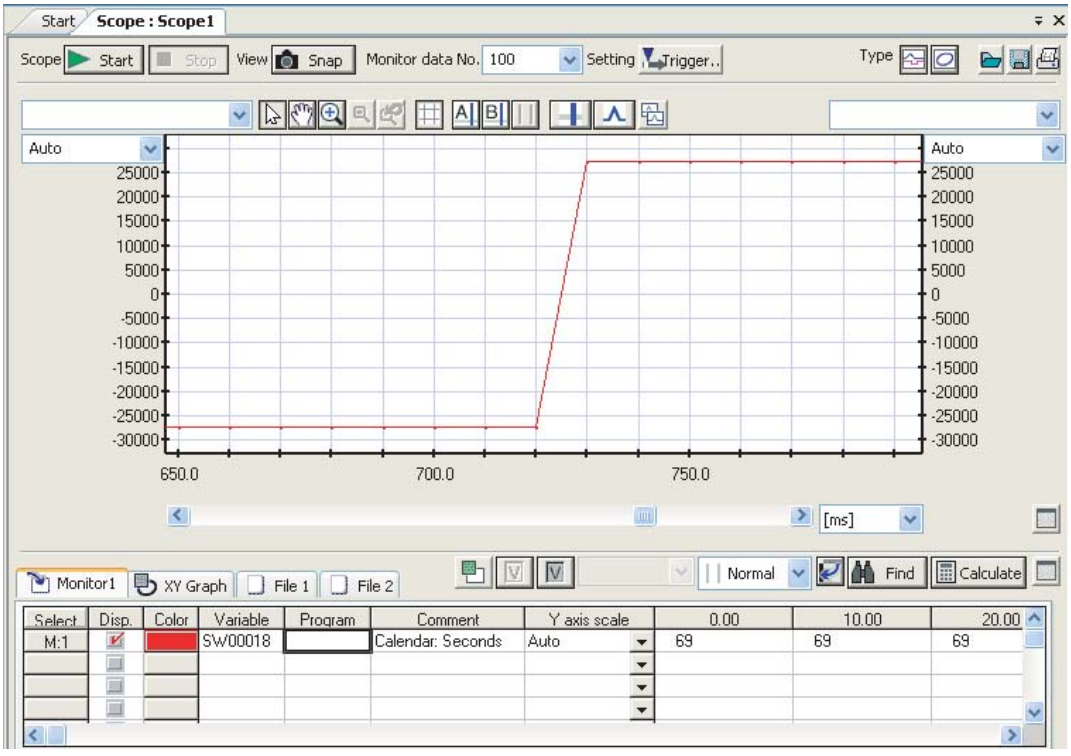
You can have the Machine Controller collect data for specified registers during a specified time period, and perform operations on that data and plot it on a graph.
This allows you to analyze register data that is acquired during specific time periods to debug ladder programs.

### ■ XY Tracing

This trace mode acquires the position data of the X axis and Y axis every scan, and displays the data in a 2-dimensional graph.

All three modes support exporting the trace data to CSV files.
Use tracing to check operation and to debug the ladder programs and motion programs.

## Data Tracing Display Example




Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for detailed setting procedures for tracing.

# 5

# Instructions

This chapter describes the ladder programming instructions in detail.


5.1 How to Read the Instructions ----- 5-4

5.2 Relay Circuit Instructions ----- 5-5

5.2.1 NO Contact (NOC) ----- 5-5
5.2.2 NC Contact (NCC) ----- 5-6
5.2.3 10-ms ON-Delay Timer (TON[10ms]) ----- 5-7
5.2.4 10-ms OFF-Delay Timer (TOFF[10ms]) ----- 5-9
5.2.5 1-s ON-Delay Timer (TON[1s]) ----- 5-11
5.2.6 1-s OFF-Delay Timer (TOFF[1s]) ----- 5-13
5.2.7 Rising-edge Pulses (ON-PLS) ----- 5-15
5.2.8 Falling-edge Pulses (OFF-PLS) ----- 5-17
5.2.9 Coil (COIL) ----- 5-19
5.2.10 Set Coil (S-COIL) ----- 5-20
5.2.11 Reset Coil (R-COIL) ----- 5-21

5.3 Numeric Operation Instructions ----- 5-22

5.3.1 Store (STORE) ----- 5-22
5.3.2 Add (ADD (+)) ----- 5-24
5.3.3 Extended Add (ADDX (++)) ----- 5-26
5.3.4 Subtract (SUB (–)) ----- 5-28
5.3.5 Extended Subtract (SUBX (– –)) ----- 5-30
5.3.6 Multiply (MUL (x)) ----- 5-32
5.3.7 Divide (DIV (÷)) ----- 5-34
5.3.8 Integer Remainder (MOD) ----- 5-36
5.3.9 Real Remainder (REM) ----- 5-38
5.3.10 Increment (INC) ----- 5-40
5.3.11 Decrement (DEC) ----- 5-42
5.3.12 Add Time (TMADD) ----- 5-44
5.3.13 Subtract Time (TMSUB) ----- 5-46
5.3.14 Spend Time (SPEND) ----- 5-48
5.3.15 Invert Sign (INV) ----- 5-51
5.3.16 One's Complement (COM) ----- 5-52
5.3.17 Absolute Value (ABS) ----- 5-53
5.3.18 Binary Conversion (BIN) ----- 5-54
5.3.19 BCD Conversion (BCD) ----- 5-55
5.3.20 Parity Conversion (PARITY) ----- 5-56
5.3.21 ASCII Conversion 1 (ASCII) ----- 5-57
5.3.22 ASCII Conversion 2 (BINASC) ----- 5-59
5.3.23 ASCII Conversion 3 (ASCBIN) ----- 5-61

5.4 Logic Operations and Comparison Instructions ----- 5-63
5.4.1 Inclusive AND (AND) ----- 5-63
5.4.2 Inclusive OR (OR) ----- 5-65
5.4.3 Exclusive OR (XOR) ----- 5-67
5.4.4 Less Than (<) ----- 5-69
5.4.5 Less Than or Equal (≤) ----- 5-70
5.4.6 Equal (=) ----- 5-71
5.4.7 Not Equal (≠) ----- 5-72
5.4.8 Greater Than or Equal (≥) ----- 5-73
5.4.9 Greater Than (>) ----- 5-74
5.4.10 Range Check (RCHK) ----- 5-75
5.5 Program Control Instructions ----- 5-77
5.5.1 Call Sequence Program (SEE) ----- 5-77
5.5.2 Call Motion Program (MSEE) ----- 5-78
5.5.3 Call User Function (FUNC) ----- 5-80
5.5.4 Direct Input String (INS) ----- 5-81
5.5.5 Direct Output String (OUTS) ----- 5-84
5.5.6 Call Extended Program (XCALL) ----- 5-87
5.5.7 WHILE Construct (WHILE, END_WHILE) ----- 5-88
5.5.8 FOR Construct (FOR, END_FOR) ----- 5-91
5.5.9 IF Construct (IF, END_IF) ----- 5-93
5.5.10 IF-ELSE Construct (IF, ELSE, END_IF) ----- 5-95
5.5.11 Expression (EXPRESSION) ----- 5-97
5.6 Basic Function Instructions ----- 5-99
5.6.1 Square Root (SQRT) ----- 5-99
5.6.2 Sine (SIN) ----- 5-101
5.6.3 Cosine (COS) ----- 5-103
5.6.4 Tangent (TAN) ----- 5-105
5.6.5 Arc Sine (ASIN) ----- 5-106
5.6.6 Arc Cosine (ACOS) ----- 5-107
5.6.7 Arc Tangent (ATAN) ----- 5-108
5.6.8 Exponential (EXP) ----- 5-109
5.6.9 Natural Logarithm (LN) ----- 5-110
5.6.10 Common Logarithm (LOG) ----- 5-111
5.7 Data Shift Instructions ----- 5-112
5.7.1 Bit Rotate Left (ROTL) ----- 5-112
5.7.2 Bit Rotate Right (ROTR) ----- 5-114
5.7.3 Move Bit (MOVB) ----- 5-116
5.7.4 Move Word (MOVW) ----- 5-118
5.7.5 Exchange (XCHG) ----- 5-120
5.7.6 Table Initialization (SETW) ----- 5-122
5.7.7 Byte-to-word Expansion (BEXTD) ----- 5-124
5.7.8 Word-to-byte Compression (BPRESS) ----- 5-126
5.7.9 Binary Search (BSRCH) ----- 5-128
5.7.10 Sort (SORT) ----- 5-130
5.7.11 Bit Shift Left (SHFTL) ----- 5-132
5.7.12 Bit Shift Right (SHFTR) ----- 5-134
5.7.13 Copy Word (COPYW) ----- 5-136
5.7.14 Byte Swap (BSWAP) ----- 5-138
5.8 DDC Instructions ----- 5-139
5.8.1 Dead Zone A (DZA) ----- 5-139
5.8.2 Dead Zone B (DZB) ----- 5-141
5.8.3 Upper/Lower Limit (LIMIT) ----- 5-143
5.8.4 PI Control (PI) ----- 5-145
5.8.5 PD Control (PD) ----- 5-150
5.8.6 PID Control (PID) ----- 5-156

5.8.7 First-order Lag (LAG) ---------- 5-161
5.8.8 Phase Lead Lag (LLAG) ---------- 5-164
5.8.9 Function Generator (FGN) ---------- 5-167
5.8.10 Inverse Function Generator (IFGN) ---------- 5-172
5.8.11 Linear Accelerator/Decelerator 1 (LAU) ---------- 5-177
5.8.12 Linear Accelerator/Decelerator 2 (SLAU) ---------- 5-184
5.8.13 Pulse Width Modulation (PWM) ---------- 5-194

5.9 Table Manipulation Instructions ---------- 5-197
5.9.1 Read Table Block (TBLBR) ---------- 5-197
5.9.2 Write Table Block (TBLBW) ---------- 5-200
5.9.3 Search for Table Row (TBLSRL) ---------- 5-203
5.9.4 Search for Table Column (TBLSRC) ---------- 5-206
5.9.5 Clear Table Block (TBLCL) ---------- 5-209
5.9.6 Move Table Block (TBLMV) ---------- 5-212
5.9.7 Read Queue Table (QTBLR and QTBLRI) ---------- 5-215
5.9.8 Write Queue Table (QTBLW and QTBLWI) ---------- 5-219
5.9.9 Clear Queue Table Pointers (QTBLCL) ---------- 5-223

5.10 System Function Instructions ---------- 5-225
5.10.1 Counter (COUNTER) ---------- 5-225
5.10.2 First-in First-out (FINFOUT) ---------- 5-228
5.10.3 Trace (TRACE) ---------- 5-232
5.10.4 Read Data Trace (DTRC-RD) ---------- 5-234
5.10.5 Read Inverter Trace (ITRC-RD) ---------- 5-238
5.10.6 Send Message (MSG-SND) ---------- 5-241
5.10.7 Receive Message (MSG-RCV) ---------- 5-253
5.10.8 Write Inverter Parameter (ICNS-WR) ---------- 5-261
5.10.9 Read Inverter Parameter (ICNS-RD) ---------- 5-266
5.10.10 Write SERVOPACK Parameter (MLNK-SVW) ---------- 5-270
5.10.11 Write Motion Register (MOTREG-W) ---------- 5-275
5.10.12 Read Motion Register (MOTREG-R) ---------- 5-278

5.11 C-language Control Instructions ---------- 5-281
5.11.1 Call C-language Function (C-FUNC) ---------- 5-281
5.11.2 C-language Task Control (TSK-CTRL) ---------- 5-283

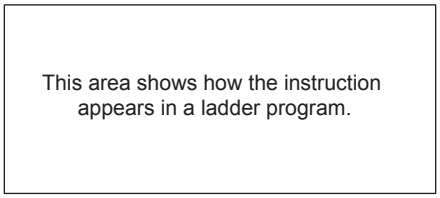
# 5.1 How to Read the Instructions

This chapter describes each instruction using the following format.

## ( 1 ) Operation

The operation performed by the instruction is described.
Figures are used to show the operation performed by the instruction.

## ( 2 ) Format




Icon:

Shows the icon used in the MPE720.

Key entry:

Shows the shortcut key combination used in the Ladder Editor.

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| The name of the parameter that appears in the ladder programs is given. | × | ○ | ○ | ○ | × | ○ | ○ |

×: This data type cannot be used.
○: All registers with this data type can be used.

## ( 3 ) Programming Example

This section gives a ladder programming example that uses the instruction.

## ( 4 ) Additional Information

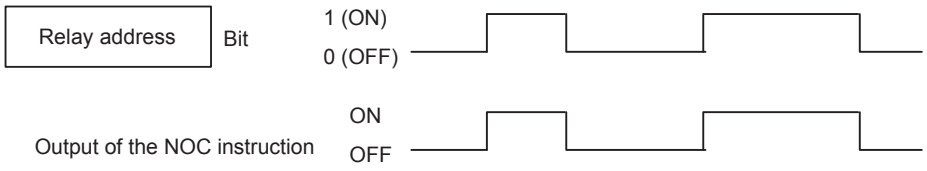
This section contains additional information about the instruction. It is omitted if there is no additional information that is required for the instruction.

# 5.2 Relay Circuit Instructions

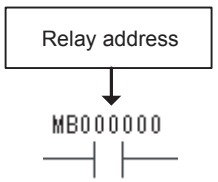
## 5.2.1 NO Contact (NOC)

### ( 1 ) Operation

The NOC instruction outputs ON whenever the bit with the specified relay address is 1 (ON).
The NOC instruction outputs OFF when the bit is 0 (OFF).




### ( 2 ) Format




Icon: ⊣⊢

Key entry: ][

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Relay address | O | × | × | × | × | × | × |

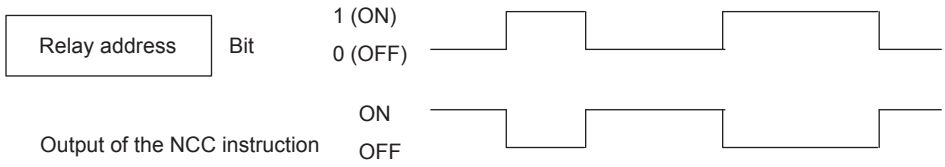
### ( 3 ) Programming Example

The DB000001 output coil is ON whenever the DB000000 relay in the NOC instruction is ON.

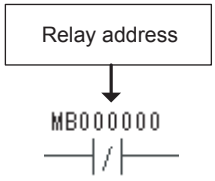
## 5.2.2 NC Contact (NCC)

### ( 1 ) Operation

The NCC instruction outputs OFF whenever the bit with the specified relay address is 1 (ON).
The NCC instruction outputs ON when the bit is 0 (OFF).




### ( 2 ) Format




Icon:

Key entry: ]/

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Relay address | O | × | × | × | × | × | × |

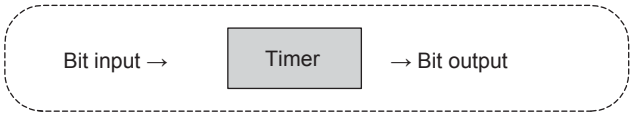
### ( 3 ) Programming Example

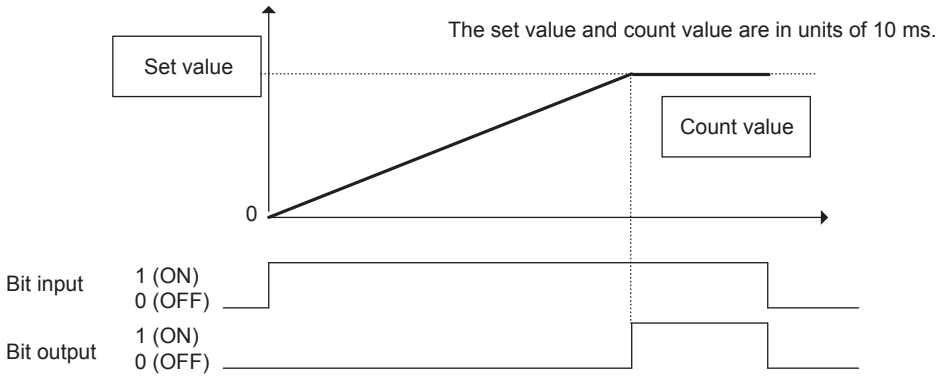
The DB000001 coil is ON whenever the DB000000 relay in the NCC instruction is OFF.

## 5.2.3 10-ms ON-Delay Timer (TON[10ms])

### ( 1 ) Operation

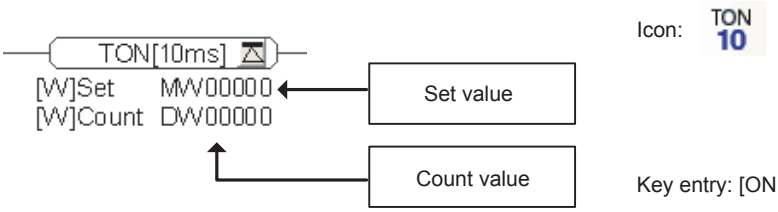
The timer counts the time whenever the timer bit input is 1 (ON). The bit output is set to 1 (ON) when the count value equals the set value.
If the bit input changes to 0 (OFF) during counting, the timer will stop counting. If the bit input changes to 1 (ON) again, the timer starts counting again from the beginning (i.e., from 0). The actual counted time (in units of 10 ms) is stored in the Count register.




- The counting error is 10 ms or less.

### ( 2 ) Format

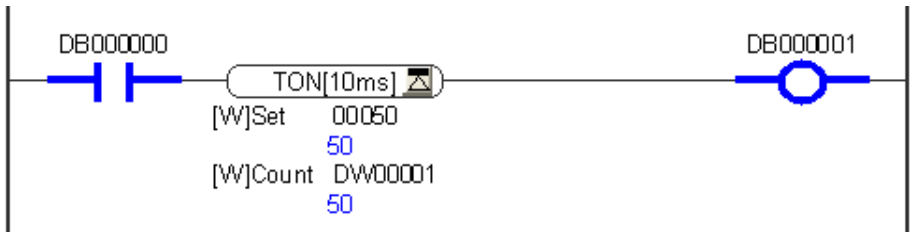



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Set value (Set) | × | O | × | × | × | × | O |
| Count value (Count) | × | O* | × | × | × | × | × |

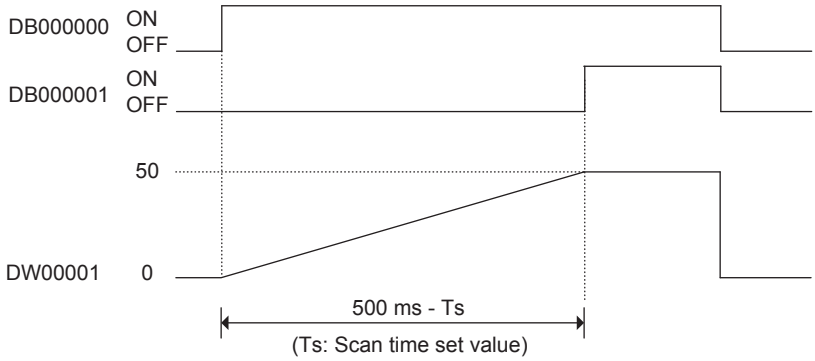
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the set value of the TON instruction is 50, and the count value is stored in the DW00001 register.
The DB000001 coil will turn ON after the DB000000 relay stays ON for 500 ms.



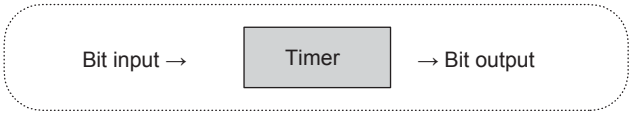

The timing chart is shown below.

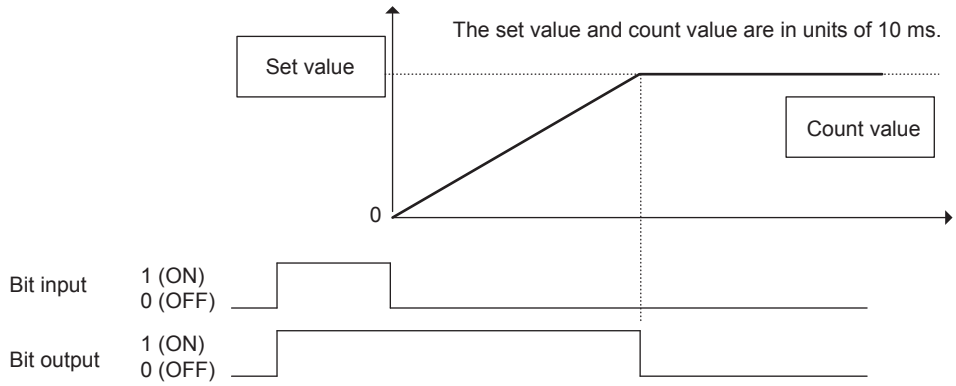


## 5.2.4 10-ms OFF-Delay Timer (TOFF[10ms])

### ( 1 ) Operation

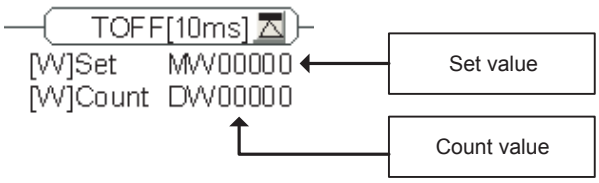
The timer counts the time whenever the timer bit input is 0 (OFF). The bit output is set to 0 (OFF) when the count value equals the set value.
If the bit input changes to 1 (ON) during counting, the timer will stop counting. If the bit input changes to 0 (OFF) again, the timer starts counting again from the beginning (i.e., from 0). The actual counted time (in units of 10 ms) is stored in the Count register.




- The counting error is 10 ms or less.

### ( 2 ) Format




Icon: TOFF 10

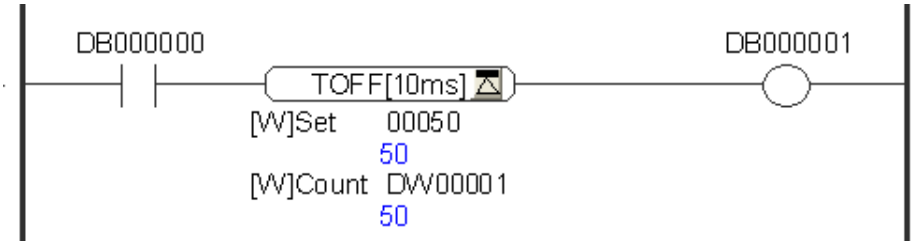
Key entry: [OFF

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Set value (Set) | × | O | × | × | × | × | O |
| Count value (Count) | × | O* | × | × | × | × | × |

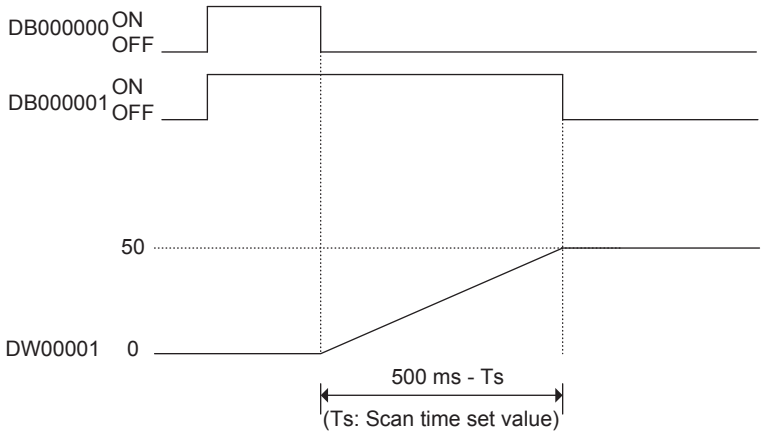
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the set value of the TOFF instruction is 50, and the count value is stored in the DW00001 register.
The DB000001 coil will turn OFF after the DB000000 relay stays OFF for 500 ms.



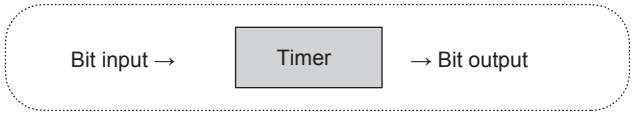

The timing chart is shown below.

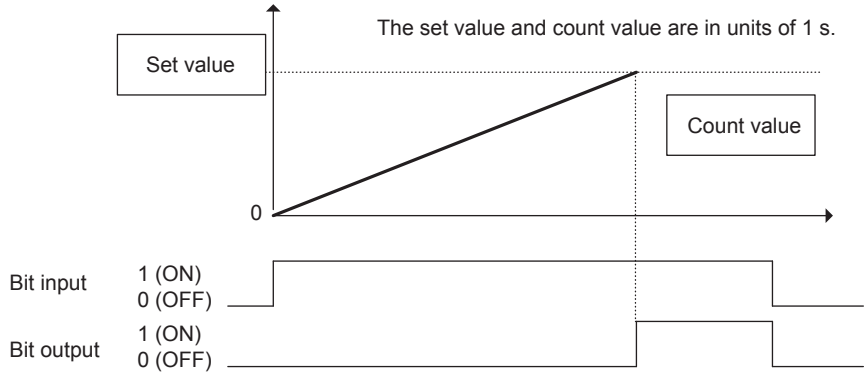


## 5.2.5 1-s ON-Delay Timer (TON[1s])

### ( 1 ) Operation

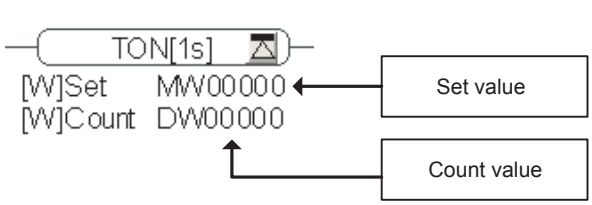
The timer counts the time whenever the timer bit input is 1 (ON). The bit output is set to 1 (ON) when the count value equals the set value.
If the bit input changes to 0 (OFF) during counting, the timer will stop counting. If the bit input changes to 1 (ON) again, the timer starts counting again from the beginning (i.e., from 0). The actual counted time (in units of 1 s) is stored in the Count register.




- The counting error is 1 s or less.

### ( 2 ) Format




Icon: TON 1s

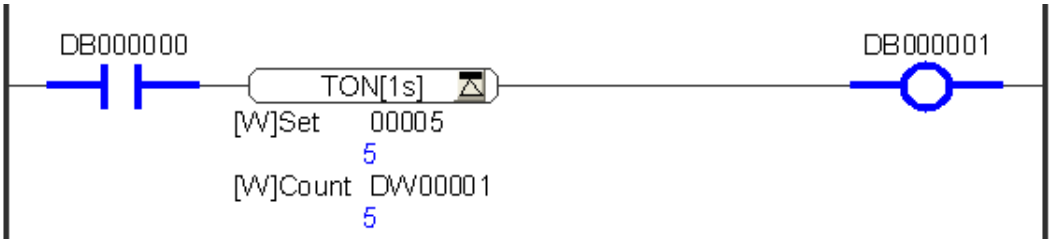
Key entry: [SON

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Set value (Set) | × | O | × | × | × | × | O |
| Count value (Count) | × | O* | × | × | × | × | × |

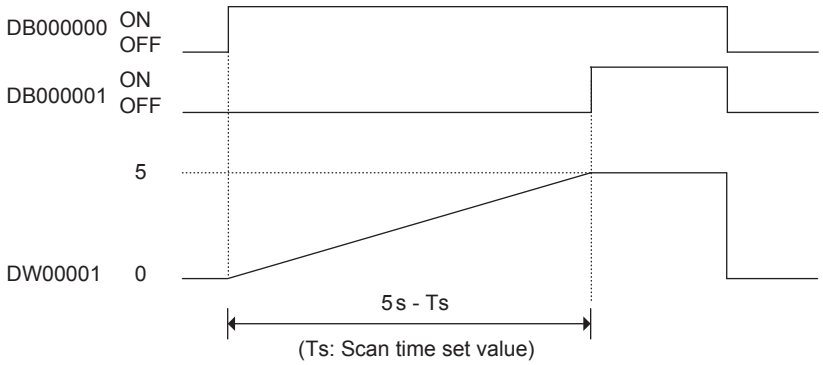
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the set value of the TON instruction is 5, and the count value is stored in the DW00001 register.
The DB000001 coil will turn ON after the DB000000 relay stays ON for 5 s.



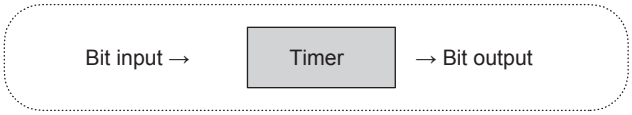

The timing chart is shown below.

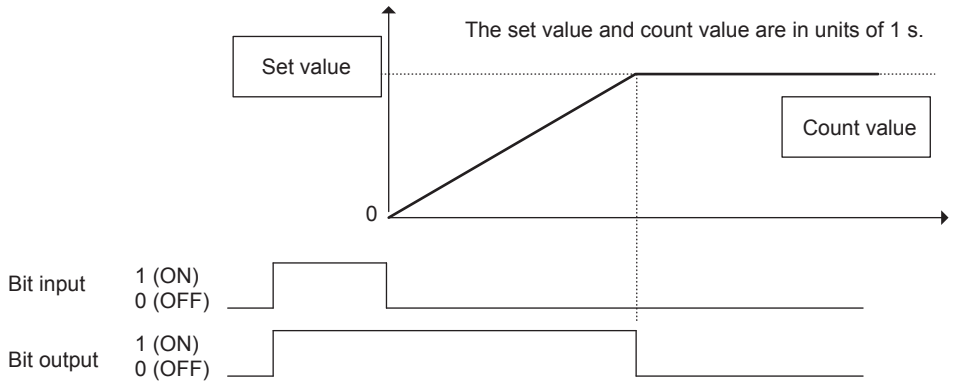


## 5.2.6 1-s OFF-Delay Timer (TOFF[1s])

### ( 1 ) Operation

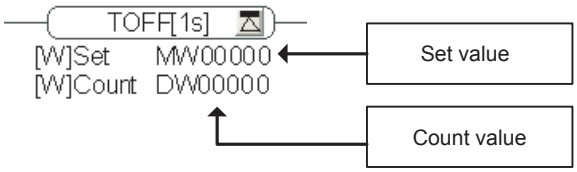
The timer counts the time whenever the timer bit input is 0 (OFF). The bit output is set to 0 (OFF) when the count value equals the set value.
If the bit input changes to 1 (ON) during counting, the timer will stop counting. If the bit input changes to 0 (OFF) again, the timer starts counting again from the beginning (i.e., from 0). The actual counted time (in units of 1 s) is stored in the Count register.




- The counting error is 1 s or less.

### ( 2 ) Format




Icon: TOFF 1s

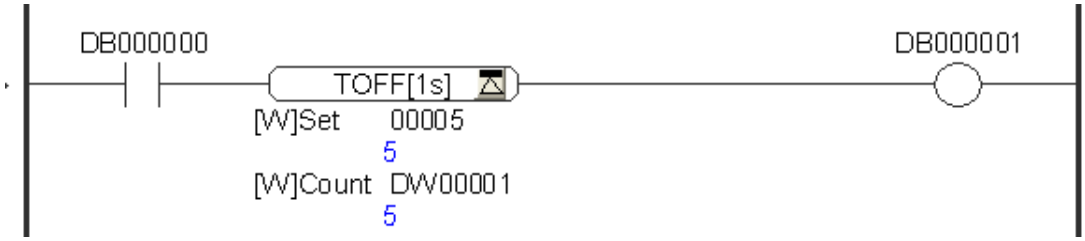
Key entry: [SOFF

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Set value (Set) | × | O | × | × | × | × | O |
| Count value (Count) | × | O* | × | × | × | × | × |

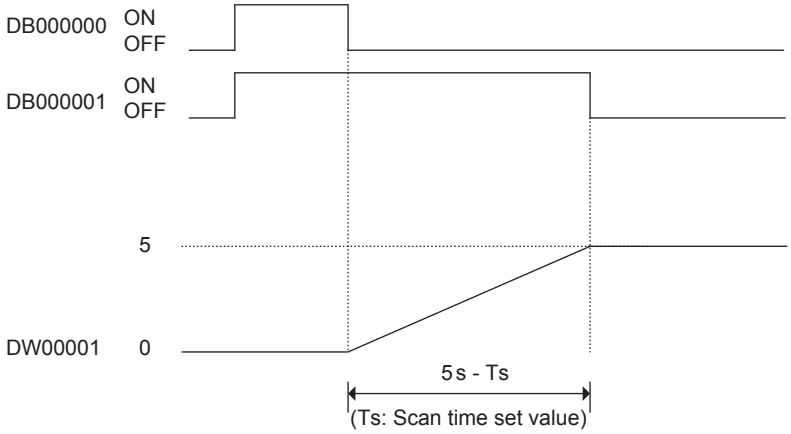
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the set value of the TOFF instruction is 5, and the count value is stored in the DW00001 register.
The DB000001 coil will turn OFF after the DB000000 relay stays OFF for 5 s.



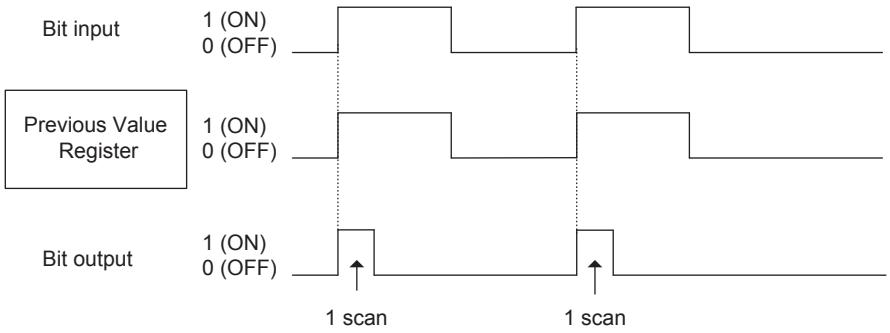

The timing chart is shown below.



## 5.2.7 Rising-edge Pulses (ON-PLS)

### ( 1 ) Operation

The ON-PLS instruction sets the bit output to 1 (ON) for only one scan when the bit input changes from 0 (OFF) to 1 (ON). The previous value of the bit input is saved in the Previous Value Register of the ON-PLS instruction.

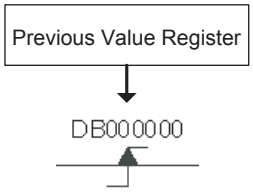



The following table shows the relationship between the bit input of the ON-PLS instruction, the Previous Value Register, and the bit output.

| Bit Input | Previous Value Register | ON-PLS Instruction | Bit Output |
|---|---|---|---|
| 0 (OFF) | 0 (OFF) | → | 0 (OFF) |
| 0 (OFF) | 1 (ON) | → | 0 (OFF) |
| 1 (ON) | 0 (OFF) | → | 1 (ON) |
| 1 (ON) | 1 (ON) | → | 0 (OFF) |

In the third row of the table, notice how the bit input changes from 0 (OFF) in the Previous Value Register to 1 (ON), causing the ON-PLS instruction to set the bit output to 1 (ON).

### ( 2 ) Format




Icon:

Key entry: ]P

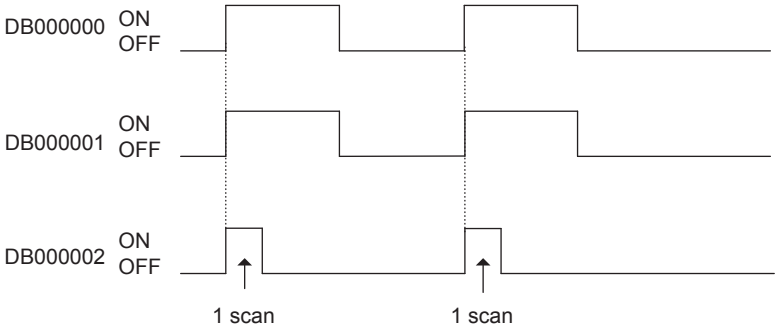
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Previous Value Register | O* | × | × | × | × | × | × |

* C and # registers cannot be used.
- The Previous Value Register holds the previous value of the bit input. Do not use other instructions to set the value of this register.

### ( 3 ) Programming Example

The DB000002 output coil turns ON for only one scan if the status of DB000001 changes when the DB000000 relay changes from OFF to ON.

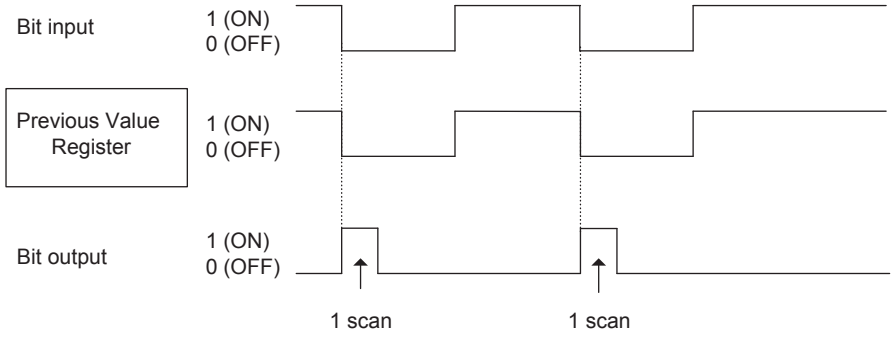
The timing chart is shown below.



# 5.2.8 Falling-edge Pulses (OFF-PLS)

## ( 1 ) Operation

The OFF-PLS instruction sets the bit output to 1 (ON) for only one scan when the bit input changes from 1 (ON) to 0 (OFF). The previous value of the bit input is saved in the Previous Value Register of the OFF-PLS instruction.

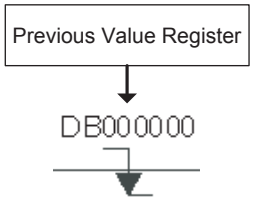



The following table shows the relationship between the bit input of the OFF-PLS instruction, the Previous Value Register, and the bit output.

| Bit Input | Previous Value Register | OFF-PLS Instruction | Bit Output |
|---|---|---|---|
| 0 (OFF) | 0 (OFF) | → | 0 (OFF) |
| 0 (OFF) | 1 (ON) | → | 1 (ON) |
| 1 (ON) | 0 (OFF) | → | 0 (OFF) |
| 1 (ON) | 1 (ON) | → | 0 (OFF) |

In the second row of the table, notice how the bit input changes from 1 (ON) in the Previous Value Register to 0 (OFF), causing the OFF-PLS instruction to set the bit output to 1 (ON).

## ( 2 ) Format




Icon:

Key entry: ]N

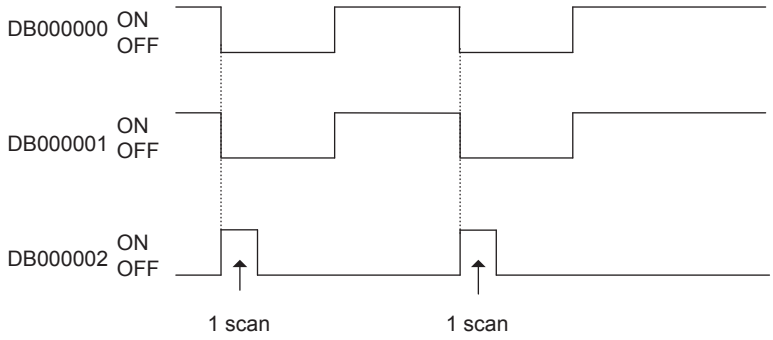
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Previous Value Register | O* | × | × | × | × | × | × |

* C and # registers cannot be used.
- The Previous Value Register holds the previous value of the bit input. Do not use other instructions to set the value of this register.

### ( 3 ) Programming Example

The DB000002 output coil turns ON for only one scan if the status of DB000001 changes when the DB000000 relay changes from ON to OFF.

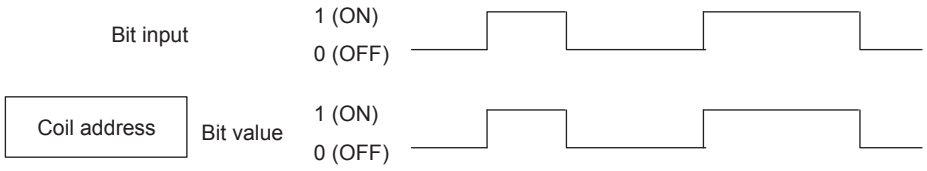
The timing chart is shown below.



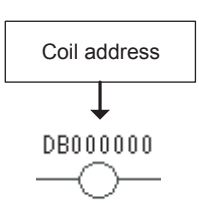
# 5.2.9 Coil (COIL)

## ( 1 ) Operation

The COIL instruction sets the value of the bit at the coil address to 1 (ON) whenever the bit input is 1 (ON). The value of the bit at the coil address is set to 0 (OFF) whenever the bit input is 0 (OFF).




## ( 2 ) Format




Icon:

Key entry: @

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Coil address | O* | × | × | × | × | × | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

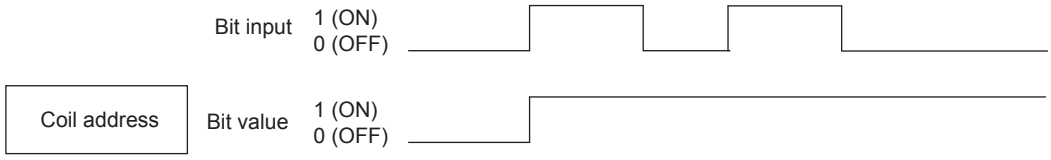
The DB000000 coil turns ON when the DB000001 relay turns ON.

If there are no instructions on the left side, the DB000000 coil is OFF because there is no input.

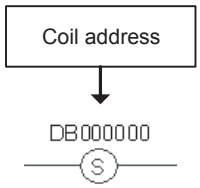
# 5.2.10 Set Coil (S-COIL)

## ( 1 ) Operation

The S-COIL instruction sets the value of the bit at the coil address to 1 (ON) when the bit input is 1 (ON). The set coil stays in the ON state.




## ( 2 ) Format




Icon:

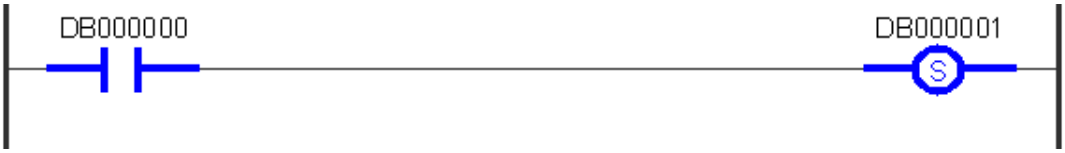
Key entry: @S

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Coil address | O* | × | × | × | × | × | × |

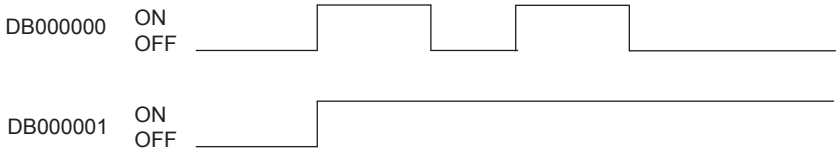
* C and # registers cannot be used.

## ( 3 ) Programming Example

The DB000001 set coil stays in the ON state when the DB000000 relay turns ON.



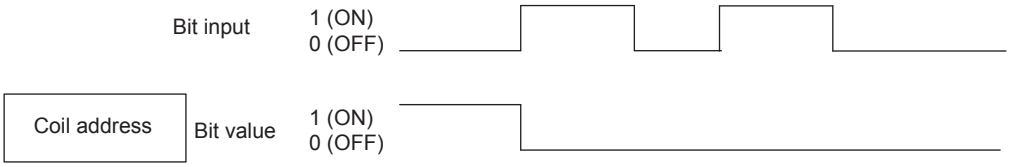

The timing chart is shown below.



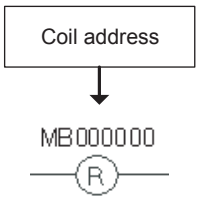
# 5.2.11 Reset Coil (R-COIL)

## ( 1 ) Operation

The R-COIL instruction sets the bit at the reset coil address to 1 (ON) when the bit input is 1 (ON). The set coil is changed to OFF.




## ( 2 ) Format




Icon: Ⓡ

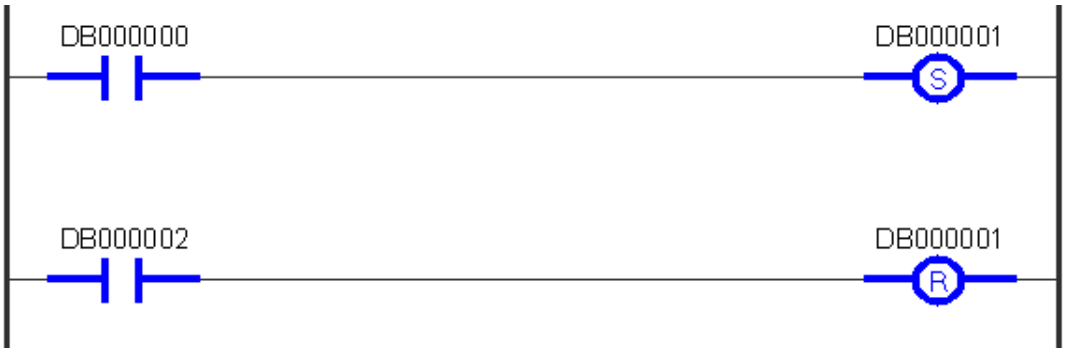
Key entry: @R

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Coil address | O* | × | × | × | × | × | × |

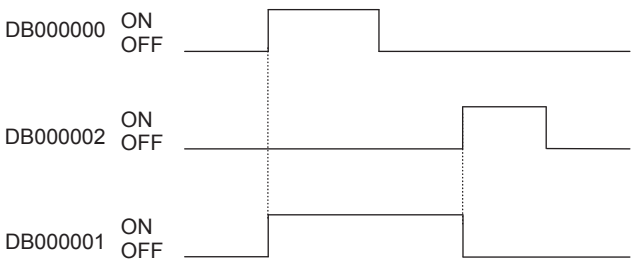
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the reset coil is used to turn OFF the set coil that was turned ON in the first line.
The DB000001 reset coil in the second line turns ON if the DB000002 relay turns ON while the DB000001 set coil is ON, therefore turning OFF the DB000001 set coil.




The timing chart is shown below.

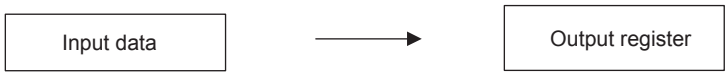


# 5.3 Numeric Operation Instructions

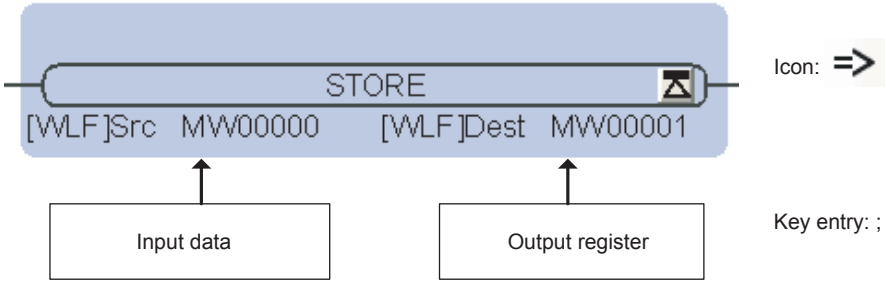
## 5.3.1 Store (STORE)

### ( 1 ) Operation

The input data is stored in the output register.




### ( 2 ) Format



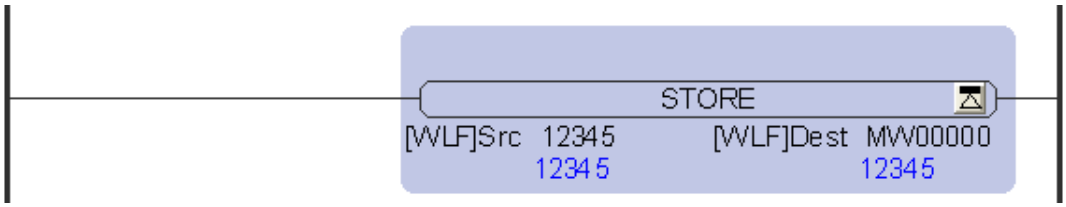

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | ○ | ○ | ○ | ○ |
| Output register (Dest) | × | ○* | ○* | ○* | ○* | ○ | × |

＊ C and # registers cannot be used.

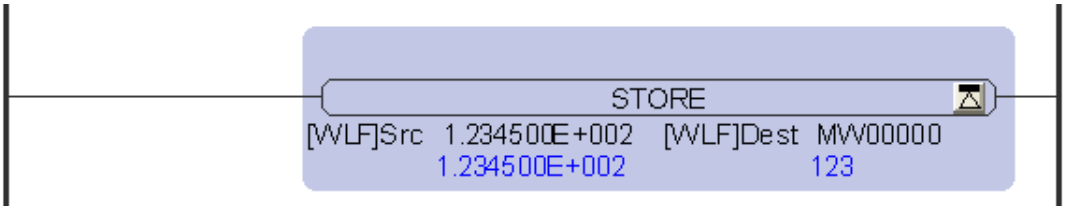
## ( 3 ) Programming Examples

In the following programming examples, the input data is stored in the output register.

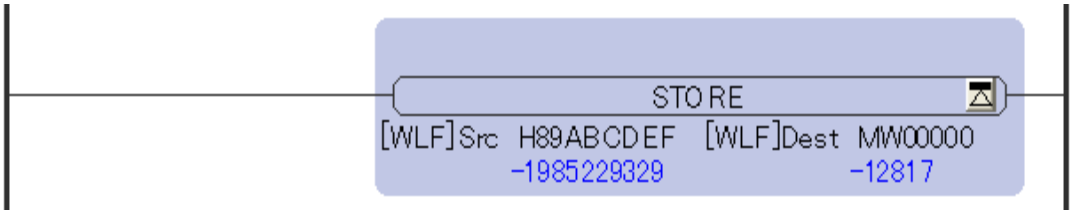
- Storing the Input Data, an Integer Value of 12345, in the MW00000 Output Register



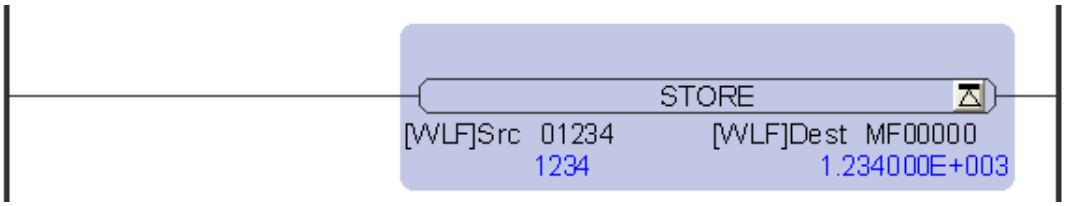

- Storing the Input Data, a Real Value of 123.45, in the MW00000 Output Register




- Storing the Double-length Integer 89ABCDEF Hex in the MW00000 Output Register
  The lower word of the double-length integer –12,817 (CDEF hex) is stored in MW00000.




- Storing the Input Data, an Integer Value of 1234, in the MF00000 Output Register



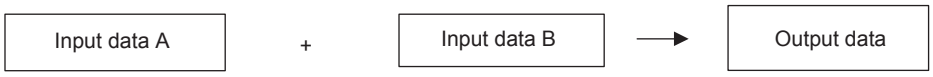

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

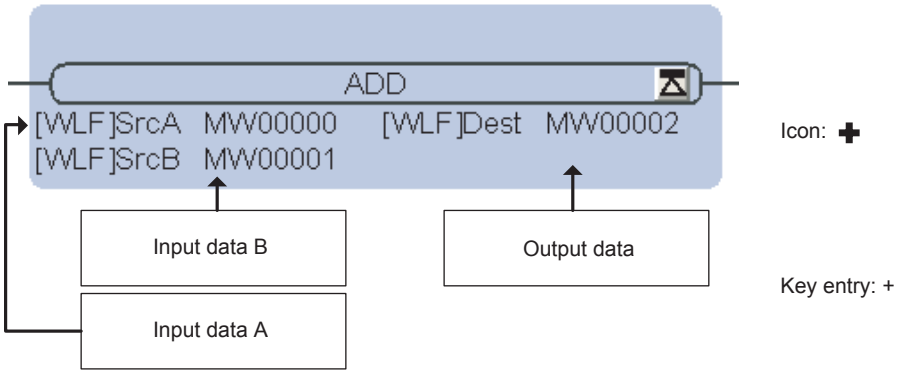
# 5.3.2 Add (ADD (+))

## ( 1 ) Operation

Input data A and input data B are added and the result is stored in the output data.
An operation error occurs if the result produces an overflow or underflow.




## ( 2 ) Format



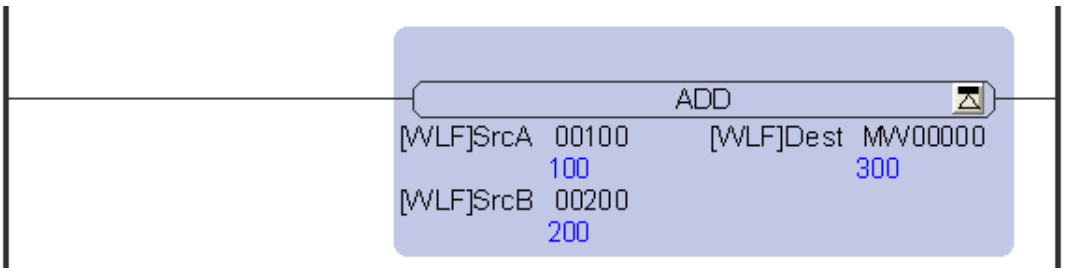

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

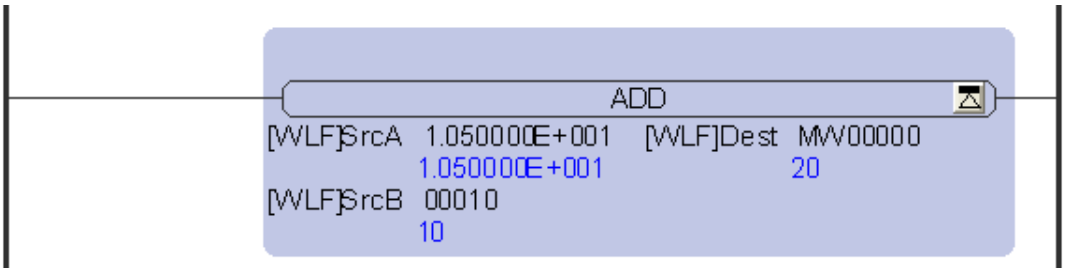
## ( 3 ) Programming Examples

In the following programming examples, input data A and input data B are added and the result is stored in the output data.

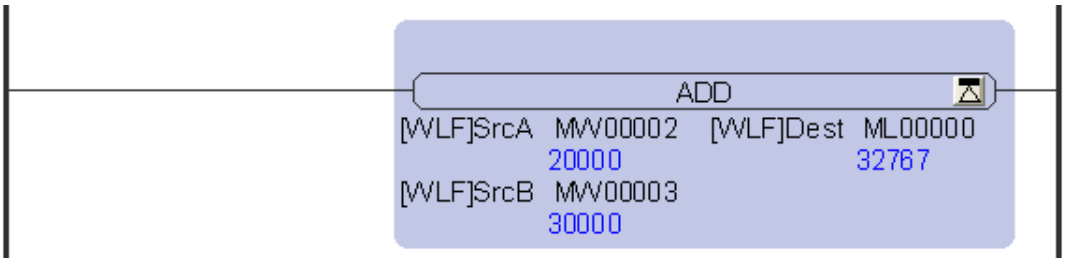
- Storing the Output Data in MW00000 When Input Data A Is 100 and Input Data B Is 200
  100 + 200 → MW00000 = 300




- Storing the Output Data in MW00000 When Input Data A Is 10.5 and Input Data B Is 10
  10.5 + 10 → MW00000 = 20 (when truncating below the decimal point is set)




- Storing the Output Data in ML00000 When Input Data A in MW00002 Is 20,000 and Input Data B in MW00003 Is 30,000
  MW00002 (20,000) + MW00003 (30,000) → ML00000 = 32,767*




* In the example given above, an overflow error occurs because both input data A and B are integers, which limits the result to a number within the range for integers.

## ( 4 ) Additional Information

With integer operations, an overflow operation error occurs if the result exceeds 32,767 and an underflow operation error occurs if the result is less than -32,768.
With double-length integer operations, an overflow operation error occurs if the result exceeds 2,147,483,647 and an underflow operation error occurs if the result is less than -2,147,483,648.

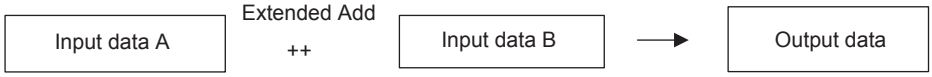
INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

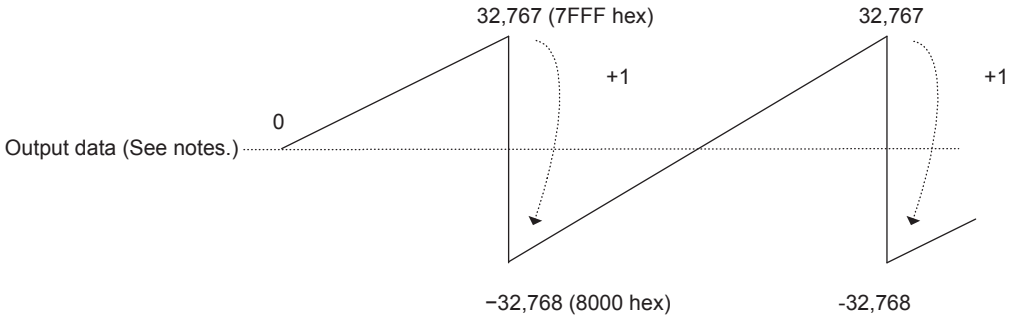
# 5.3.3 Extended Add (ADDX (++))

## ( 1 ) Operation

Input data A and input data B are added and the result is stored in the output data.
Overflows are not treated as operation errors. Operation continues from the maximum value in the negative direction.
Underflows are not treated as operation errors. Operation continues from the maximum value in the positive direction.

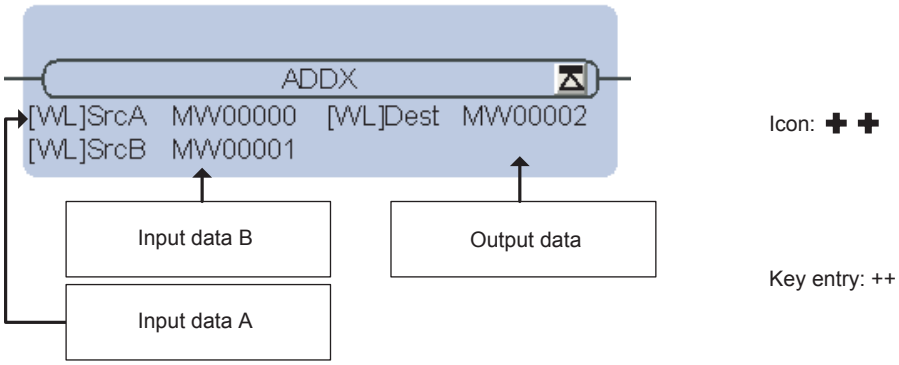



### ■ Output Data Behavior




- In the example shown above, the output data is integer data. With double-length integers, adding 1 to 2,147,483,647 (7FFFFFFF hex) results in -2,147,483,648 (80000000 hex).
- Unlike operations for the ADD, SUB, or EXPRESSION instructions, overflows and underflows do not occur.

## ( 2 ) Format



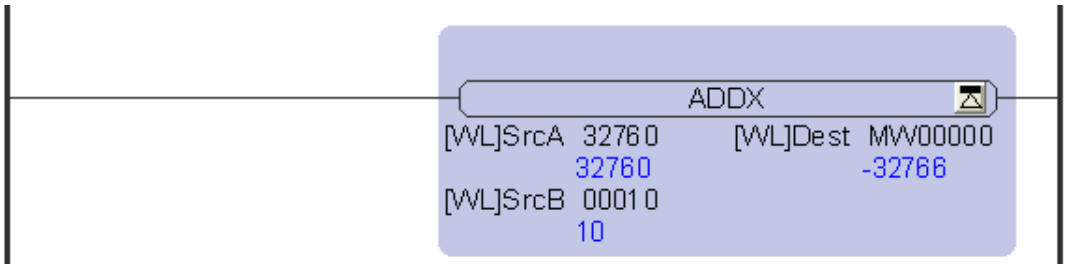

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | × | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | × | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

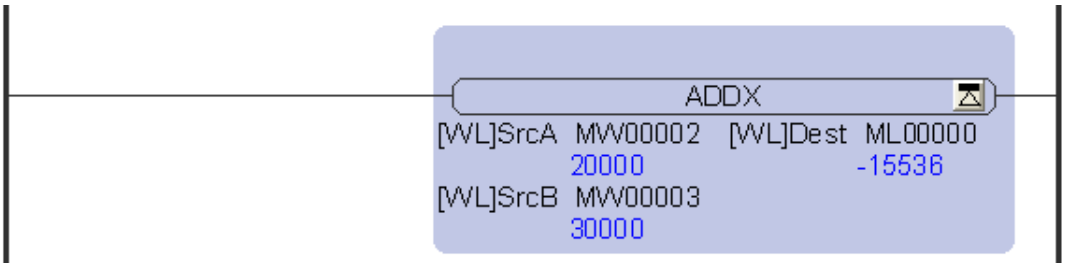
## ( 3 ) Programming Examples

In the following programming examples, input data A and input data B are extended-added and the result is stored in the output data.

- Storing the Output Data in MW00000 When Input Data A Is 32,760 and Input Data B Is 10
  32,760 ++ 10 → MW00000 = -32,766

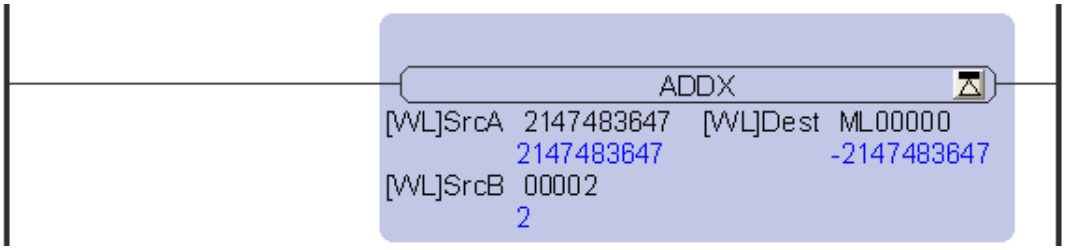



- Storing the Output Data in ML00000 When Input Data A in MW00002 Is 20,000 and Input Data B in MW00003 is 30,000
  20,000 ++ 30,000 → ML00000 = -15,536*

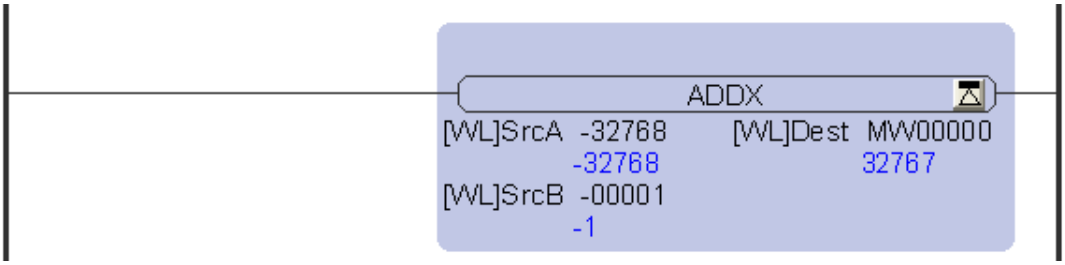



* In the example given above, ML00000 does not equal 50,000 because both input data A and B are integers, which limits the result to a number within the range for integers.

- Storing the Output Data in ML00000 When Input Data A Is 2,147,483,647 and Input Data B Is 2
  2,147,483,647 ++ 2 → ML00000 = -241,783,647




- Storing the Output Data in MW00000 When Input Data A Is -32,768 and Input Data B Is -1
  -32,768 ++ -1 → MW00000 = 32,767



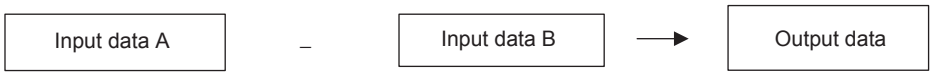

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

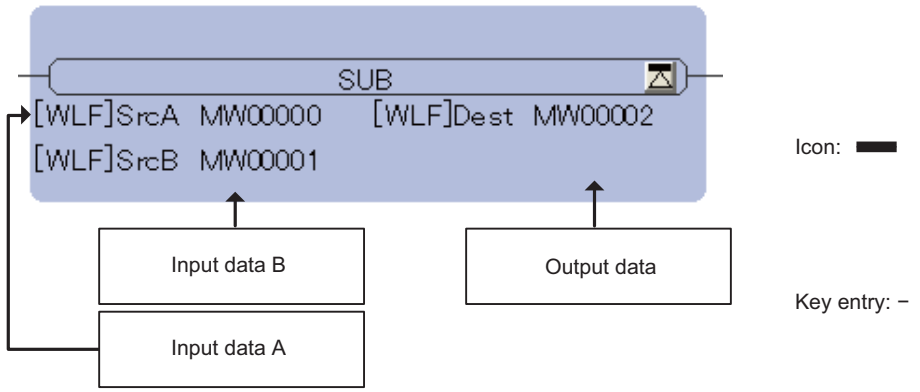
# 5.3.4 Subtract (SUB (–))

## ( 1 ) Operation

Input data B is subtracted from input data A and the result is stored in the output data.
An operation error occurs if the result produces an overflow or underflow.




## ( 2 ) Format



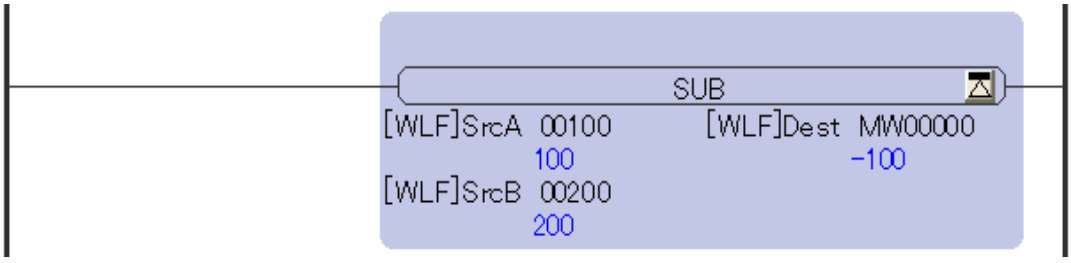

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

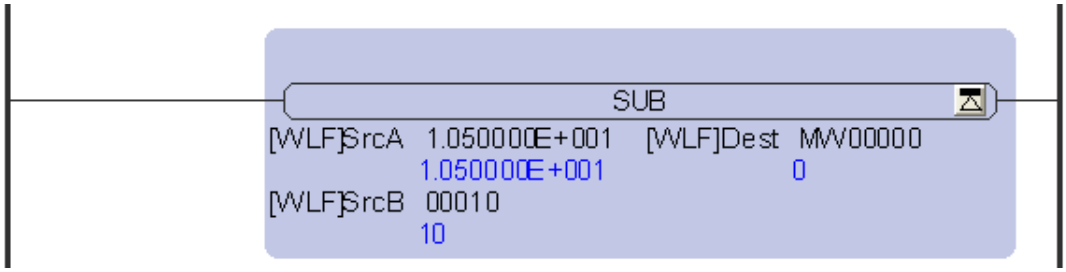
## ( 3 ) Programming Examples

In the following programming examples, input data B is subtracted from input data A and the result is stored in the output data.

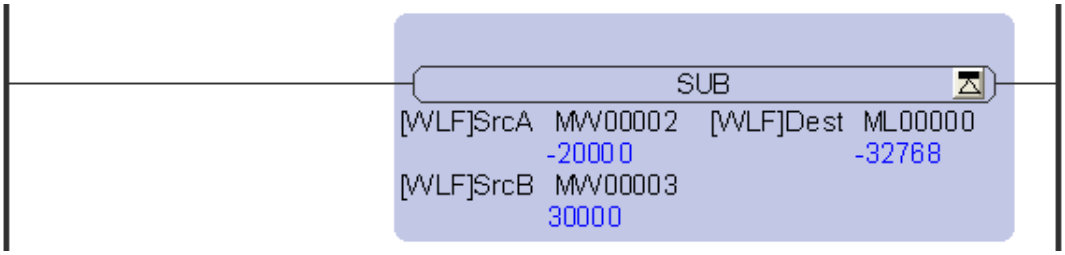
- Storing the Output Data in MW00000 When Input Data A Is 100 and Input Data B Is 200
  100 – 200 → MW00000 = -100



- Storing the Output Data in MW00000 When Input Data A Is 10.5 and Input Data B Is 10
  10.5 – 10 → MW00000 = 0 (when truncating below the decimal point is set)




- Storing the Output Data in ML00000 When Input Data A in MW00002 Is -20,000 and Input Data B in MW00003 Is 30,000
  -20,000 – 30,000 → ML00000 = -32,768*




* In the example given above, an underflow error occurs because both input data A and B are integers, which limits the result to a number within the range for integers.

## ( 4 ) Additional Information

With integer operations, an overflow operation error occurs if the result exceeds 32,767 and an underflow operation error occurs if the result is less than -32,768.
With double-length integer operations, an overflow operation error occurs if the result exceeds 2,147,483,647 and an underflow operation error occurs if the result is less than -2,147,483,648.

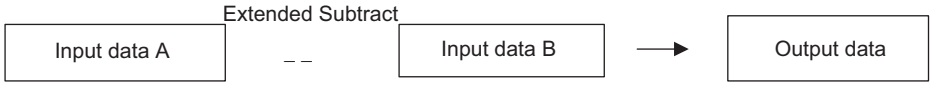
INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

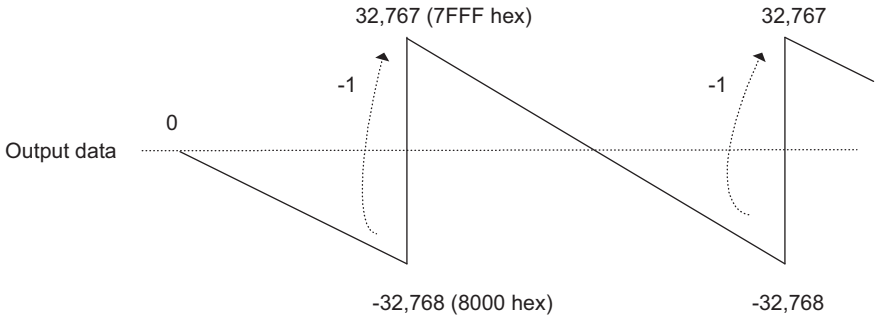
# 5.3.5 Extended Subtract (SUBX (– –))

## ( 1 ) Operation

Input data B is subtracted from input data A and the result is stored in the output data.
Overflows are not treated as operation errors. Operation continues from the maximum value in the negative direction.
Underflows are not treated as operation errors. Operation continues from the maximum value in the positive direction.

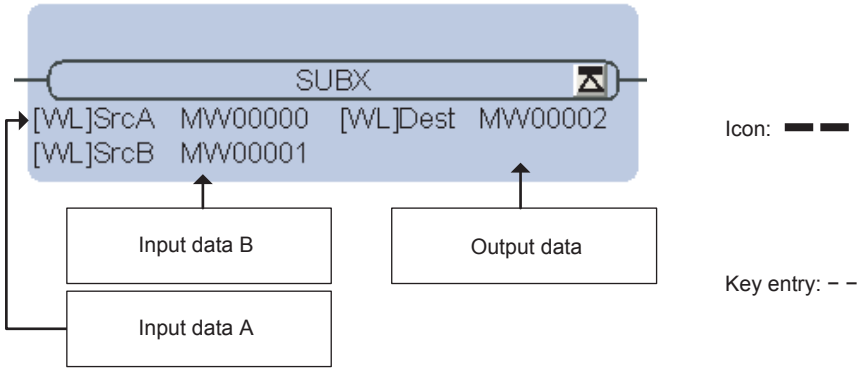



### ■ Output Data Behavior




- In the example shown above, the output data is integer data. With double-length integers, subtracting 1 from -2,147,483,647 (80000000 hex) results in 2,147,483,647(7FFFFFFF hex).
- Unlike operations for the ADD, SUB, or EXPRESSION instructions, overflows and underflows do not occur.

## ( 2 ) Format



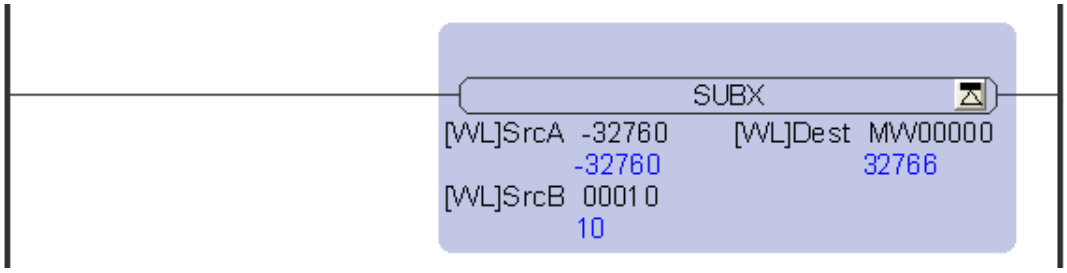

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | × | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | × | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

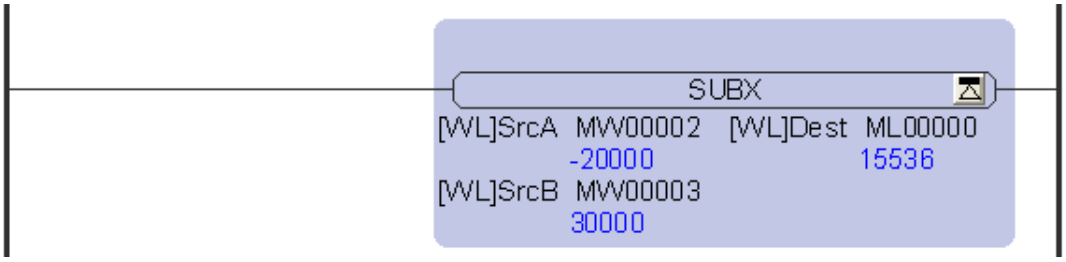
## ( 3 ) Programming Examples

In the following programming examples, input data B is extended-subtracted from input data A and the result is stored in the output data.

- Storing the Output Data in MW00000 When Input Data A Is -32,760 and Input Data B Is 10
  -32,768 – – 10 → MW00000 = 32,766

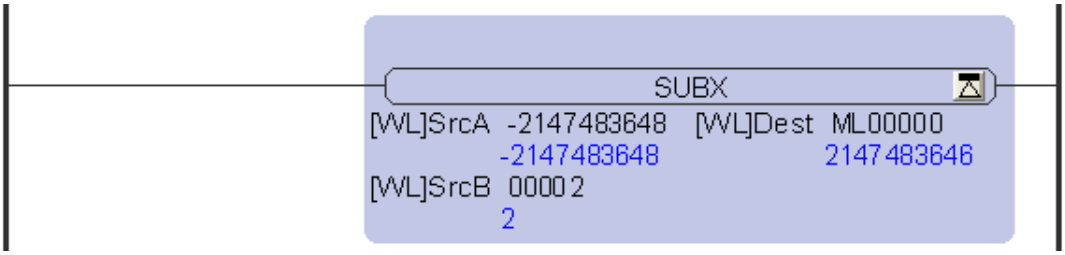



- Storing the Output Data in ML00000 When Input Data A in MW00002 Is -20,000 and Input Data B in MW00003 Is 30,000
  -20,000 – –30,000 → ML00000 = 15,536*

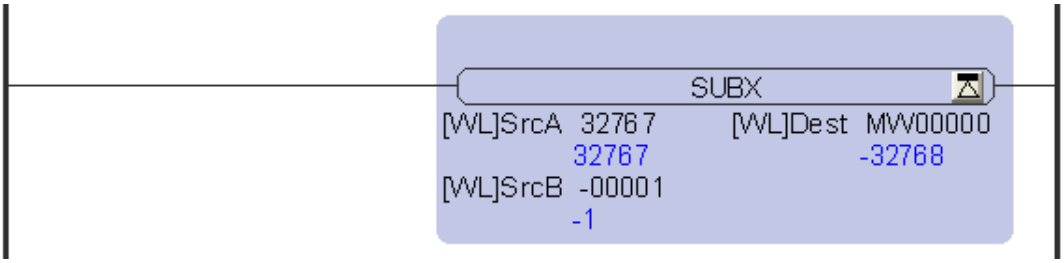



  * In the example given above, ML00000 does not equal -50,000 because both input data A and B are integers, which limits the result to a number within the range for integers.

- Storing the Output Data in ML00000 When Input Data A Is -2,147,483,648 and Input Data B Is 2
  -2,147,483,648 – – 2 → ML00000 = 241,783,646




- Storing the Output Data in MW00000 When Input Data A Is 32,767 and Input Data B Is -1
  32,767 – – -1 → MW00000 = -32,768



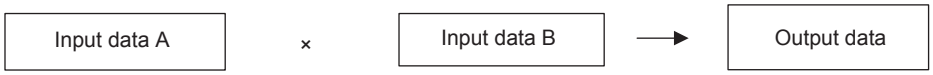

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

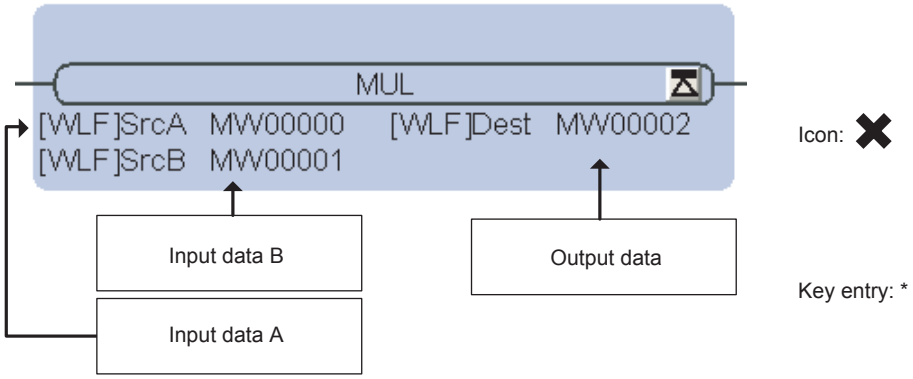
# 5.3.6 Multiply (MUL (x))

## ( 1 ) Operation

Input data A and input data B are multiplied and the result is stored in the output data.




## ( 2 ) Format



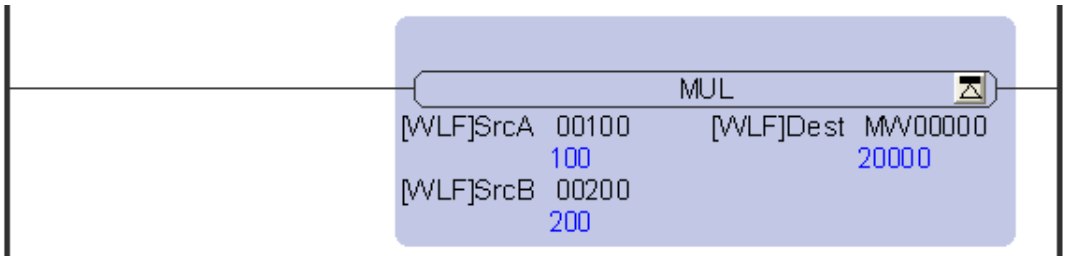

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

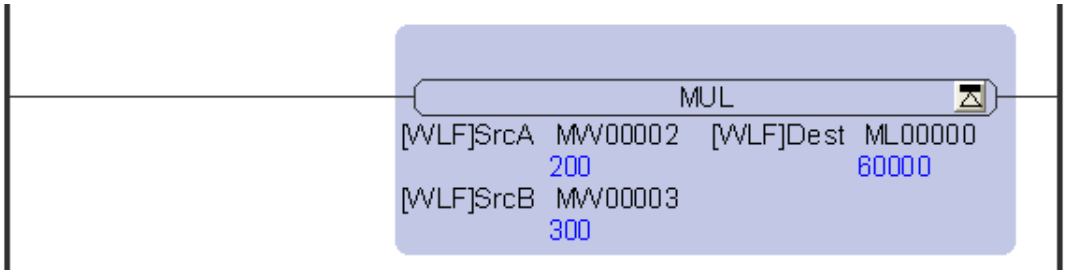
## ( 3 ) Programming Examples

In the following programming examples, input data A and input data B are multiplied and the result is stored in the output data.

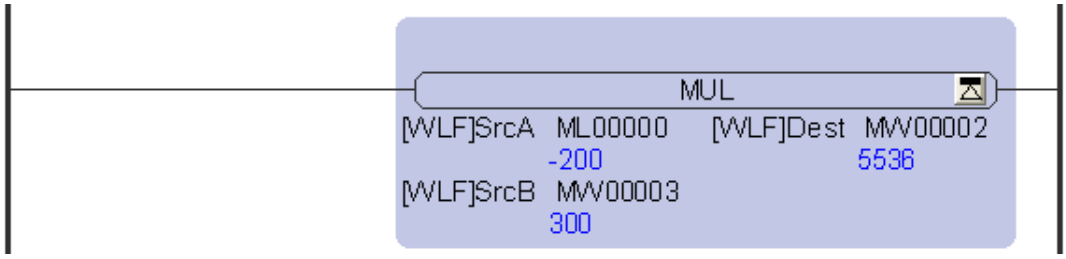
- Storing the Output Data in MW00000 When Input Data A Is 100 and Input Data B Is 200
  100 × 200 → MW00000 = 20,000



- Storing the Output Data in ML00000 When Input Data A in MW00002 Is 200 and Input Data B in MW00003 Is 300
  200 × 300 → ML00000 = 60,000




- Storing the Output Data in MW00002 When Input Data A in ML00000 Is -200 and Input Data B in MW00003 Is 300
  -200 × 300 → MW00002 = 5,536*




* The input data contains a double-length integer, so this operation is performed as a double-length integer operation. However, the output data is integer data, so if the operation result exceeds the range for integers, the lower 16-bits of the original operation result is stored in the output data.

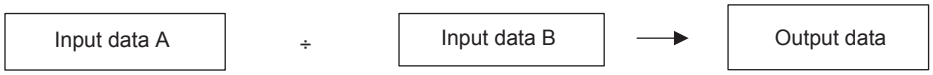
INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

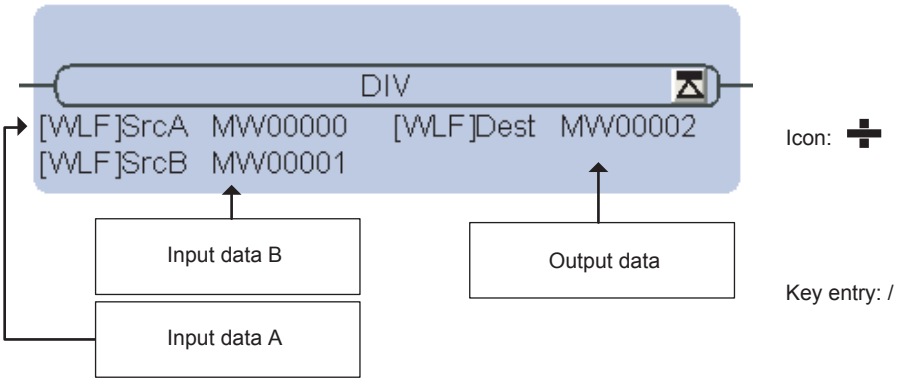
# 5.3.7 Divide (DIV (÷))

## ( 1 ) Operation

Input data A is divided by input data B and the result is stored in the output data.




## ( 2 ) Format



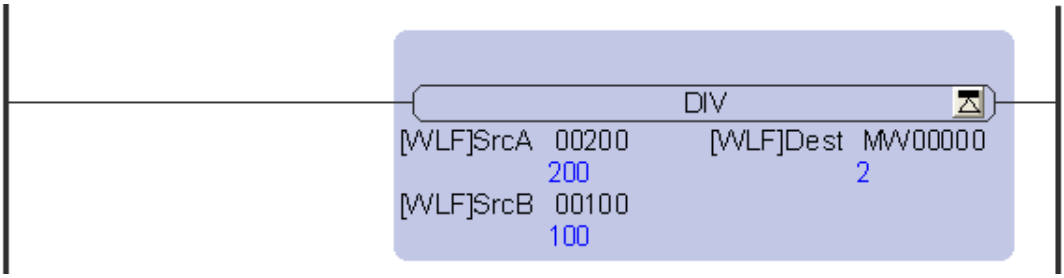

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

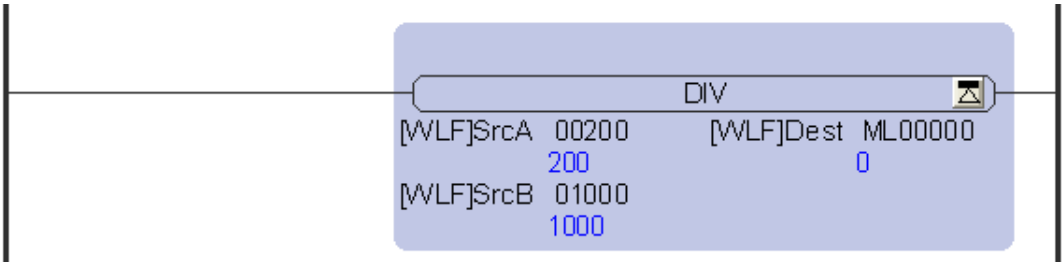
## ( 3 ) Programming Examples

In the following programming examples, input data A is divided by input data B and the result is stored in the output data.

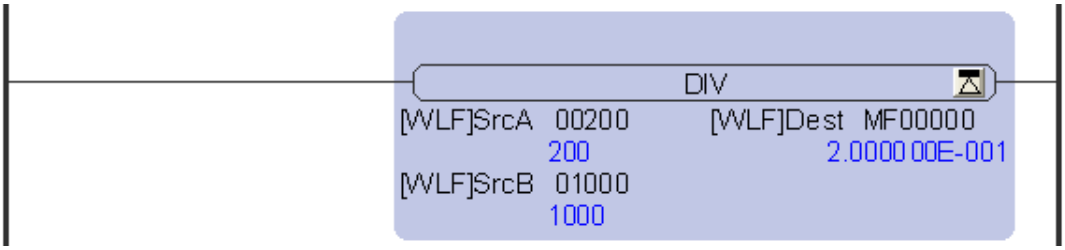
- Storing the Output Data in MW00000 When Input Data A Is 200 and Input Data B Is 100
  200 ÷ 100 → MW00000 = 2




- Storing the Output Data in ML00000 When Input Data A Is 200 and Input Data B Is 1,000
  200 ÷ 1,000 → ML00000 = 0




- Storing the Output Data in MF00000 When Input Data A Is 200 and Input Data B Is 1,000
  200 ÷ 1,000 → MF00000 = 0.2



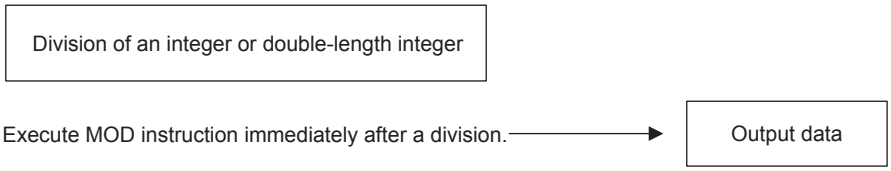

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.
Normally, addition and subtraction instructions (+, –, ++, and – –) involving double-length integers are performed as 32-bit operations.
However, these instructions are performed as 64-bit operations if they are used to correct the remainder produced by an immediately preceding MUL instruction (×) and are immediately followed by a DIV instruction (÷).

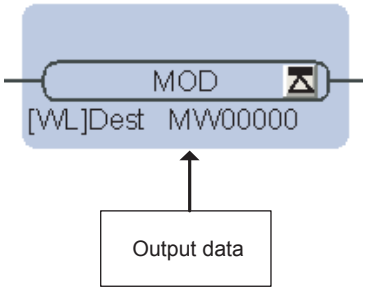
# 5.3.8 Integer Remainder (MOD)

## ( 1 ) Operation

The remainder of the immediately preceding integer or double-length integer division is stored in the output data. The MOD instruction must be executed immediately after the DIV instruction. If the MOD instruction is executed at any other time, the operation result obtained before the next numeric operation instruction will be invalid.




## ( 2 ) Format




Icon: MOD

Key entry: MOD

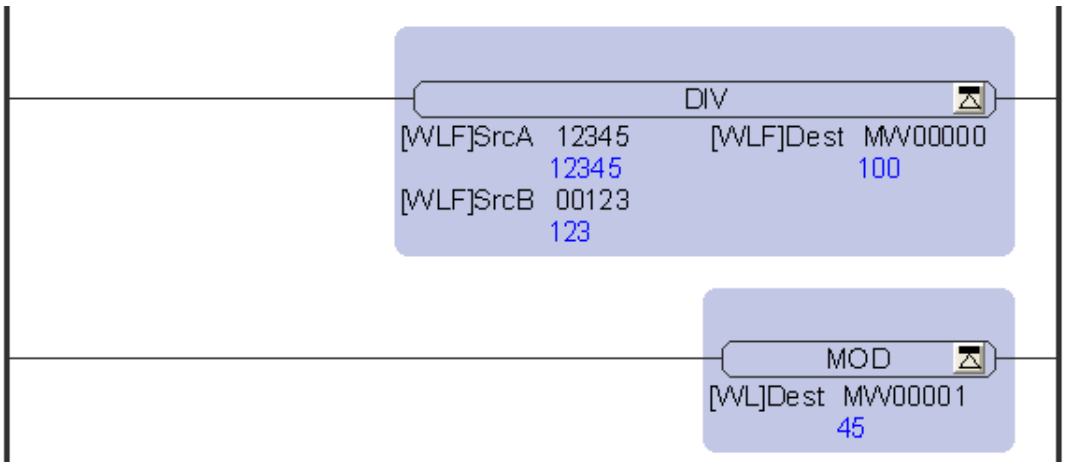
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

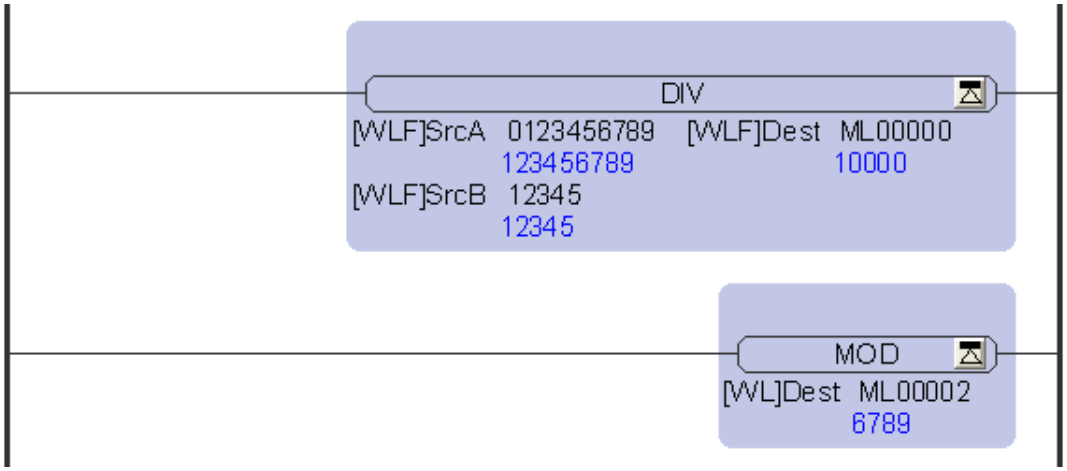
## ( 3 ) Programming Examples

In the following programming examples, input data A is divided by input data B and the remainder is stored in the output data.

- If the Immediately Preceding Division Is as Follows: 12,345 ÷ 123 → MW00000 = 100
  And then the MOD instruction is executed immediately afterward → MW00001 = 45.




- If the Immediately Preceding Division Is as Follows: 123,456,789 ÷ 12,345 → ML00000 = 10,000
  And then the MOD instruction is executed immediately afterward → ML00002 = 6,789




INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

# 5.3.9 Real Remainder (REM)

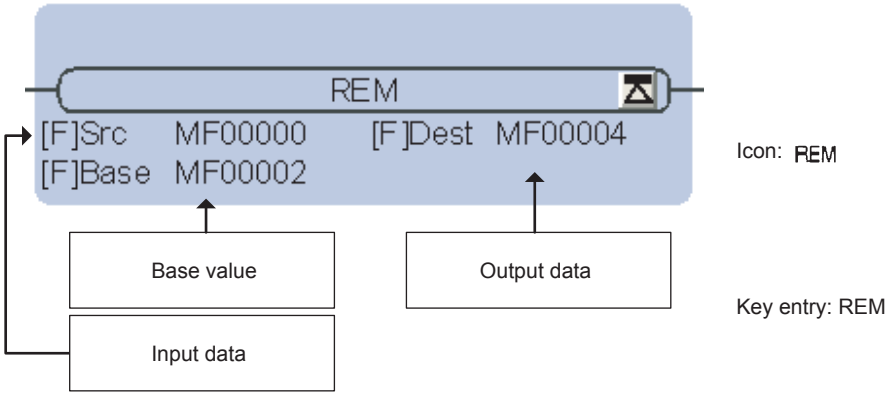
## ( 1 ) Operation

The remainder from a real number division is stored in the output data. Here, the remainder refers to the remainder obtained by repeatedly subtracting the base value from the input data.
Specifically, the value obtained by subtracting the base value from the input data n number of times (input data - base value × n) is output when it becomes less than the base value.

### ■ Condition n

The output data is computed by using the first value of n that satisfies the following formula when the value of n is incremented from 0, 1, 2, 3, etc.
(Input data – Base value × n) < Base value

## ( 2 ) Format



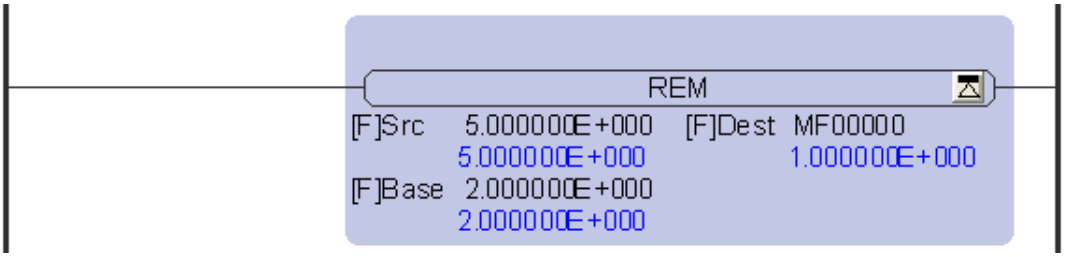

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | ○ | × | × | ○ |
| Base (Base) | × | × | × | ○ | × | × | ○ |
| Output data (Dest) | × | × | × | ○* | × | ○ | × |

＊ C and # registers cannot be used.

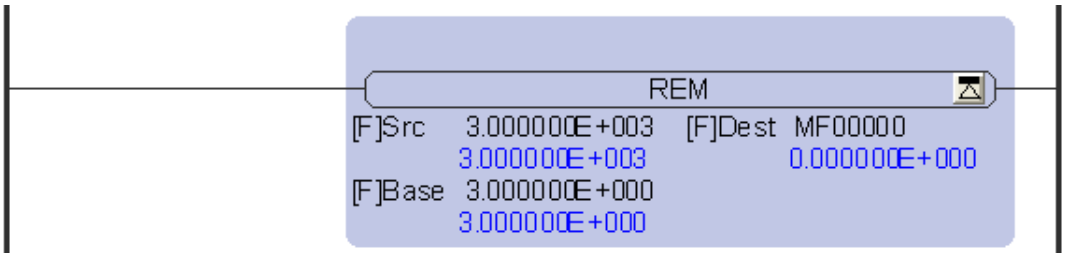
## ( 3 ) Programming Examples

In the following programming examples, the base value is subtracted from the input data n times and the remainder is stored in the output data.

- Storing the Output Data in MF00000 When the Input Data Is 5.0 and the Base Value Is 2.0
  5.0 – 2.0 – 2.0 = 1.0 < Base (2.0) → MF00000 = 1.0




- Storing the Output Data in MF00000 When the Input Data Is 3000.0 and the Base Value Is 3.0
  3,000.0 – 3.0 – 3.0 ... = 0.0 < Base (3.0) → MF00000 = 0.0

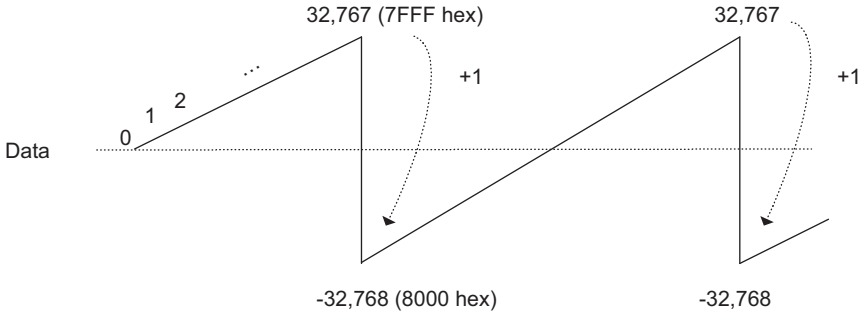


# 5.3.10 Increment (INC)

## ( 1 ) Operation

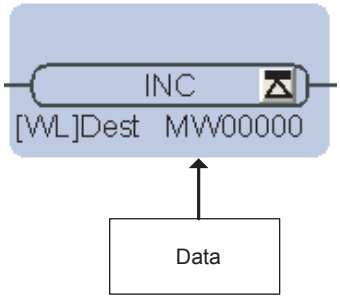
A value of 1 is added to the integer or double-length integer data. No overflow or underflow will occur for either an integer or double-length integer. This operation handles overflows and underflows in the same way as the ADDX instruction.

### ■ Output Data Behavior




- In the example shown above, the data is an integer. With double-length integers, adding 1 to 2,147,483,647 (7FFFFFFF hex) results in -2,147,483,648(80000000 hex).

## ( 2 ) Format




Icon: **+1**

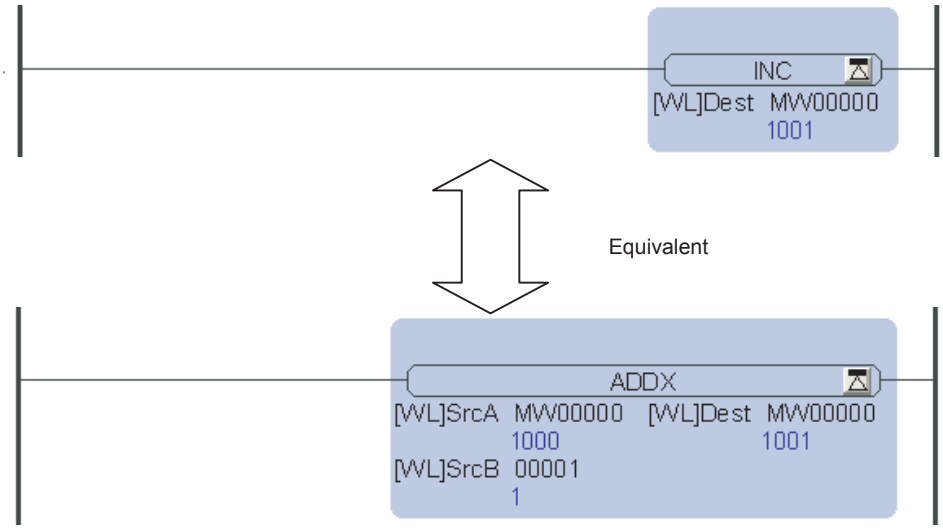
Key entry: INC

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Examples

The following programming examples achieve the same result by using the INC instruction and by using the ADDX instruction.
The INC instruction is equivalent to adding 1 to the data 1,000 in MW00000 using the ADDX instruction.

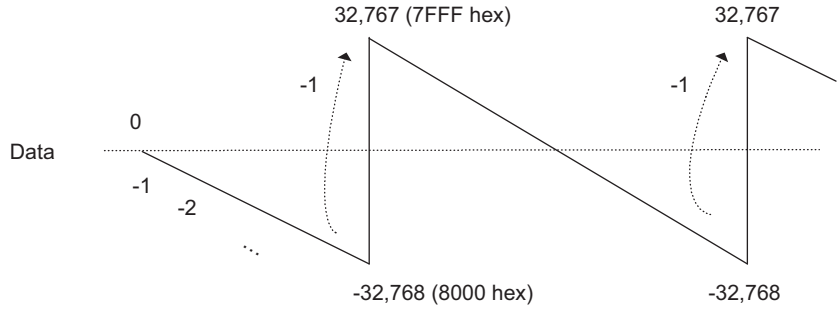


# 5.3.11 Decrement (DEC)

## ( 1 ) Operation

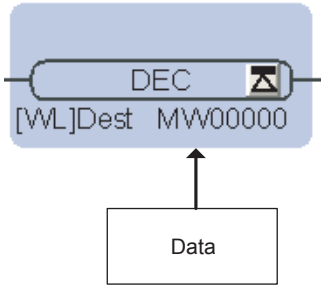
A value of 1 is subtracted from the integer or double-length integer data. No overflow or underflow will occur for either an integer or double-length integer. This operation handles overflows and underflows in the same way as the SUBX instruction.

### ■ Output Data Behavior




- In the example shown above, the data is an integer. With double-length integers, subtracting 1 from -2,147,483,648 (80000000 hex) results in 2,147,483,647(7FFFFFFF hex).

## ( 2 ) Format




Icon: **-1**

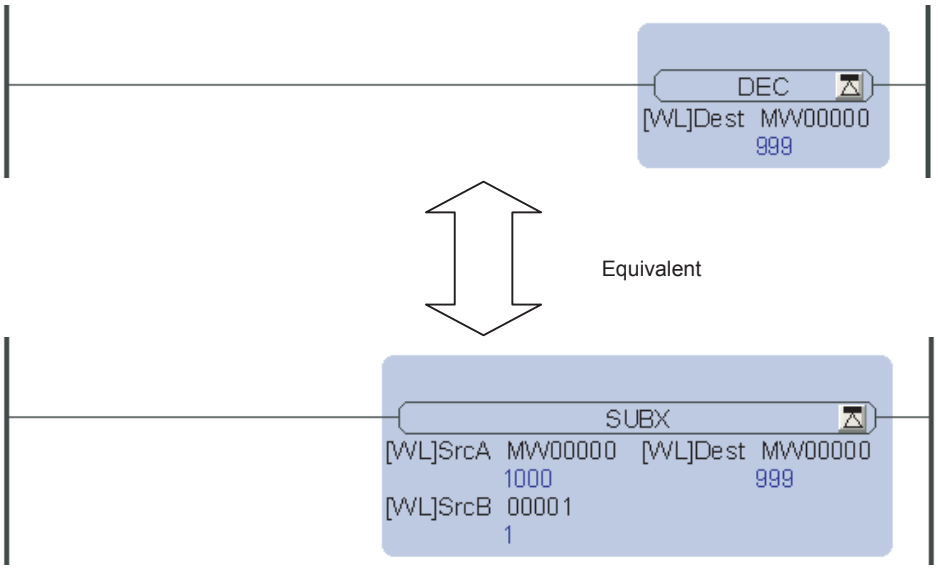
Key entry: DEC

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Data (Dest) | × | O* | O* | × | × | O | × |

* C and # registers cannot be used.

## ( 3 ) Programming Examples

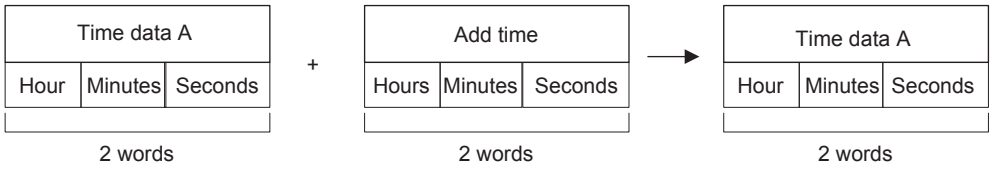
The following programming examples achieve the same result by using the DEC instruction and by using the SUBX instruction.
The DEC instruction is equivalent to subtracting 1 from the data 1,000 in MW00000 using the SUBX instruction.



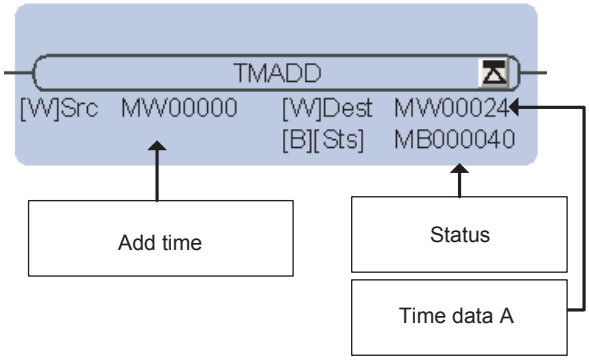
# 5.3.12 Add Time (TMADD)

## ( 1 ) Operation

A duration (hours/minutes/seconds) is added to a time (hour/minutes/seconds). The add time is added to time data A and the result is stored in time data A. Time data is two words long.




## ( 2 ) Format




Icon:

Key entry: TMADD

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Add time (Src) | × | ○*2 | × | × | × | × | × |
| Time data A (Dest) | × | ○*2 | × | × | × | × | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

The time data is formatted as shown below.

| Offset | Contents | Data Range (BCD) |
|---|---|---|
| 0 | Hour/minutes | Upper byte (hour): 00 to 23<br>Lower byte (minutes): 00 to 59 |
| 1 | Seconds | 0000 to 0059 |

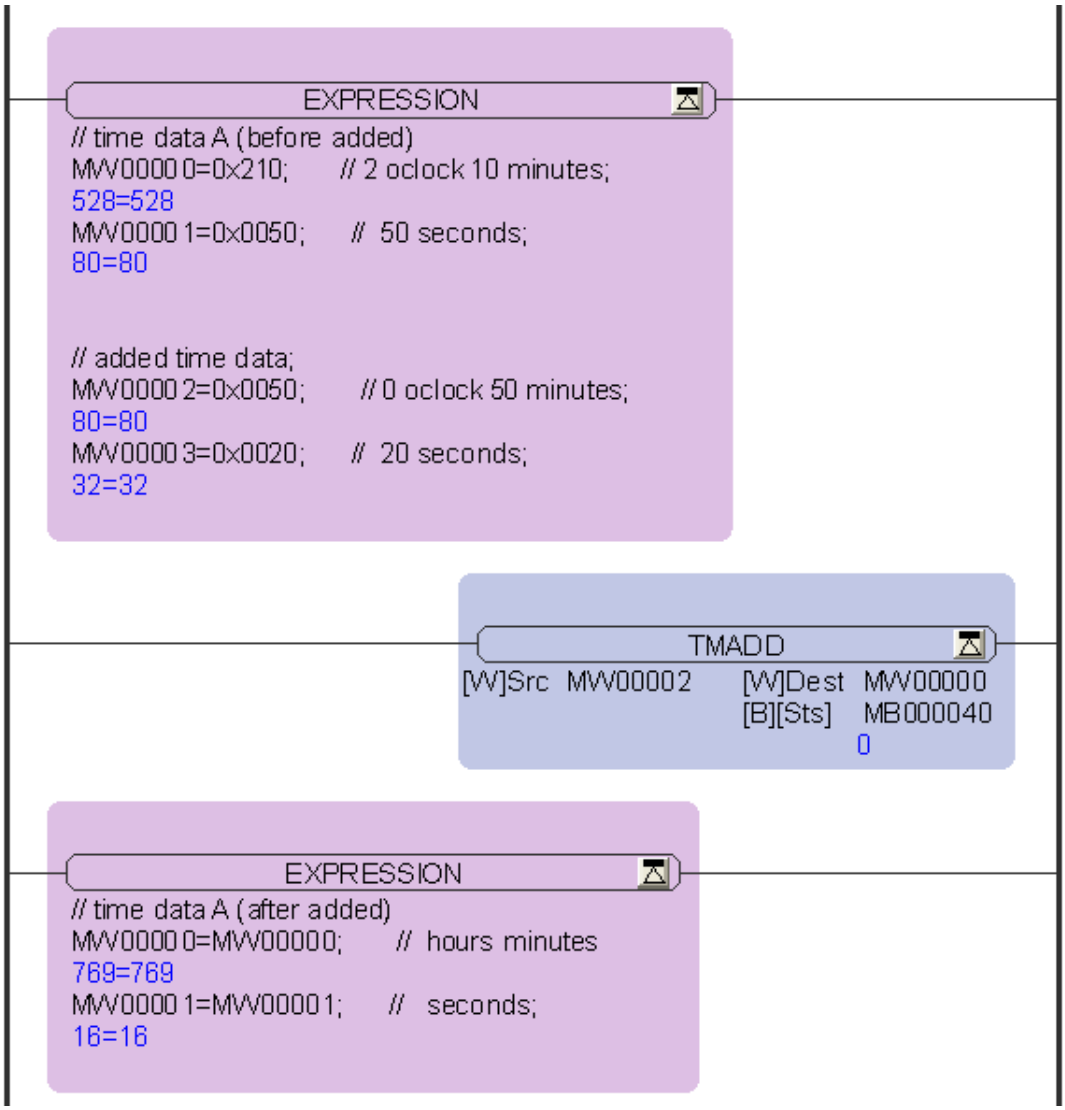
If the operation result exceeds any of the data ranges given above, time data A is not updated, the seconds data is set to 9,999, and the status bit is set to 1 (ON).
If the operation result is within the ranges, the status bit is set to 0 (OFF).

## ( 3 ) Programming Example

The following table gives typical conditions for creating ladder programming that uses the TMADD instruction. The examples show time data A before instruction execution, and the add time.

| Time | Time Data A before Execution of Instruction | Add Time |
|---|---|---|
| Hour/minutes | MW00000 = 0210 hex<br>(2:10) | MW00002 = 0050 hex<br>(0 hours 50 minutes) |
| Seconds | MW00001 = 0050 hex<br>(50 seconds) | MW00003 = 0020 hex<br>(20 seconds) |

In the following programming example, the time data is added to the time under the above conditions and the resulting time data is stored.



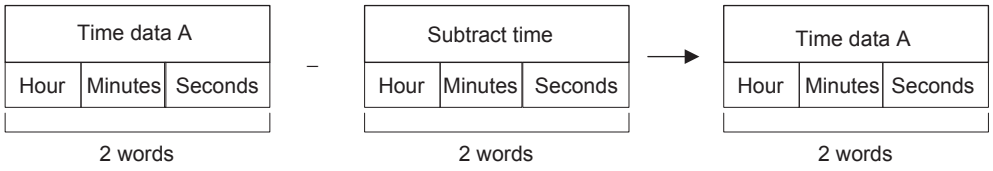

The result of adding the add time to the value of time data A before instruction execution is shown below.

| Time | Time Data A after Execution of Instruction |
|---|---|
| Hour/minutes | MW00000 = 769 = 0301 hex<br>(3:01) |
| Seconds | MW00001 = 16 = 0010 hex<br>(10 seconds) |

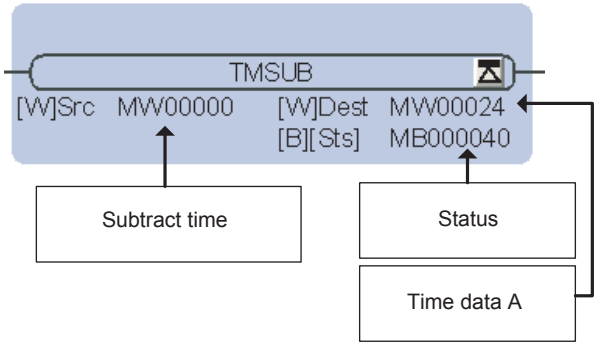
# 5.3.13 Subtract Time (TMSUB)

## ( 1 ) Operation

A duration (hours/minutes/seconds) is subtracted from a time (hour/minutes/seconds). The subtract time is subtracted from time data A and the result is stored in time data A. Time data is two words long.




## ( 2 ) Format




Icon:

Key entry: TMSUB

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Subtract time (Src) | × | ○*2 | × | × | × | × | × |
| Time data A (Dest) | × | ○*2 | × | × | × | × | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

The time data is formatted as shown below.

| Offset | Contents | Data Range (BCD) |
|---|---|---|
| 0 | Hour/minutes | Upper byte (hour): 00 to 23<br>Lower byte (minutes): 00 to 59 |
| 1 | Seconds | 0000 to 0059 |

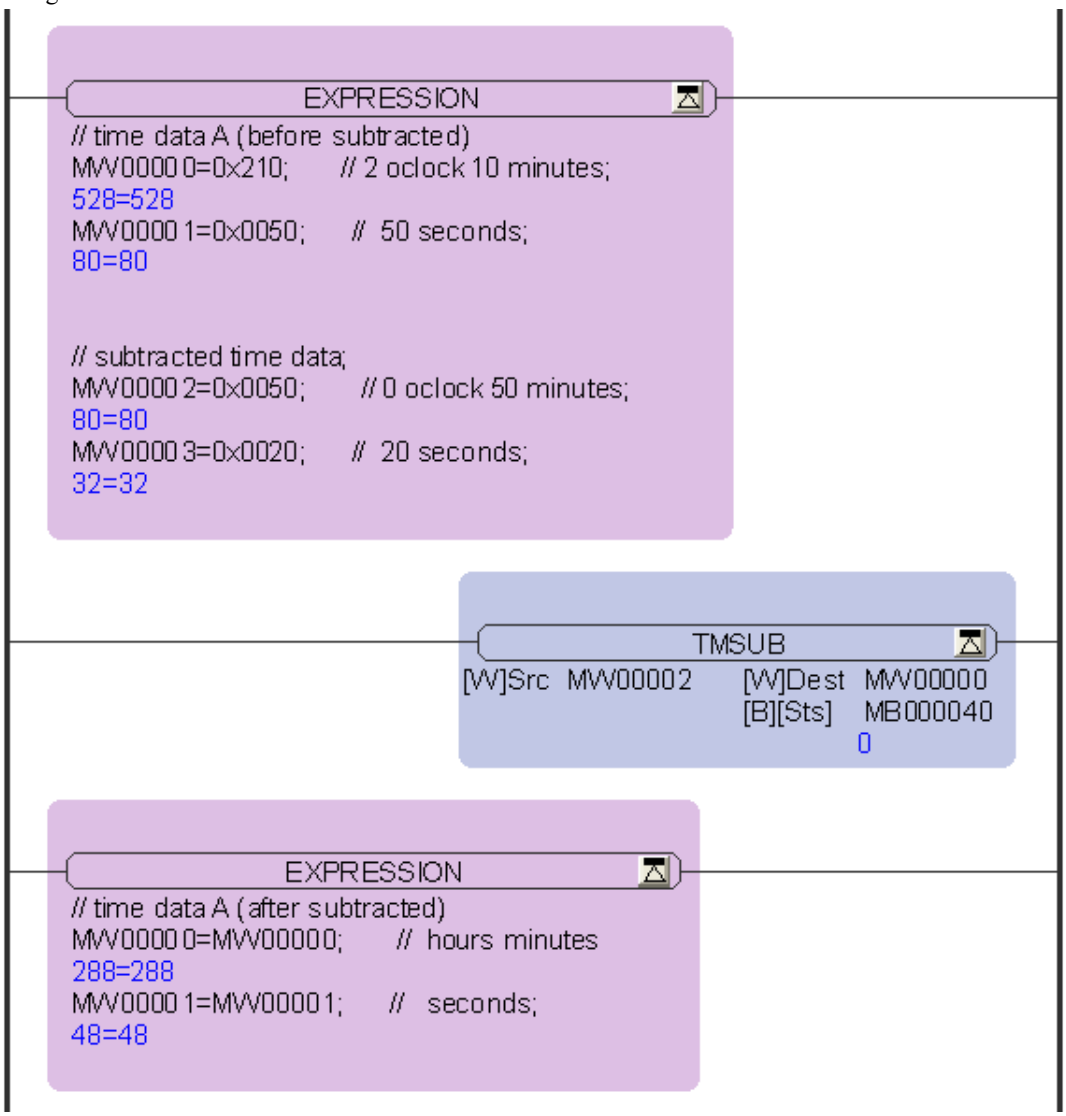
If the operation result exceeds any of the data ranges given above, time data A is not updated, the seconds data is set to 9,999, and the status bit is set to 1 (ON).
If the operation result is within the ranges, the status bit is set to 0 (OFF).

## ( 3 ) Programming Example

The following table gives typical conditions for creating ladder programming that uses the TMSUB instruction. The examples show time data A before instruction execution, and the subtract time.

| Time | Time Data A before Execution of Instruction | Subtract Time |
|---|---|---|
| Hour/minutes | MW00000 = 0210 hex<br>(2:10) | MW00002 = 0050 hex<br>(0 hours 50 minutes) |
| Seconds | MW00001 = 0050 hex<br>(50 seconds) | MW00003 = 0020 hex<br>(20 seconds) |

In the following programming example, the subtract time is subtracted from the time under the above conditions and the resulting time data is stored.



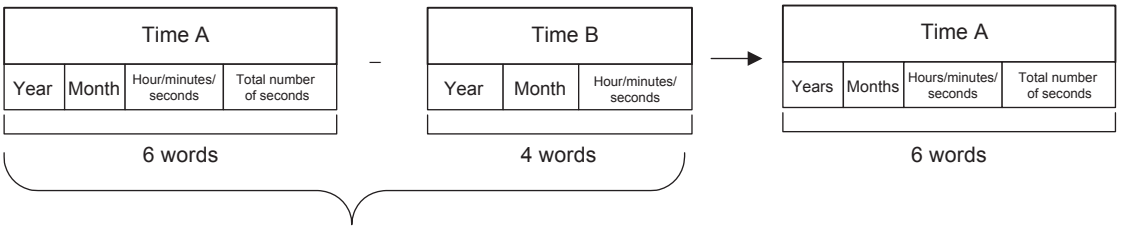

The result of subtracting the subtract time from the value of time data A before instruction execution is shown below.

| Time | Time Data A after Execution of Instruction |
|---|---|
| Hour/minutes | MW00000 = 288 = 0120 hex<br>(1:20) |
| Seconds | MW00001 = 48 = 0030 hex<br>(30 seconds) |

# 5.3.14 Spend Time (SPEND)

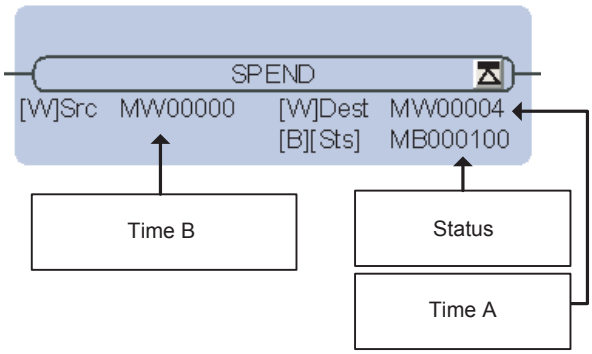
## ( 1 ) Operation

The elapsed time is calculated by subtracting two data items (year/month/day/hour/minutes/seconds). The instruction subtracts time B from time A, which gives the time elapsed from time B to time A and the result is stored in time A. Time data is four words long.




The time elapsed from time B to time A is calculated.

## ( 2 ) Format




Icon:

Key entry: SPEND

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Time B (Src) | × | ○*2 | × | × | × | × | × |
| Time A (Dest) | × | ○*2 | × | × | × | × | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

Time data B is formatted as shown below.

| Offset | Contents | Data Range (BCD) | I/O |
|---|---|---|---|
| 0 | Year (BCD) | 0000 to 0099 | IN |
| 1 | Month/day (BCD) | Upper byte (month): 01 to 12<br>Lower byte (day): 01 to 31 | IN |
| 2 | Hour/minutes (BCD) | Upper byte (hour): 00 to 23<br>Lower byte (minutes): 00 to 59 | IN |
| 3 | Seconds (BCD) | 0000 to 0059 | IN |

Time data A is formatted as shown below.

| Offset | Contents | Data Range (BCD) | I/O |
|---|---|---|---|
| 0 | Year (BCD) | 0000 to 0099 | IN/OUT |
| 1 | Month/day (BCD) | Upper byte (month): 01 to 12<br>Lower byte (day): 01 to 31 | IN/OUT |
| 2 | Hour/minutes (BCD) | Upper byte (hour): 00 to 23<br>Lower byte (minutes): 00 to 59 | IN/OUT |
| 3 | Seconds (BCD) | 0000 to 0059 | IN/OUT |
| 4 | Total number of seconds | Operation result of years, months, days, hours, minutes, and seconds converted into seconds (double-length integer) | IN/OUT |
| 5 | | | |

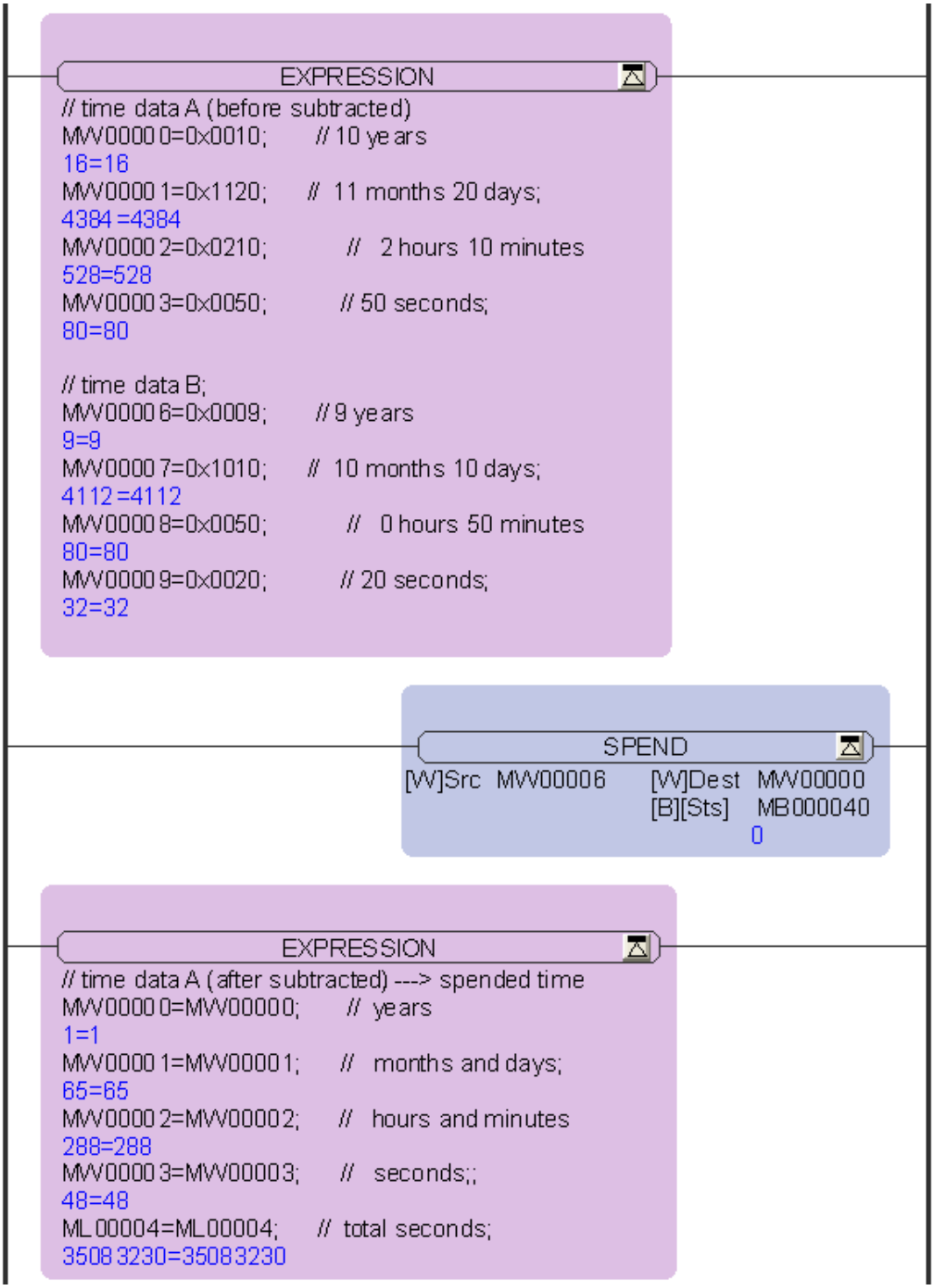
If the operation result exceeds any of the data ranges given above, time data A is not updated, the seconds data is set to 9,999, and the status bit is set to 1 (ON).
If the operation result is within the ranges, the status bit is set to 0 (OFF).

A year is calculated as 365 days. Leap years are not supported.
The number of months is not calculated. Only the number of days is calculated.

## ( 3 ) Programming Example

The following table gives typical conditions for creating ladder programming that uses the SPEND instruction.
(The elapsed time between November 20, 2010, 02:10:50 and October 10, 2009, 00:50:20 is found.)

| | Time A before Execution of Instruction | Time B |
|---|---|---|
| Year | MW00000 = 0010 hex<br>(2010) | MW00006 = 0009 hex<br>(2009) |
| Month/day | MW00001 = 1120 hex<br>(November 20) | MW00007 = 1010 hex<br>(October 10) |
| Hour/minutes | MW0002 = 0210 hex<br>(2:10) | MW00008 = 0050 hex<br>(0:50) |
| Seconds | MW00003 = 0050 hex<br>(50 seconds) | MW00009 = 0020 hex<br>(20 seconds) |




The execution result of this SPEND instruction example is shown below.

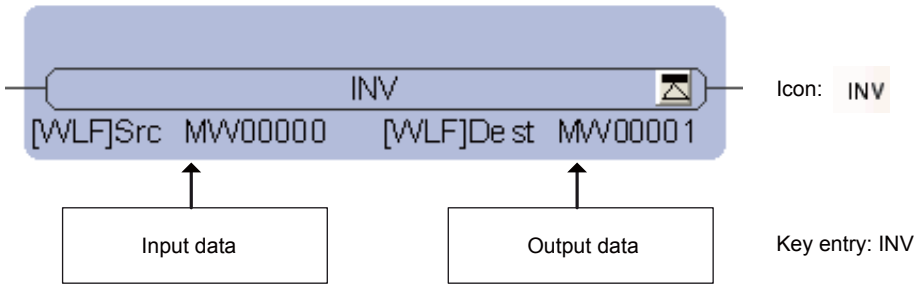
| | Time A after Execution of Instruction |
|---|---|
| Years | MW00000 = 1 = 0001 hex<br>(1 year) |
| Months/days | MW00001 = 65 = 0041 hex<br>(0 months, 41 days) |
| Hours/minutes | MW00002 = 288 = 0120 hex<br>(1 hour, 20 minutes) |
| Seconds | MW00003 = 48 = 0030 hex<br>(30 seconds) |
| Total number of seconds | ML00004 = 35083230 |

# 5.3.15 Invert Sign (INV)

## ( 1 ) Operation

The sign of the input data is inverted and the result is stored in the output data.

## ( 2 ) Format

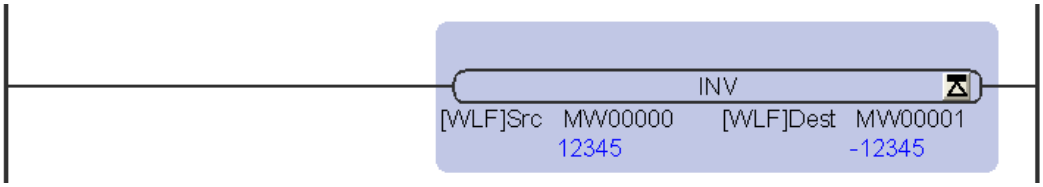



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the INV instruction inverts the sign of 12,345 in input data A in MW00000 and stores the result in the output data in ML00002.
-1 × MW00000 (12,345) → ML00002 = -12,345



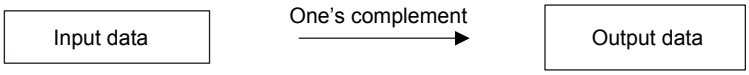

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

# 5.3.16 One's Complement (COM)

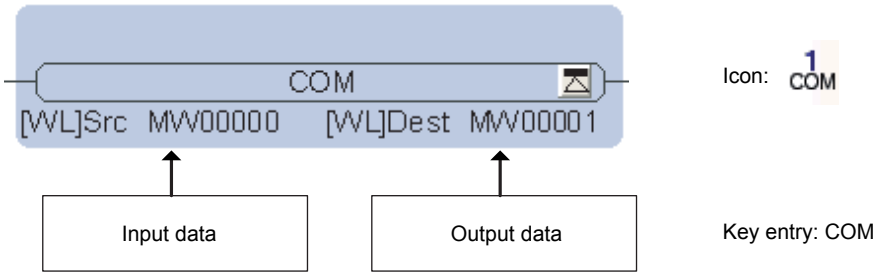
## ( 1 ) Operation

The one's complement of the input data is stored in the output data.




- This instruction inverts the 0's and 1's in the binary representation of the input data and stores the result in the output data.

## ( 2 ) Format

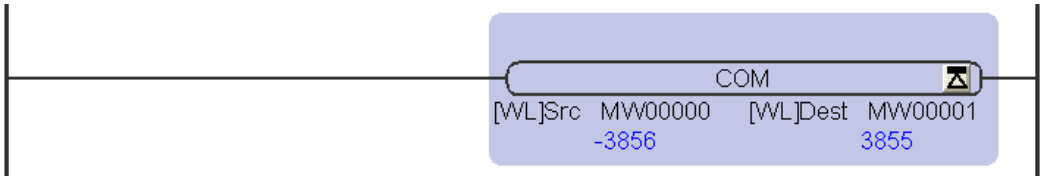



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | × | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the one's complement of -3,856 (F0F0 hex) in the input data in MW00000 is stored in the output data in MW00001.
MW00000 = -3,856 (F0F0 hex) → MW00001 = 3,855 (0F0F hex)



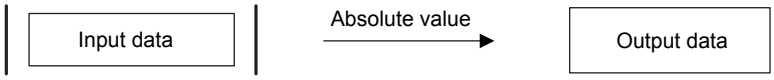

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

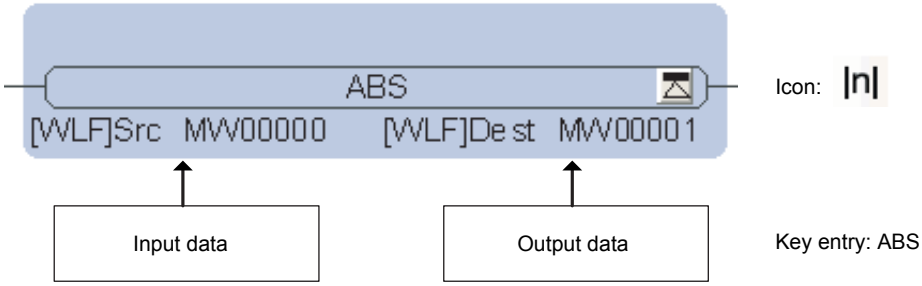
# 5.3.17 Absolute Value (ABS)

## ( 1 ) Operation

The absolute value of the input data is stored in the output data.




## ( 2 ) Format

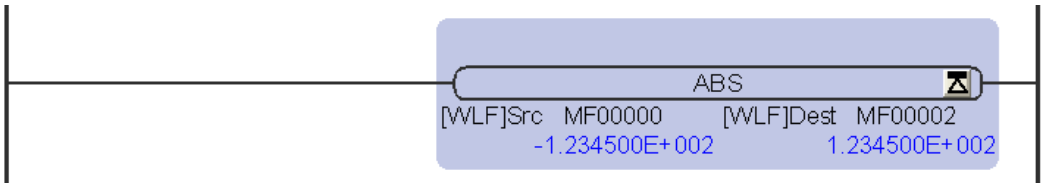



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the absolute value of -123.45 in the input data in MF00000 is stored in the output data in MF00002.
| MF00000(-123.45) | → MF00002 = 123.45



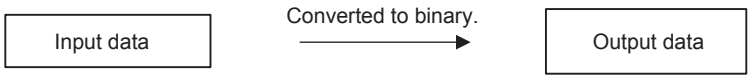

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

# 5.3.18 Binary Conversion (BIN)

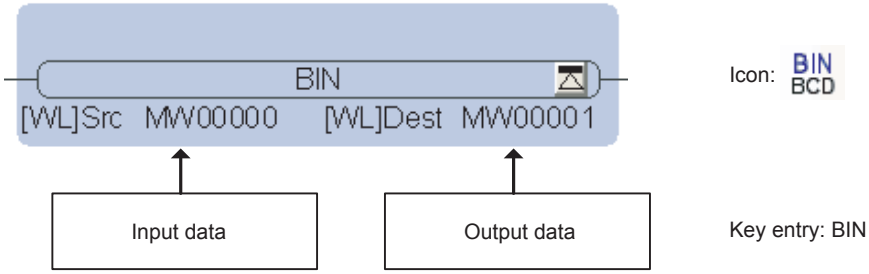
## ( 1 ) Operation

The value of the input data is converted from BCD data to binary data and stored in the output data.
If the input data is not BCD data, such as 123F hex, the result of the binary conversion will be incorrect.




The output data is computed as shown below when the input BCD data is abcd.
Output data = (a × 1,000) + (b × 100) + (c × 10) + d

## ( 2 ) Format

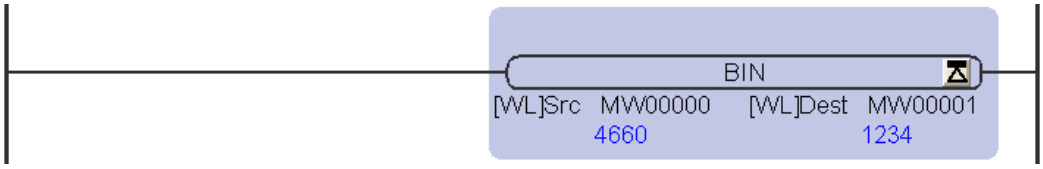



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | O | O | × | × | O | O |
| Output data (Dest) | × | O* | O* | × | × | O | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the value 4,660 (1234 hex) in input data A in MW00000 is converted to binary and stored in the output data in MW00001.
MW00000 = 1234 hex: (1 × 1,000) + (2 × 100) + (3 × 10) + 4 → MW00001 = 1,234



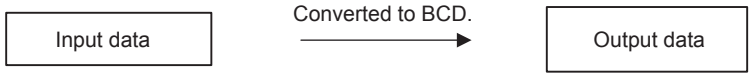

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

# 5.3.19 BCD Conversion (BCD)

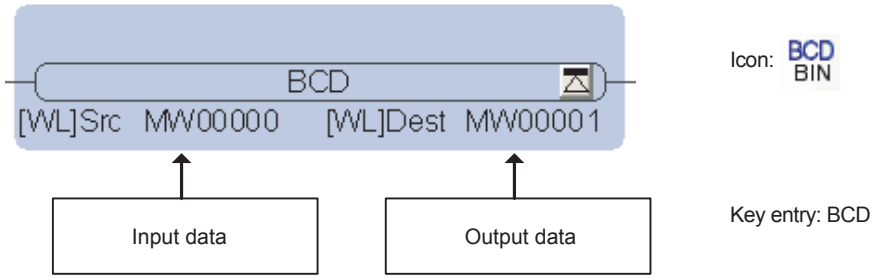
## ( 1 ) Operation

The input data is converted from binary data to BCD data and stored in the output data.
If the input data is greater than 9,999, or a negative value, the result will be incorrect.




The output data is computed as shown below when the input decimal data is abcd.
Output data = (a × 49) + (b × 256) + (c × 16) + d

## ( 2 ) Format

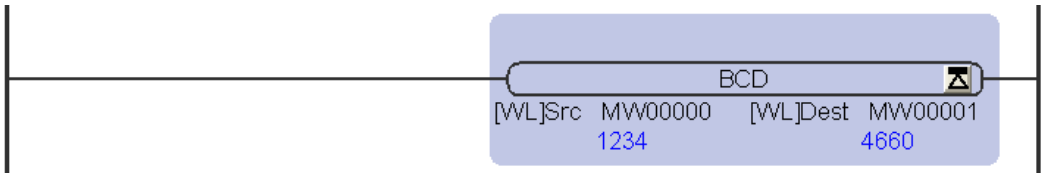



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | × | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the value 1,234 in input data A in MW00000 is converted to BCD and stored in the output data in MW00001.
MW00000 = 1,234: (1 × 4,096) + (2 × 256) + (3 × 16) + 4 → MW00001 = 4,660 (1234 hex)



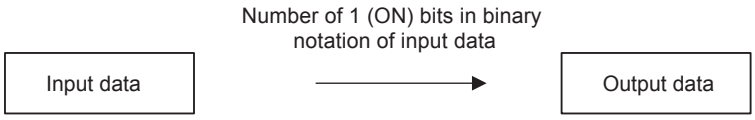

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

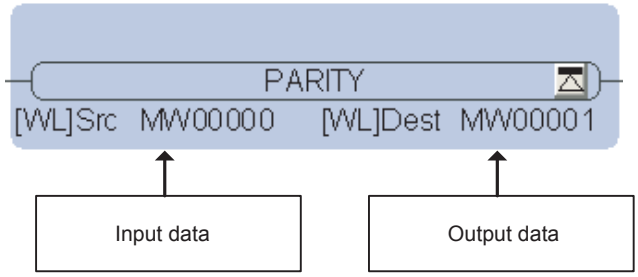
# 5.3.20 Parity Conversion (PARITY)

## ( 1 ) Operation

The number of bits set to 1 (ON) in the input data is calculated in binary notation and stored in the output data.




## ( 2 ) Format




Icon: 0101 #?

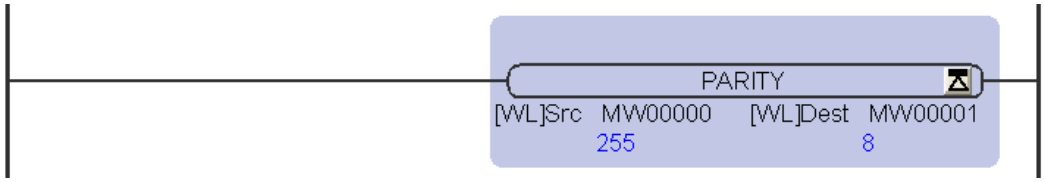
Key entry: PARITY

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | ○ | × | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | × | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the number of bits set to 1 (ON) in 255 (00FF hex) in the input data A in MW00000 is stored in the output data in MW00001.
Number of 1 bits in MW00000 (0FF hex) = 8 → MW00001 = 8



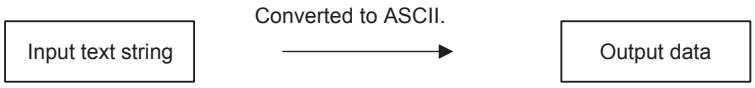

INFO

When performing operations with different data types, the result of the operation will depend on the data type of the output register.
Refer to *4.4.2 ( 3 ) Precautions When Using Local Registers within a User Function* for details.

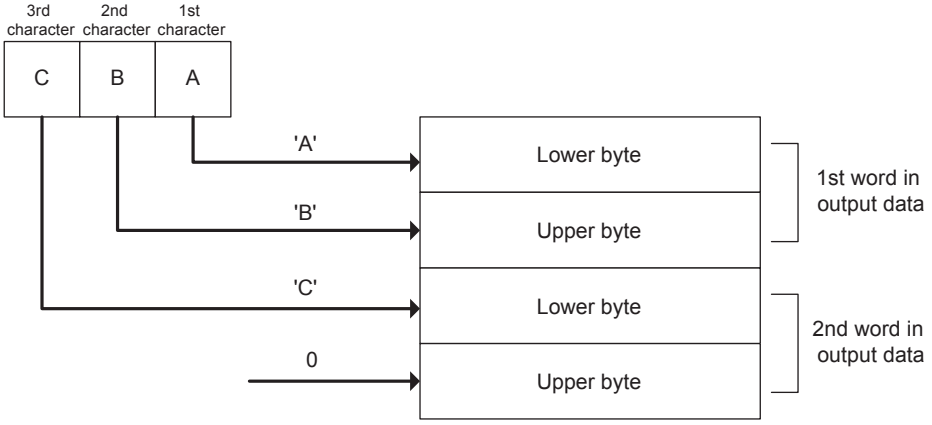
# 5.3.21 ASCII Conversion 1 (ASCII)

## ( 1 ) Operation

The input text string is converted to ASCII and stored in the output data. The text string is case sensitive.
The input text string can contain up to 32 characters (16 words).

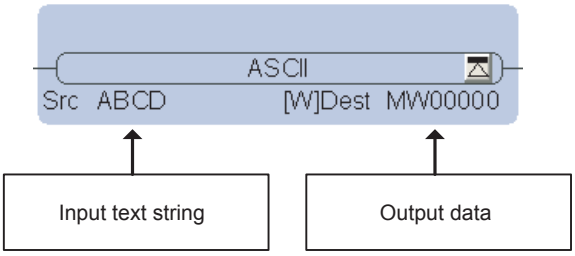



### ■ Storage Location of ASCII Values for Input Text String




- If the text string contains an odd number of characters, the upper byte of the last word is set to zeros.

## ( 2 ) Format




Icon: ASCII

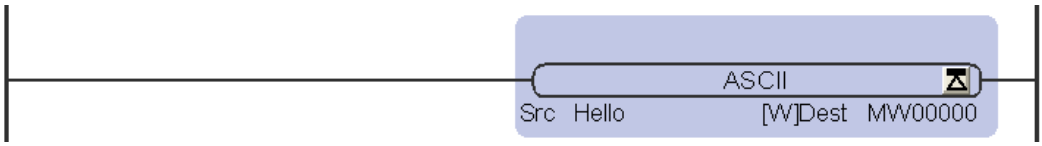
Key entry: ASCII

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input text string (Src) | | | | ×*1 | | | |
| Output data (Dest) | × | ○*2 | × | × | × | × | × |

∗ 1. ASCII text
∗ 2. C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the input string “Hello” is converted to ASCII and stored in the output data in MW00000.



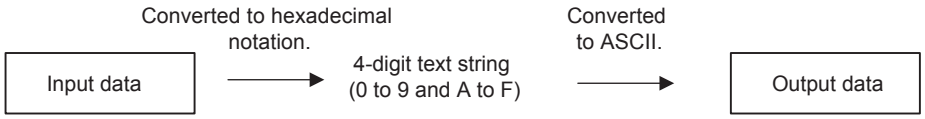

The ASCII values are stored as given in the following table.

| Address | ASCII Value | Character |
|---|---|---|
| MW00000 (lower byte) | 48 hex | H |
| MW00000 (upper byte) | 65 hex | e |
| MW00001 (lower byte) | 6C hex | l |
| MW00001 (upper byte) | 6C hex | l |
| MW00002 (lower byte) | 6F hex | o |
| MW00002 (upper byte) | 0 | - |

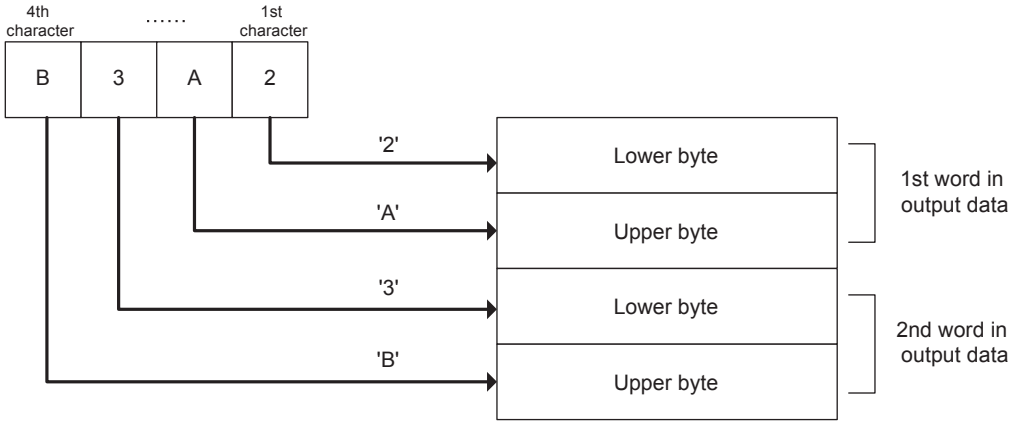
# 5.3.22 ASCII Conversion 2 (BINASC)

## ( 1 ) Operation

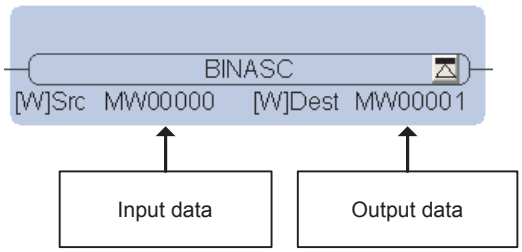
The 16-bit binary data stored in the 1-word input data is converted to four-digit hexadecimal ASCII and stored in the 2-word output data.




■ Storage Location of ASCII Values for Input Data of 10,811(2A3B hex)




## ( 2 ) Format




Icon: BIN ASC

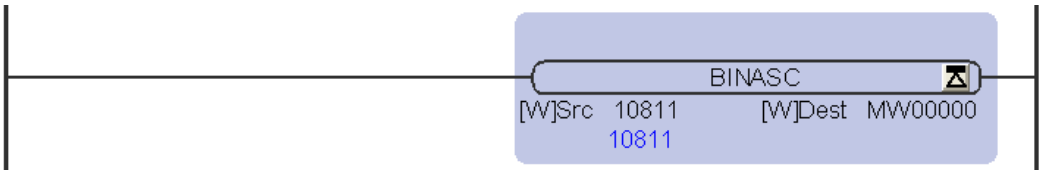
Key entry: BINASC

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | O | × | × | × | × | O |
| Output data (Dest) | × | O* | × | × | × | × | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, 10,811 (2A3B hex) in the input data is converted to ASCII and stored in the output data in MW00000.



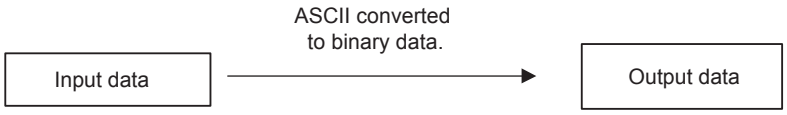

The ASCII values are stored as given in the following table.

| Address | ASCII Value | Character |
|---|---|---|
| MW00000 (lower byte) | 32 hex | 2 |
| MW00000 (upper byte) | 41 hex | A |
| MW00001 (lower byte) | 33 hex | 3 |
| MW00001 (upper byte) | 42 hex | B |

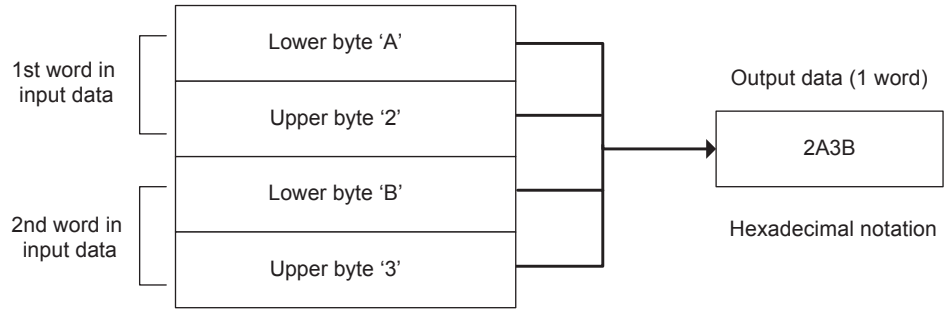
# 5.3.23 ASCII Conversion 3 (ASCBIN)

## ( 1 ) Operation

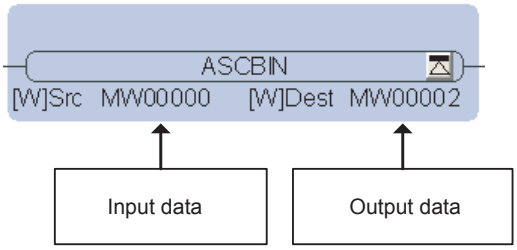
The value given in 4-digit hexadecimal ASCII and stored in the 2-word input data is converted to 16-bit binary data and stored in 1-word output data.




■ Output Data When First Word of Input Data Is 4132 Hex (‘2’ ‘A’) and Second Word Is 4232 Hex (‘3’ ‘B’)




## ( 2 ) Format




Icon: ASC BIN

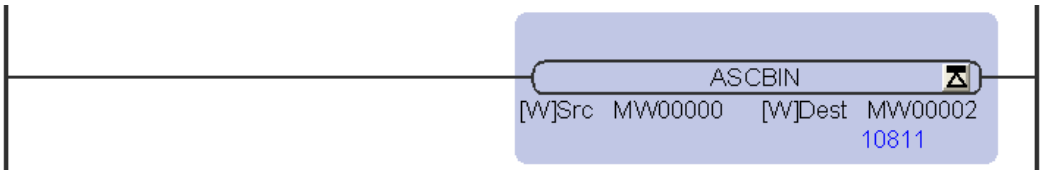
Key entry: ASCBIN

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | O | × | × | × | × | O |
| Output data (Dest) | × | O* | × | × | × | × | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the ASCBIN instruction is used to store the input data in MW00000 in the output data in MW00002.




The ASCII values are stored as given in the following table.

| Address | ASCII Value | Character |
|---|---|---|
| MW00000 (lower byte) | 32 hex | 2 |
| MW00000 (upper byte) | 41 hex | A |
| MW00001 (lower byte) | 33 hex | B |
| MW00001 (upper byte) | 42 hex | 3 |

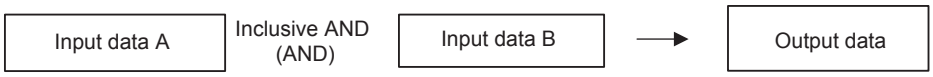
The output data in MW00000 is set to 10,811 (2A3B hex).

# 5.4 Logic Operations and Comparison Instructions

## 5.4.1 Inclusive AND (AND)

### ( 1 ) Operation

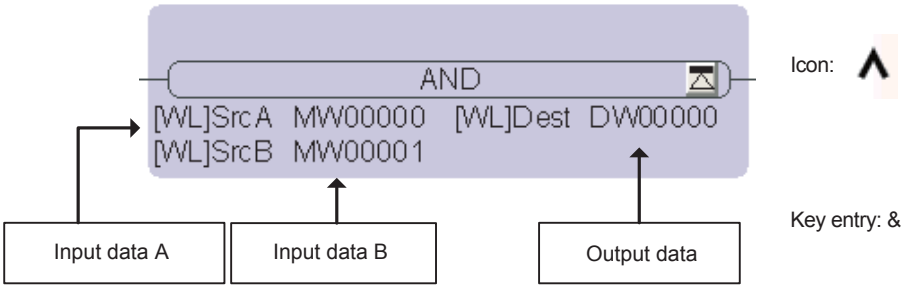
The AND instruction performs a logical AND operation on input data A and input data B and the result is stored in the output data.
This instruction can be used only with integer or double-length integer data.




Each bit in the input data is evaluated as shown in the following truth table.

| Input data A | Input data B | Output data |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

### ( 2 ) Format

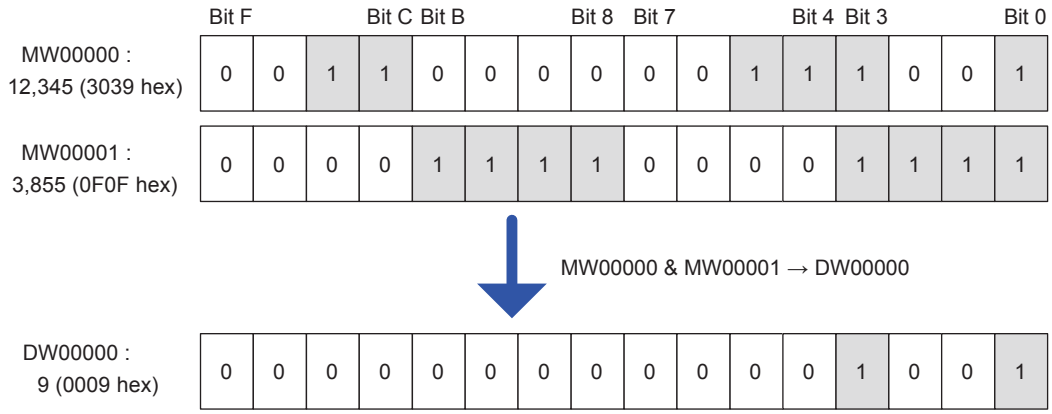


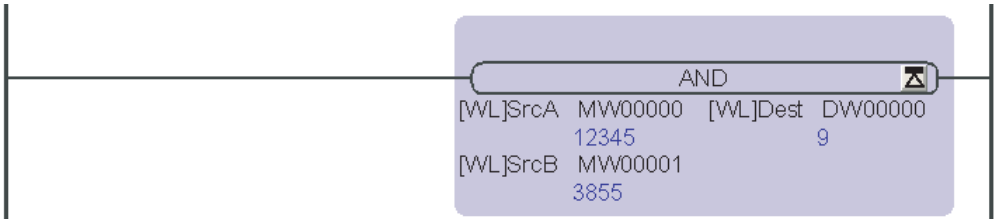

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, a logical AND is performed on 12,345 (3039 hex) in input data A in MW00000 and 3,855 (0F0F hex) in input data B in MW00001, and the result is stored in the output data in DW00000.

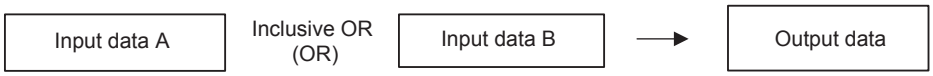


## 5.4.2 Inclusive OR (OR)

### ( 1 ) Operation

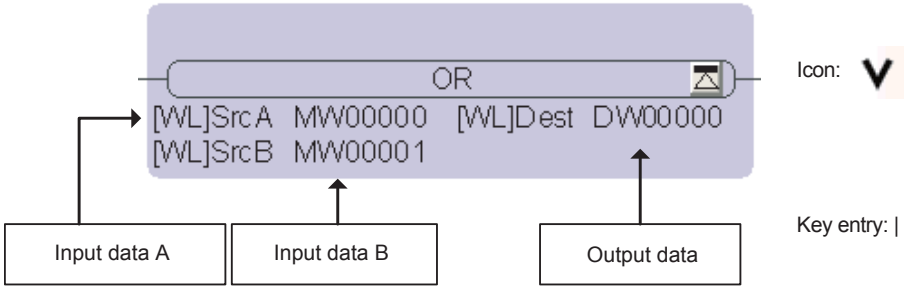
The OR instruction performs a logical OR operation on input data A and input data B and the result is stored in the output data.
This instruction can be used only with integer or double-length integer data.




Each bit in the input data is evaluated as shown in the following truth table.

| Input data A | Input data B | Output data |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

### ( 2 ) Format

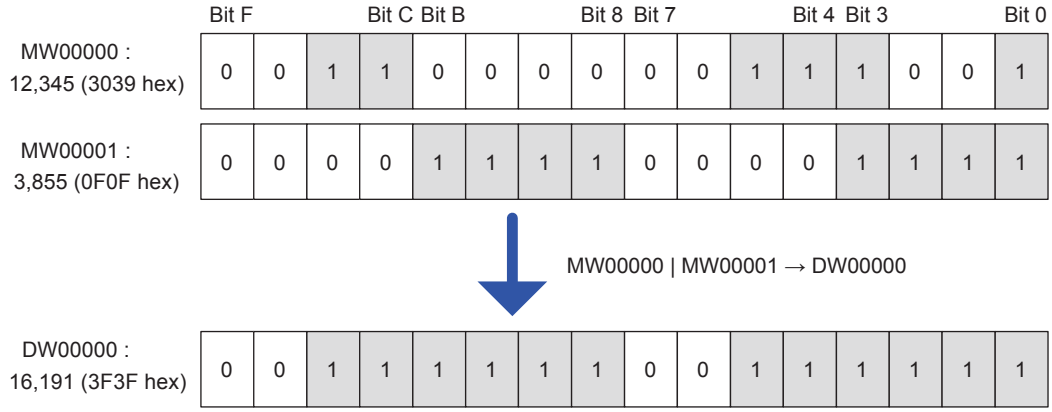


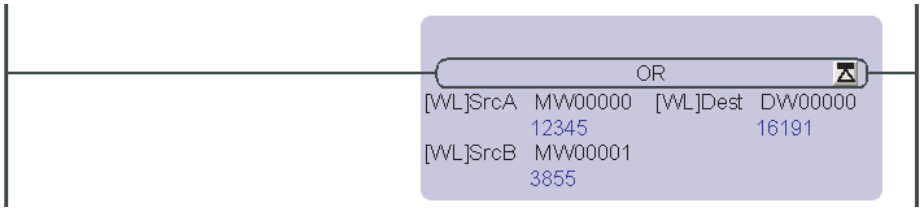

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, a logical OR is performed on 12,345 (3039 hex) in input data A in MW00000 and 3,855 (0F0F hex) in input data B in MW00001, and the result is stored in the output data in DW00000.

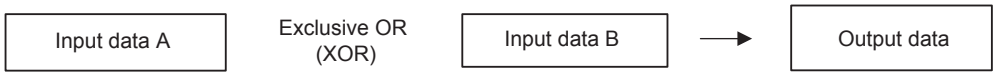


## 5.4.3 Exclusive OR (XOR)

### ( 1 ) Operation

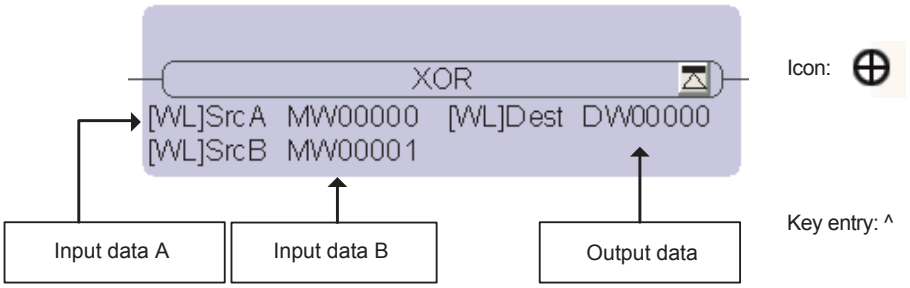
The XOR instruction performs an exclusive logical OR operation on input data A and input data B and the result is stored in the output data.
This instruction can be used only with integer or double-length integer data.




Each bit in the input data is evaluated as shown in the following truth table.

| Input data A | Input data B | Output data |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### ( 2 ) Format

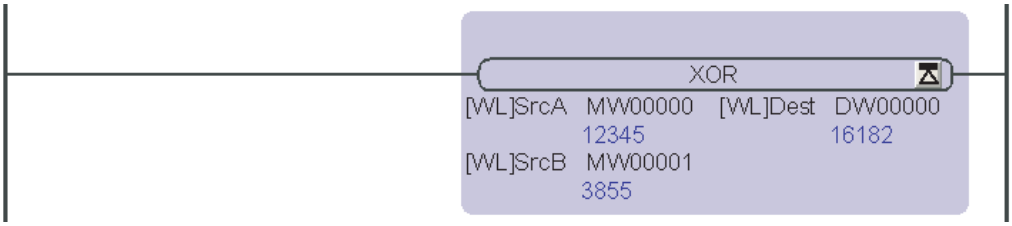



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

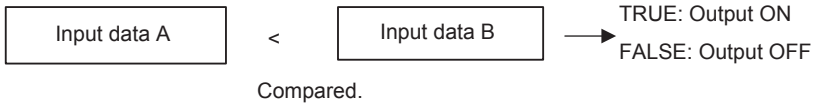
In the following programming example, an exclusive logical OR is performed on 12,345 (3039 hex) in input data A in MW00000 and 3,855 (0F0F hex) in input data B in MW00001, and the result is stored in the output data in DW00000.



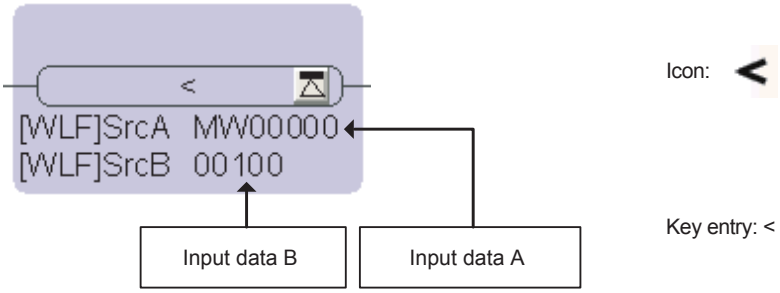
## 5.4.4 Less Than (<)

### ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.




### ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

### ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is executed because the comparison is true and turns the output ON; that is, input data A is less than input data B when input data A in MW00000 is 90 and input data B is 100.

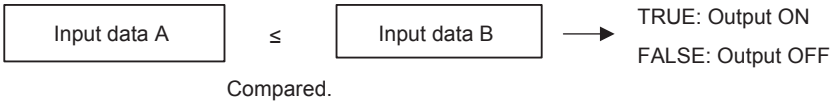
INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

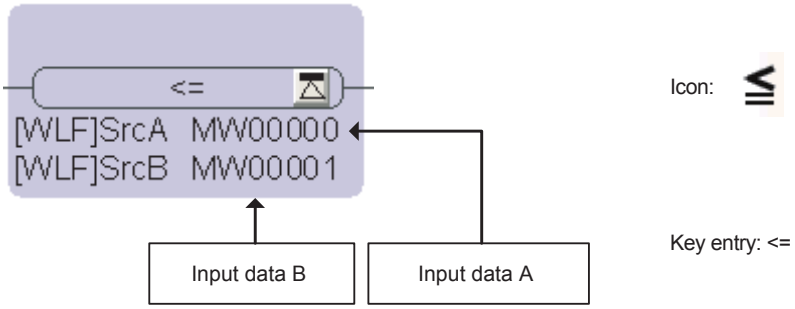
## 5.4.5 Less Than or Equal (≤)

### ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.



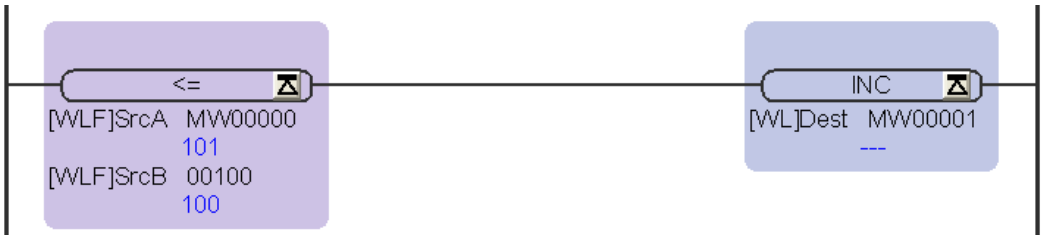

### ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

### ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is not executed because the comparison is false and turns the output OFF; that is, input data A is not less than or equal to input data B when input data A in MW00000 is 101 and input data B is 100.



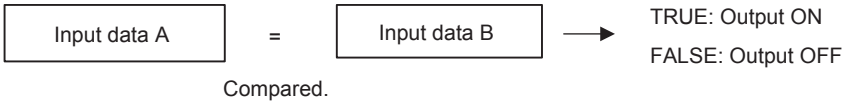

INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

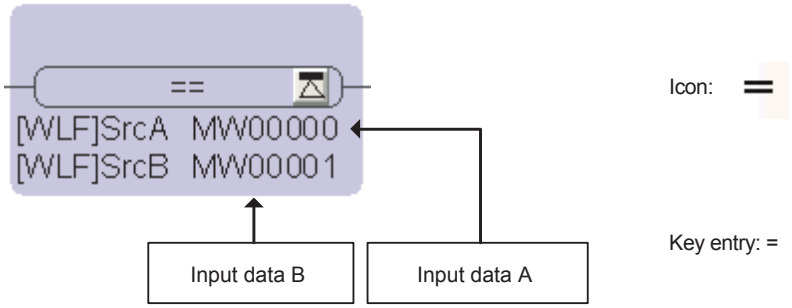
# 5.4.6 Equal (=)

## ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.




## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

## ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is executed because the comparison is true and turns the output ON; that is, input data A is equal to input data B when input data A in MW00000 is 100 and input data B is 100.

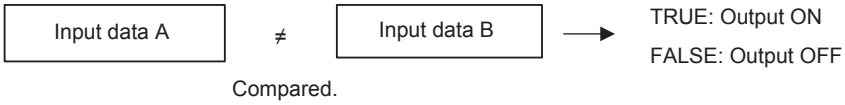
INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

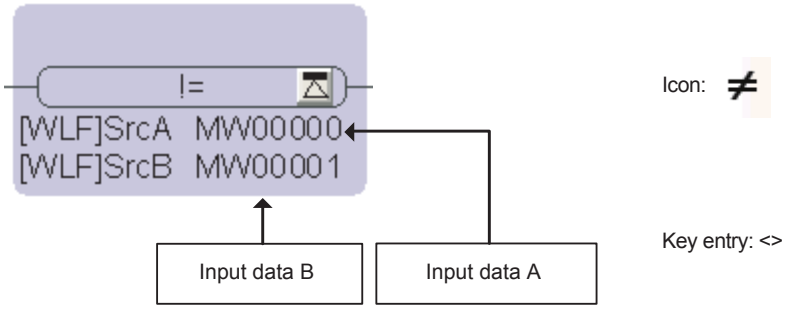
# 5.4.7 Not Equal (≠)

## ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.




## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

## ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is not executed because the comparison is false and turns the output OFF; that is, input data A is equal to input data B when input data A in MW00000 is 100 and input data B is 100.

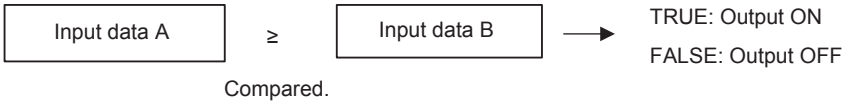
INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

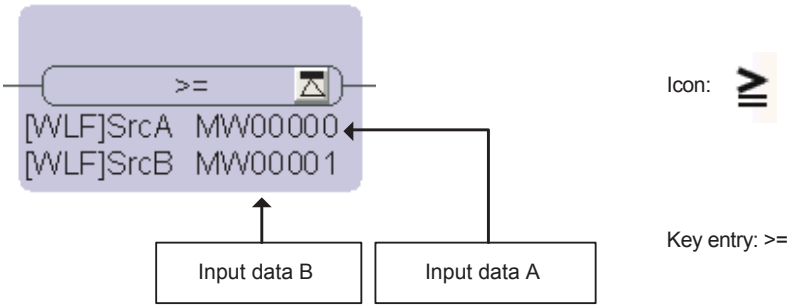
## 5.4.8 Greater Than or Equal (≥)

### ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.




### ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

### ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is executed because the comparison is true and turns the output ON; that is, input data A is greater than or equal to input data B when input data A in MW00000 is 100 and input data B is 100.

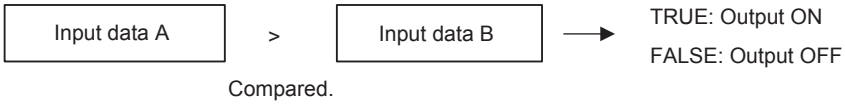
INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

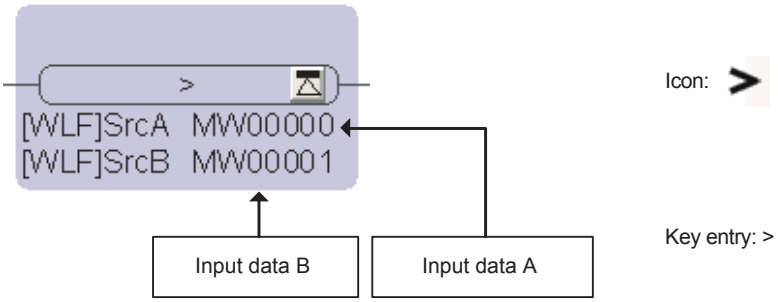
# 5.4.9 Greater Than (>)

## ( 1 ) Operation

Input data A and input data B are compared and the result is stored in the bit output.




## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data A (SrcA) | × | ○ | ○ | ○ | × | ○ | ○ |
| Input data B (SrcB) | × | ○ | ○ | ○ | × | ○ | ○ |

## ( 3 ) Programming Example

In the following programming example, the INC instruction on the right end of the line is not executed because the comparison is false and turns the output OFF; that is, input data A is not greater than input data B when input data A in MW00000 is 100 and input data B is 100.

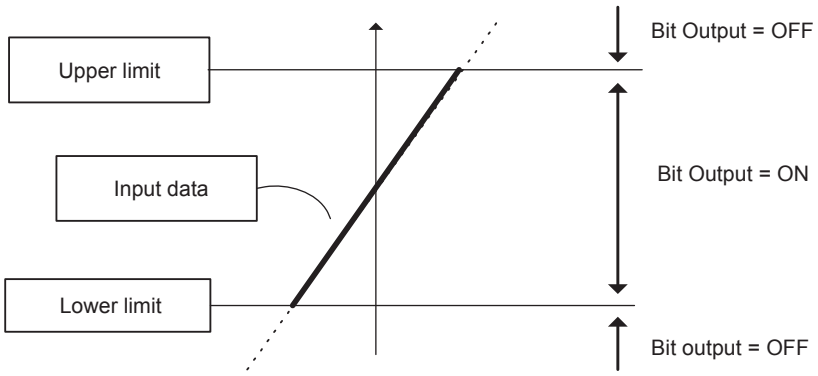
INFO

With real number data, the value displayed by the MPE720 may not match the execution result of the comparison instruction due to a slight precision error.

# 5.4.10 Range Check (RCHK)

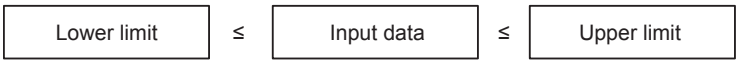
## ( 1 ) Operation

The RCHK instruction checks to see if the input data is between the upper limit and lower limit and the result is stored in the bit output.




### [ a ] Bit Output = ON

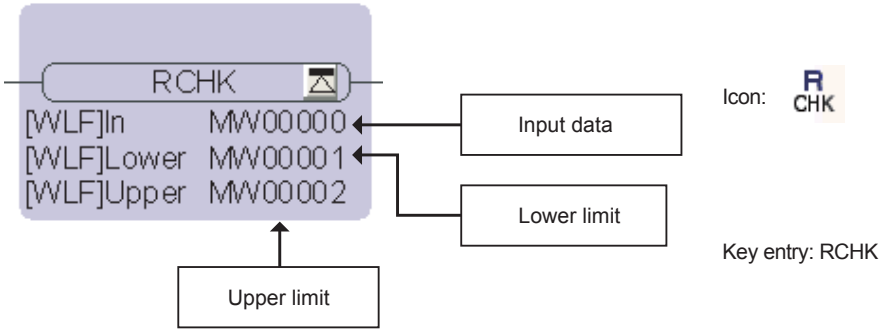
The bit output is turned ON if the value of the input data is within the range that is greater than or equal to the lower limit, and less than or equal to the upper limit.




### [ b ] Bit Output = OFF

The bit output is turned OFF if the value of the input data is outside the range that is greater than or equal to the lower limit, and less than or equal to the upper limit.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| Lower limit (Lower) | × | ○ | ○ | ○ | × | ○ | ○ |
| Upper limit (Upper) | × | ○ | ○ | ○ | × | ○ | ○ |

INFO

Always set the lower limit to a value that is less than or equal to the upper limit. If the lower limit is greater than the upper limit, the result will be invalid.

## ( 3 ) Programming Examples

The following programming examples execute the RCHK instruction.

- When Input Data (MW00000) = 80, Lower Limit = 100, and Upper Limit = 1,000
  The INC instruction on the right end of the line is not executed because the input data is less than the lower limit and turns the bit output OFF.

- When Input Data (MW00000) = 500, Lower Limit = 100, and Upper Limit = 1,000
  The INC instruction on the right end of the line is executed because the value of the input data is within the range that is greater than or equal to the lower limit and less than or equal to the upper limit, which turns ON the bit output.

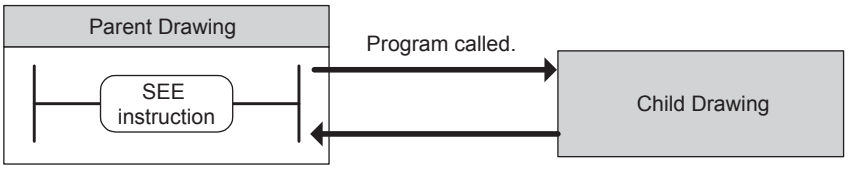
- When Input Data (MW00000) = 1,000, Lower Limit = 100, and Upper Limit = 1,000
  The INC instruction on the right end of the line is executed because the value of the input data is within the range that is greater than or equal to the lower limit and less than or equal to the upper limit, which turns ON the bit output.

# 5.5 Program Control Instructions

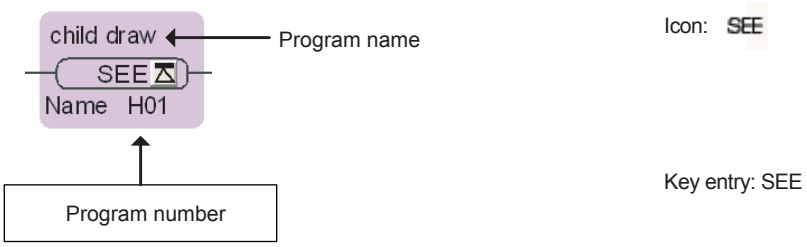
## 5.5.1 Call Sequence Program (SEE)

### ( 1 ) Operation

The SEE instruction calls a child drawing from a parent drawing, or a grandchild drawing from a child drawing.



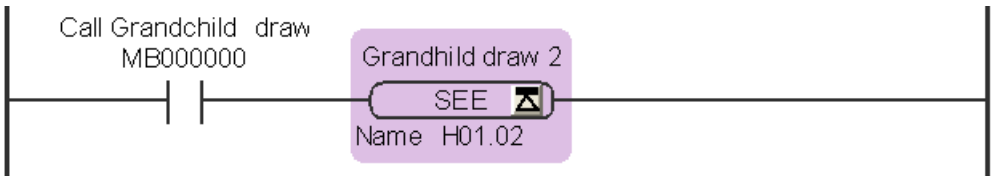

### ( 2 ) Format




| Parameter Name | Applicable Data Types |
|---|---|
| Program number (Name) | Registers cannot be used. Specify the program number directly.<br>The name of the specified program appears above the instruction. |

### ( 3 ) Programming Example

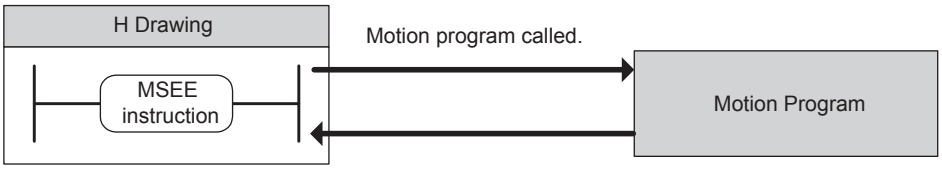
In the following programming example, the SEE instruction calls drawing H01.02 when the MB000000 relay is ON. Thereafter, the process is executed and execution resumes from the next step after the SEE instruction. The SEE instruction does not call drawing H01.02 if the MB000000 relay is OFF.



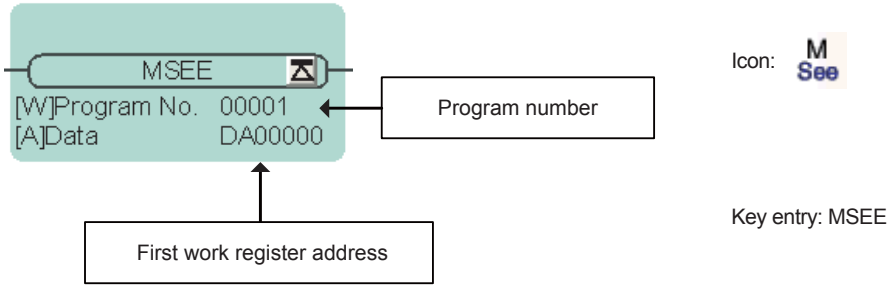
# 5.5.2 Call Motion Program (MSEE)

## ( 1 ) Operation

The MSEE instruction calls the specified motion program.
Motion programs can be called only from H drawings.




## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Program number (Program No.) | × | ○* | × | × | × | × | ○ |
| First work register address (Data) | × | × | × | × | ○* | × | × |

* M or D register only.

### ■ Work Register Configuration

| Address | Data Type | Name | Description | I/O |
|---|---|---|---|---|
| 0 | W | Status Flags | Motion program status flags | OUT |
| 1 | W | Control Signals | Motion program control signals | IN |
| 2 | W | Interpolation Override | The override is used when executing interpolation instructions.<br>Range: 0 to 32,767<br>Unit: 1 = 0.01% | IN |
| 3 | W | System Work Number | This is the system work number that calls the motion program. | IN |

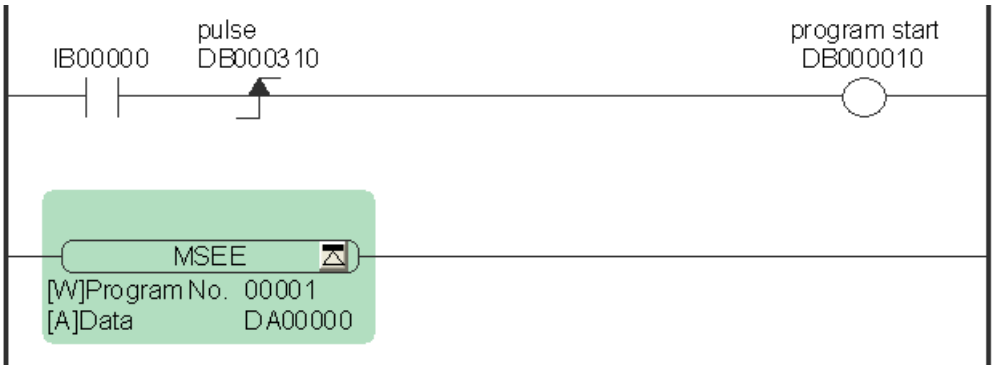
INFO

Specify the program number from 1 to 256.
For details on motion programs, refer to the *Machine Controller MP2000 Series User's Manual for Motion Programming* (Manual No.: SIEP C880700 38).

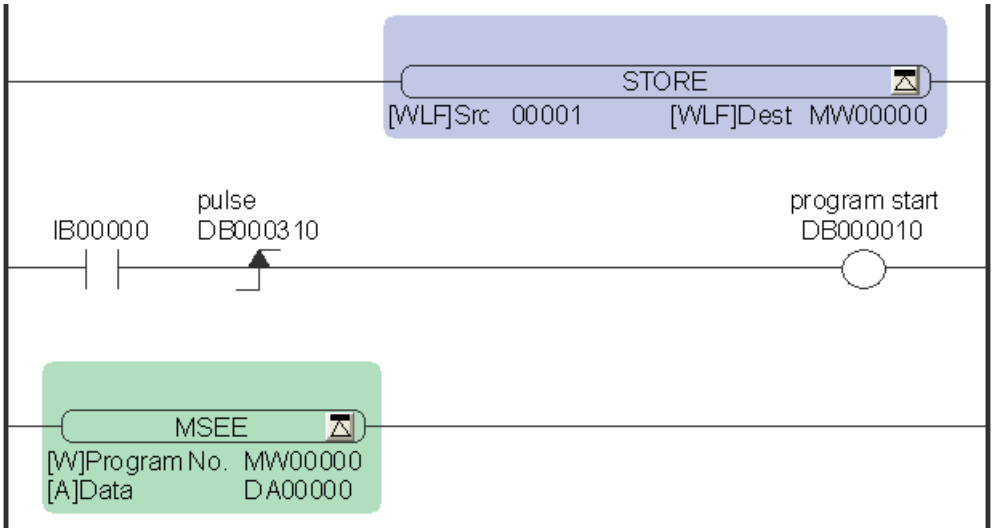
## ( 3 ) Programming Examples

The following programming examples show how to execute the motion program MPM001 with program number 1. When the IB00000 relay turns ON, the Request for Start of Program Operation (DB000010) in the control signals turns ON and executes the MPM001 motion program.

- Direct Designation
  The program number is directly set to 1.




- Indirect Designation
  The program number is set in MW00000.



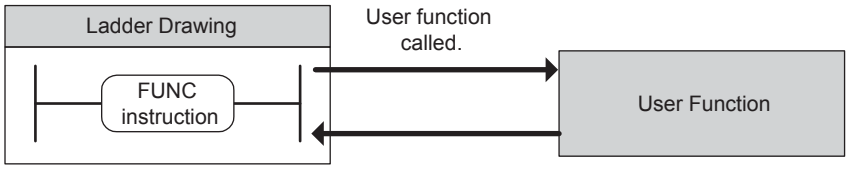

IMPORTANT

Continue execution of the MSEE instruction until execution of the motion program is completed.
When using indirect designation, do not change the register value until the execution of the motion program is completed.

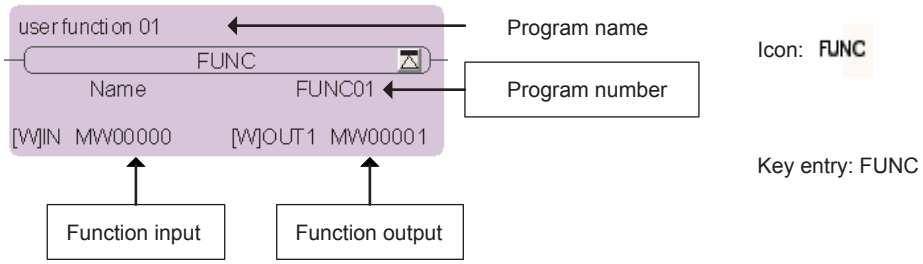
# 5.5.3 Call User Function (FUNC)

## ( 1 ) Operation

The FUNC instruction calls a user function. The user function must be defined before it can be called.
Refer to *4.3 User Functions* for details on user functions.




## ( 2 ) Format




| Parameter Name | Applicable Data Types |
|---|---|
| Program number (Name) | Registers cannot be used. Specify the program number directly.<br>The name of the specified program appears above the instruction. |
| Function input | The register that is set in the function's input definition can be used. |
| Function output | The register that is set in the function's output definition can be used. |

## ( 3 ) Programming Example

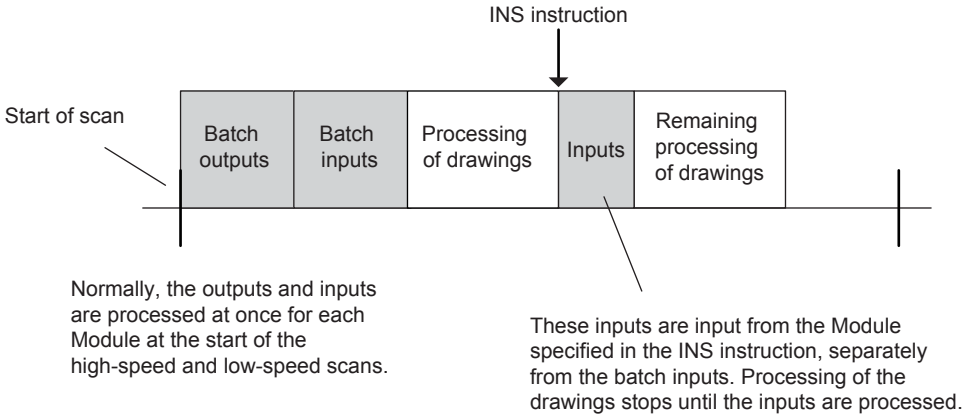
Refer to *4.3 User Functions* for programming examples for user functions.

## 5.5.4 Direct Input String (INS)

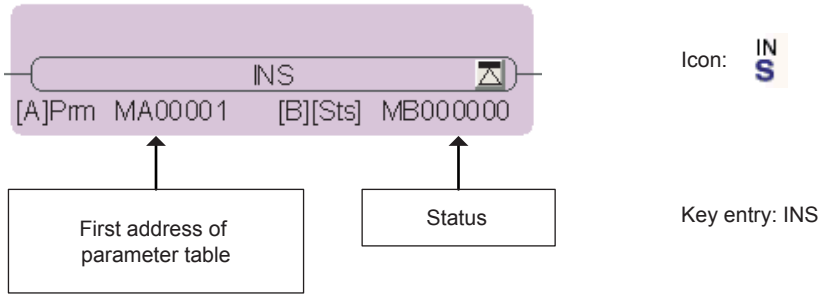
### ( 1 ) Operation

The INS instruction is executed in user programs to input data independently from the I/O batch processing that is performed by the system at the start of the high-speed and low-speed scans. When the INS instruction is executed, the inputs from the specified Module are processed according to the settings in a parameter table. The next instruction is not executed until input processing is completed.
The following Modules can be specified.

- CPU Module (IO)
- LIO-01/02 Module (LIO)
- LIO-04/05 Module (LIO32)
- LIO-06 Module (MIXIO)
- AI-01 Module (AI)




### ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of parameter table (Prm) | × | × | × | × | ○*1 | × | × |
| Status (Sts)*2 | ○*1 | × | × | × | × | × | × |

∗ 1. C and # registers cannot be used.
∗ 2. Optional.

## ■ Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RSSEL | Unit selection 1 | Specify the Module to input from. | IN |
| 1 | W | MDSEL | Unit selection 2 | | IN |
| 2 | W | STS | Status | Each bit receives the input status for one word.<br>0: Normal, 1: Error | OUT |
| 3 | W | N | Number of words | Specify the number of continuous words. | IN |
| 4 | W | ID1 | Input data 1 | Receives the data that was input.<br>Contains 0 if an error occurs. | OUT |
| : | : | : | : | | : |
| N + 3 | W | IDN | Input data N | | OUT |

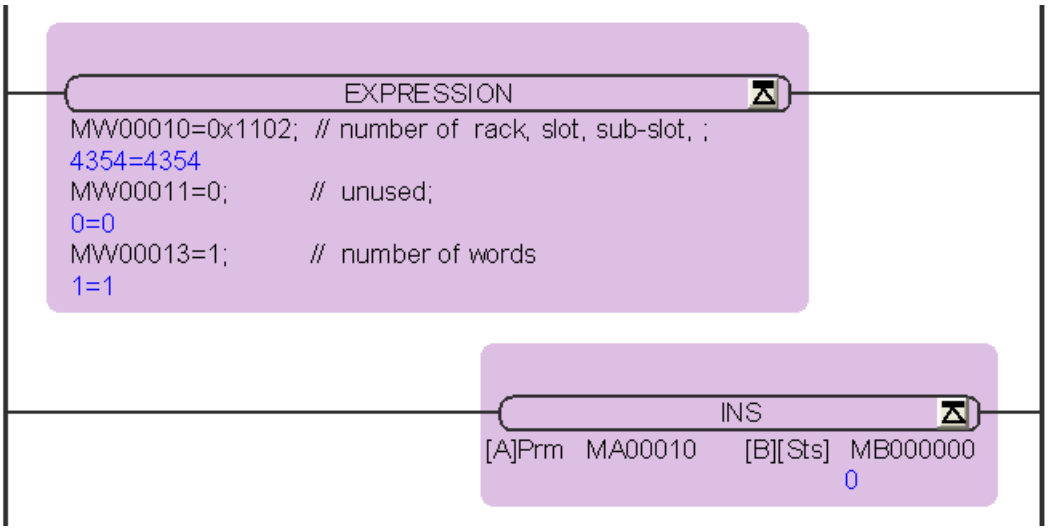
The following table gives details about the parameters in the MP2000-series Controller.

| Module Name / Parameter | CPU (IO) | LIO-01/02 (LIO) | LIO-04/05 (LIO32) | LIO-06 (MIXIO) | AI-01 (AI) |
|---|---|---|---|---|---|
| RSSEL | Specify the rack, slot, and subslot of the target Module.<br>Hexadecimal notation: zxyy hex<br>x: Rack number from 1 to 4<br>yy: Slot number from 0 to 9<br>z: Subslot number from 1 to maximum value (determined by Module specifications) | | | | |
| MDSEL | 0 (Not used.) | 0 (Not used.) | Offset: 0 or 1 | Channel number - 1: 0 or 1 | Channel number - 1: 0 to 7 |
| STS | Always 0. | Always 0. | Always 0. | Always 0. | * |
| N | 1 | 1 | 1 or 2 – MDSEL | 1 or 2 – MDSEL | 1 to 8 – MDSEL |

* If a channel for which the allocation has been deleted in the AI Module detailed definition is specified for the INS instruction, the applicable channel number is output for the bit. This is because it is not possible to read the data on channels for which allocations have been deleted.
The relation between bits and channels is shown below.
Bit 0: Channel 1
Bit 1: Channel 2
Bit 2: Channel 3
Bit 3: Channel 4
Bit 4: Channel 5
Bit 5: Channel 6
Bit 6: Channel 7
Bit 7: Channel 8

## ( 3 ) Programming Example

When one word is input from the LIO at subslot number 1 on the LIO-01 Module mounted in rack 1 and slot 2, the input data of the LIO is stored in MW00014.



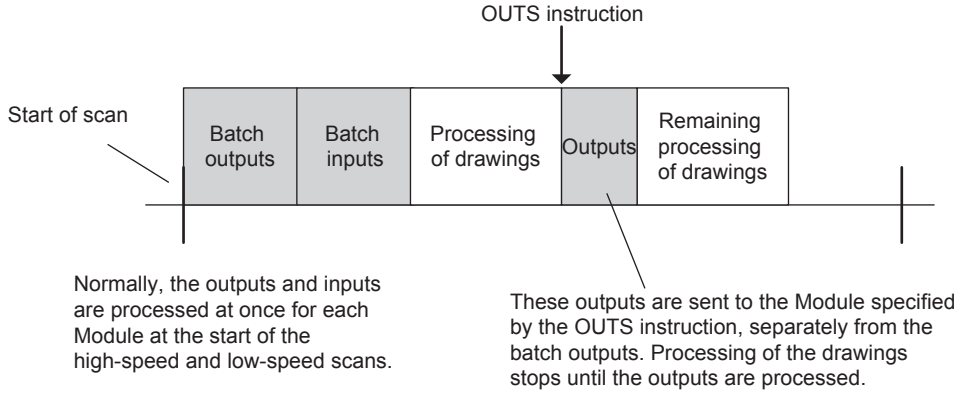
# 5.5.5 Direct Output String (OUTS)

## ( 1 ) Operation

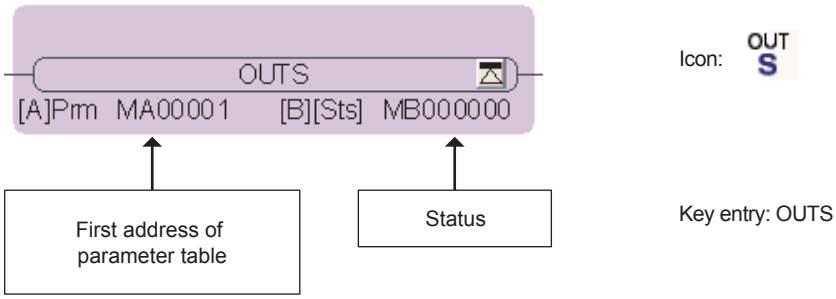
The OUTS instruction is executed in user programs to output data independently from the I/O batch processing that is performed by the system at the start of the high-speed and low-speed scans. When the OUTS instruction is executed, the outputs from the specified Module are processed according to the settings in the parameter table.

The following Modules can be specified.

- CPU Module (IO)
- LIO-01/02 (LIO)
- LIO-04/05 (LIO32)
- LIO-06 (MIXIO)
- DO-01 (DO)
- AO-01 (AO)




## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of parameter table (Prm) | × | × | × | × | ○*1 | × | × |
| Status (Sts)*2 | ○*1 | × | × | × | × | × | × |

∗ 1. C and # registers cannot be used.
∗ 2. Optional.

## ■ Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RSSEL | Unit selection 1 | Specify the Module to output to. | IN |
| 1 | W | MDSEL | Unit selection 2 | | IN |
| 2 | W | STS | Status | Each bit receives the input status for one word.<br>0: Normal, 1: Error | OUT |
| 3 | W | N | Number of words | Specify the number of output words (always 1). | IN |
| 4 | W | OD1 | Output data 1 | Specify the data to output. | OUT |
| : | : | : | : | | : |
| N + 3 | W | ODN | Output data N | | OUT |

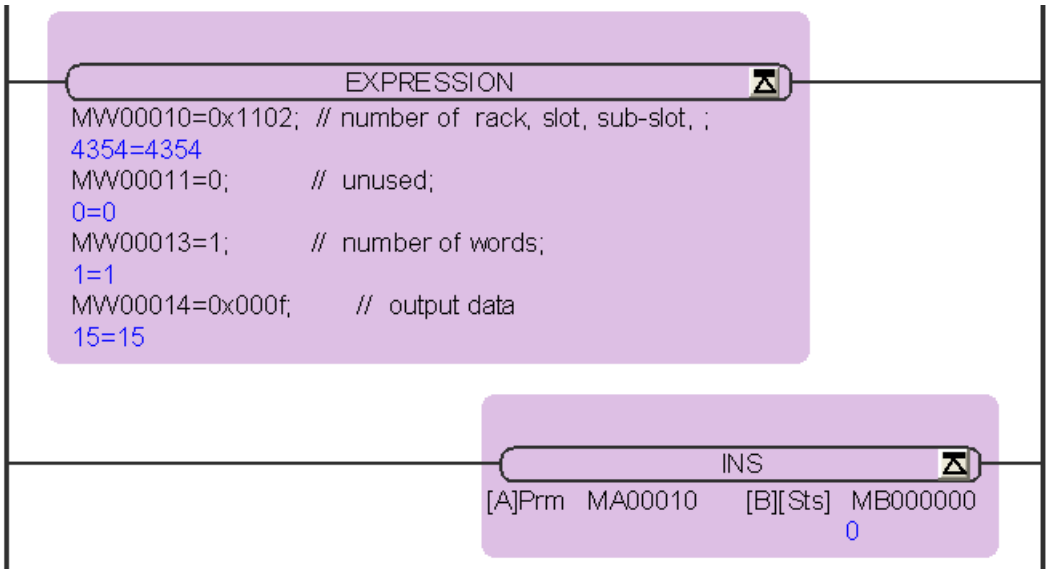
The following table gives details about the parameters in the MP2000-series Controller.

| Parameter \ Module Name | CPU (IO) | LIO-01/02 (LIO) | LIO-04/05 (LIO32) | LIO-06 (MIXIO) | DO-01 (DO) | AO-01 (AO) |
|---|---|---|---|---|---|---|
| RSSEL | Specify the rack, slot, and subslot of the target Module.<br>Hexadecimal notation: zxyy hex<br>x: Rack number from 1 to 4<br>yy: Slot number from 0 to 9<br>z: Subslot number from 1 to maximum value (determined by Module specifications) | | | | | |
| MDSEL | 0 (Not used.) | 0 (Not used.) | Offset: 0 or 1 | Offset: 0 or 1 | Offset: 0 to 3 | Channel number - 1: 0 to 3 |
| STS | Always 0. | Always 0. | Always 0. | Always 0. | Always 0. | * |
| N | 1 | 1 | 1 or 2<br>– MDSEL | 1 or 2<br>– MDSEL | 1 to 4<br>– MDSEL | 1 to 4<br>– MDSEL |

* If a channel for which the allocation has been deleted in the AO Module detailed definition is specified for the OUTS instruction, the applicable channel number is output for the bit. This is because it is not possible to read the data on channels for which allocations have been deleted.
The relation between bits and channels is shown below.
Bit 0: Channel 1
Bit 1: Channel 2
Bit 2: Channel 3
Bit 3: Channel 4

## ( 3 ) Programming Example

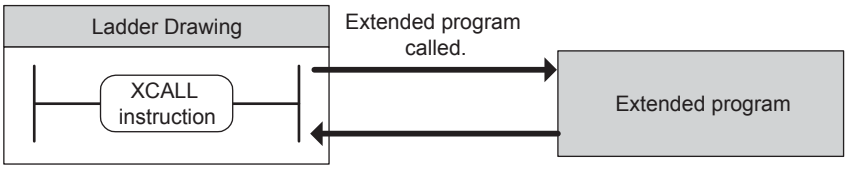
When one word is output to the LIO at subslot number 1 on the LIO-01 Module mounted in rack 1 and slot 2, the data in MW00014 is output to LIO.



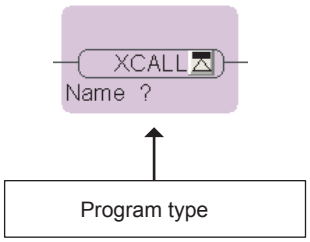
# 5.5.6 Call Extended Program (XCALL)

## ( 1 ) Operation

An extended program (i.e., a table program, such as a constants table, an I/O conversion table, an interlock table, or a part composition table) is executed. The MPE720 converts the extended program into a ladder program. Converted ladder programs can be executed with the XCALL instruction.
Although more than one XCALL instruction can be used in a single drawing, the same extended program cannot be called more than once.




## ( 2 ) Format




Icon:


Key entry: XCALL

| Parameter Name | Applicable Data Types |
|---|---|
| Program type (Name) | Registers cannot be used. Specify the following type.<br>• MCTBL: Constants table<br>• IOTBL: I/O conversion table*<br>• ILKTBL: Interlock table*<br>• ASMTBL: Part composition table* |

* I/O conversion tables, interlock tables, and part composition tables are not supported by MPE720 version 6. Use MPE720 version 5 if you have created these types of tables.

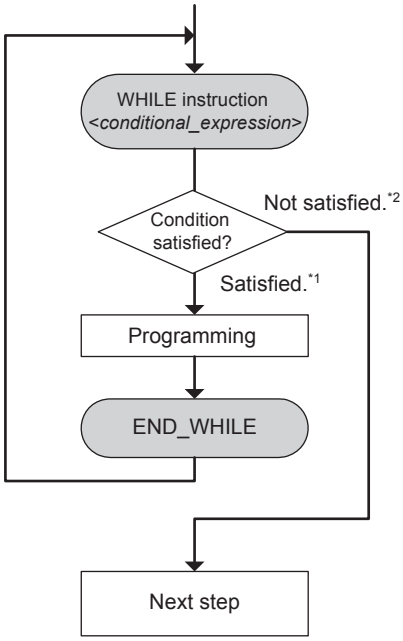
## ( 3 ) Programming Example

This example shows how to call an MCTBL constants table.

# 5.5.7 WHILE Construct (WHILE, END_WHILE)

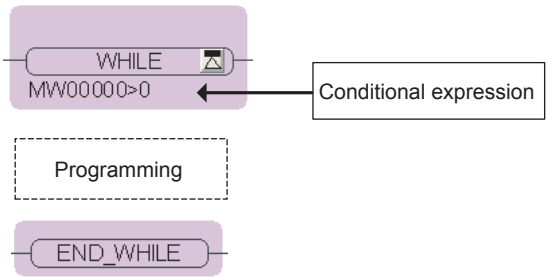
## ( 1 ) Operation

The programming between the WHILE and END_WHILE instructions are executed when the conditional expression for the WHILE instruction is satisfied. After the last line is executed, program execution returns to the WHILE instruction. Execution of the programming is repeated for as long as the conditional expression is satisfied.
If the conditional expression is not satisfied, program execution jumps to the next step following the END_WHILE instruction. None of the programming between the WHILE and END_WHILE instructions is executed.




∗ 1. The programming is executed and then execution returns to the WHILE instruction.
∗ 2. The programming is not executed and execution jumps to the next step.

## ( 2 ) Format




Icon: WHILE END_WHILE

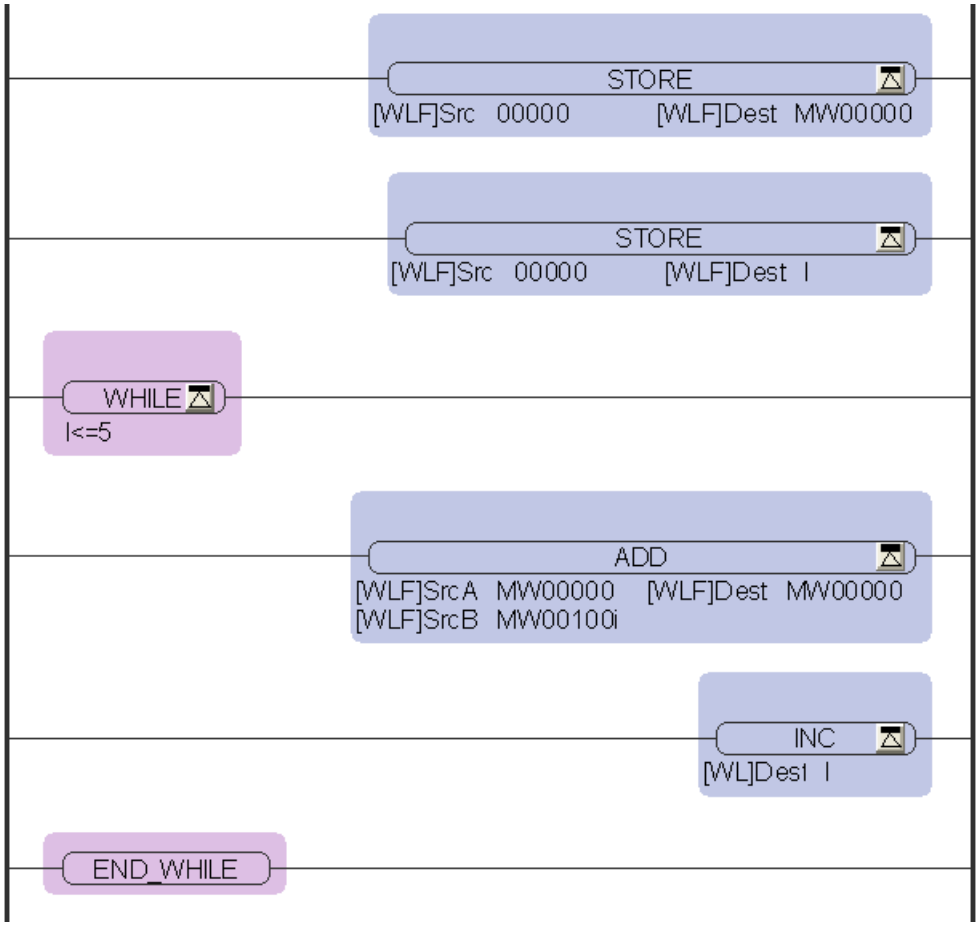
Key entry: WHILE, WEND

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Conditional expression | ○* | ○* | ○* | ○* | × | ○* | ○* |

∗ Write with the format for an EXPRESSION instruction.
Refer to *Chapter D Format for EXPRESSION Instruction* for details on the format used to write the expression.

## ( 3 ) Programming Example

In the following programming example, the registers from MW00100 to MW00105 are added together and stored in the MW00000 register.
The conditional expression is I ≤ 5, so the ADD instruction is executed while I is 0 to 5.
The conditional expression is no longer satisfied when I is 6, so program execution jumps to the next step following the END_WHILE instruction.




IMPORTANT

Execution of the programming is repeated for as long as the conditional expression for the WHILE instruction is satisfied.

If the conditional expression never becomes unsatisfied, or if it takes too much time to become unsatisfied, the Machine Controller system will shut down.

In the example given above, an endless loop would occur if the programming did not include the instruction that increments I.

## ( 4 ) Additional Information

### [ a ] Applicable Conditional Expressions

The conditional expression for a WHILE instruction must be written with the format for an EXPRESSION instruction to produce a Boolean (TRUE or FALSE) result. Numerical expressions that include substitution operators will not be recognized.

| Expression Example | Notation | Remarks |
|---|---|---|
| MB000001 == true | OK | TRUE: 1 (ON) |
| MB000001 != false | OK | FALSE: 0 (OFF) |
| MW00002 < 100 | OK | – |
| MF00002 < sin(60.0) | OK | – |
| MW00001 == 0x00FF OK | OK | Prefix hexadecimal values with 0x. |
| MB000001 = true | NG | – |
| MW00001 = MW00002 | NG | – |

* Refer to *Appendix D Format for EXPRESSION Instruction* for details on applicable instructions, operation order, and notation conventions.

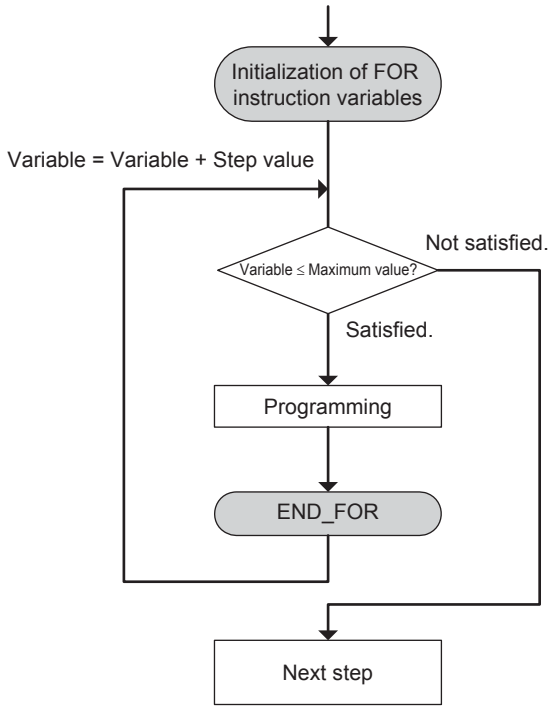
### [ b ] Nesting Depth

The FOR, WHILE, and IF constructs can contain other constructs. This is called nesting. The maximum depth of a nested structure that uses FOR, WHILE, and IF statements is limited to 8 levels.
If an instruction is preceded by a contact, it is treated like an IF construct and is included in the number of nesting levels.

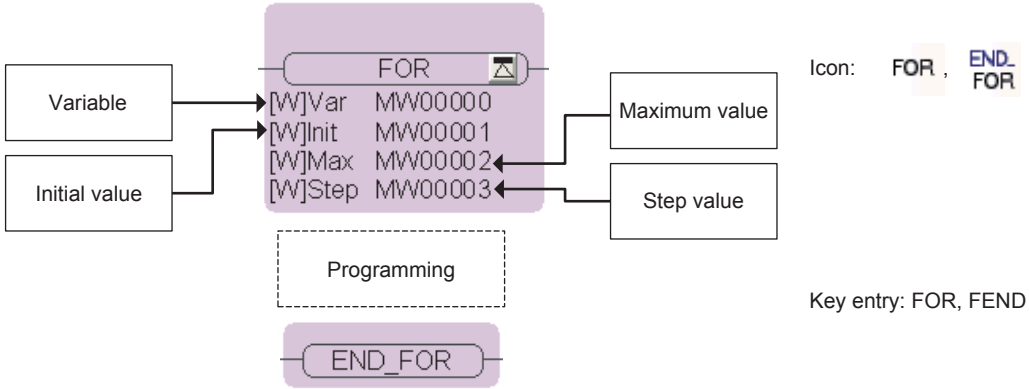
## 5.5.8 FOR Construct (FOR, END_FOR)

### ( 1 ) Operation

The programming between the FOR and END_FOR instructions is repeatedly executed.
The initial value starts with the value in a register specified as the variable. This variable is incremented by the step value each time execution is repeated.
The conditional expression for the FOR instruction is no longer satisfied when the value of the variable exceeds the maximum value, so program execution jumps to the next step.




### ( 2 ) Format

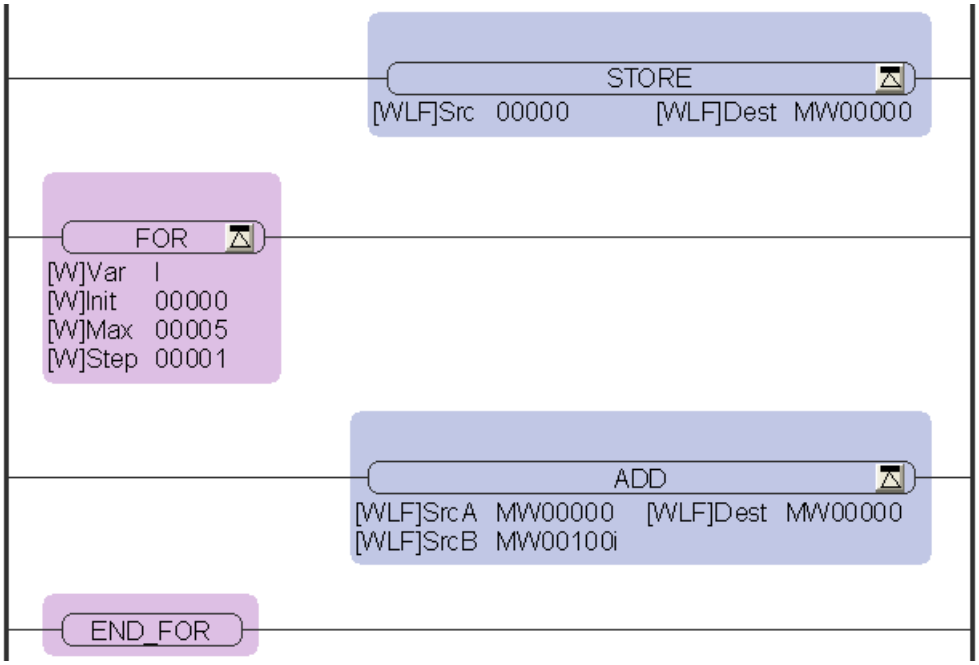



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Variable (Var) | × | ○* | × | × | × | ○ | × |
| Initial value (Init) | × | ○ | × | × | × | ○ | ○ |
| Maximum value (Max) | × | ○ | × | × | × | ○ | ○ |
| Step value (Step) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the registers from MW00100 to MW00105 are added together and stored in the MW00000 register.
In this example, variable I is initialized to 0 by storing 0. Thereafter, the ADD instruction is executed until variable I exceeds the maximum value of 5. The conditional expression is no longer satisfied when I is 6, so program execution jumps to the next step following the END_FOR instruction.



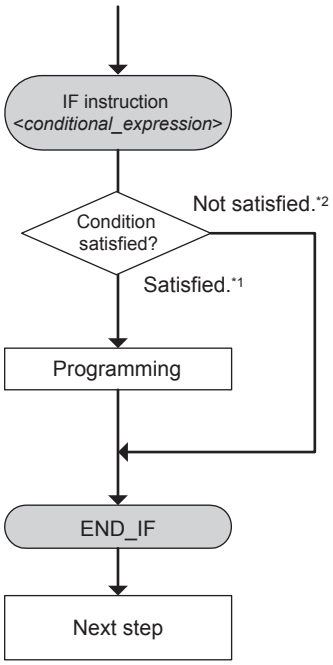

## ( 4 ) Additional Information

The FOR, WHILE, and IF constructs can contain other constructs. This is called nesting. The maximum depth of a nested structure that uses FOR, WHILE, and IF statements is limited to 8 levels.
If an instruction is preceded by a contact, it is treated like an IF construct and is included in the number of nesting levels.

## 5.5.9 IF Construct (IF, END_IF)

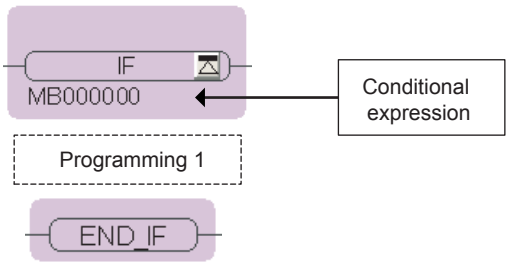
### ( 1 ) Operation

Execution of the programming between the IF and END_IF instructions is repeated for as long as the conditional expression for the IF instruction is satisfied.
The programming is not executed if the conditional expression is not satisfied.




∗ 1. The programming is executed and execution jumps to the next step.
∗ 2. The programming is not executed and execution jumps to the next step.

### ( 2 ) Format




Icon: IF , END_IF

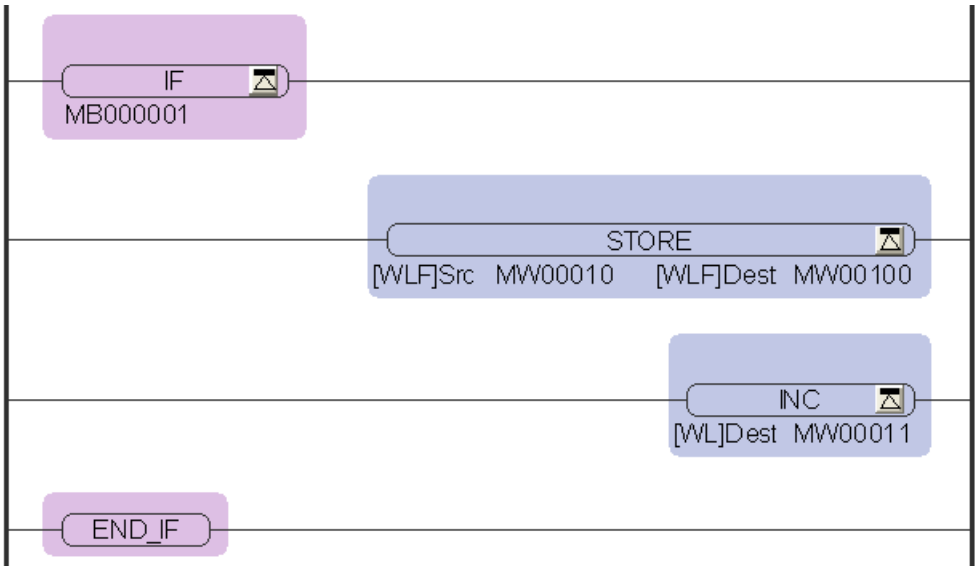
Key entry: IF, IEND

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Conditional expression | ○* | ○* | ○* | ○* | × | ○* | ○* |

∗ Write with the format for an EXPRESSION instruction.
Refer to *Appendix D Format for EXPRESSION Instruction* for details on the format used to write the expression.

## ( 3 ) Programming Example

When the conditional expression (MB000001) for the IF instruction turns ON, the value of MW00010 is set in MW01000 and MW00011 is incremented.




## ( 4 ) Additional Information

### [ a ] Applicable Conditional Expressions

The conditional expression for an IF instruction must be written with the format for an EXPRESSION instruction to produce a Boolean (TRUE or FALSE) result. Numerical expressions that include substitution operators will not be recognized.

| Expression Example | Notation | Remarks |
|---|---|---|
| MB000001 == true | OK | TRUE: 1 (ON) |
| MB000001 != false | OK | FALSE: 0 (OFF) |
| MW00002 < 100 | OK | – |
| MF00002 < sin(60.0) | OK | – |
| MW00001 == 0x00FF OK | OK | Prefix hexadecimal values with 0x. |
| MB000001 = true | NG | – |
| MW00001 = MW00002 | NG | – |

* Refer to *Appendix D Format for EXPRESSION Instruction* for details on applicable instructions, operation order, and notation conventions.

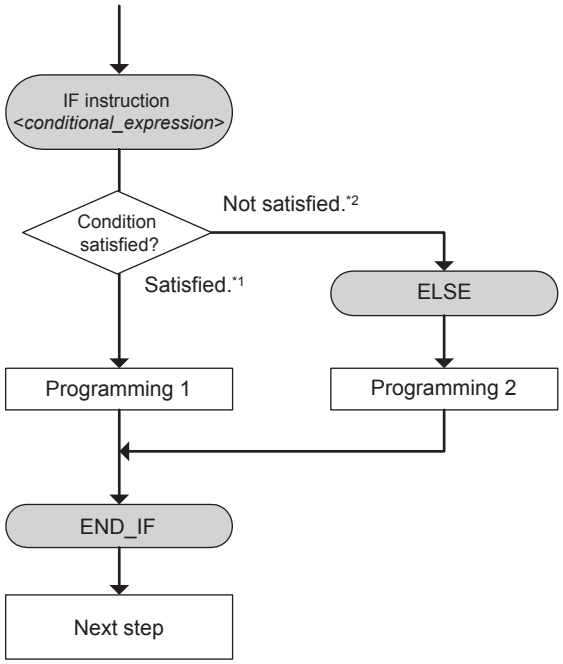
### [ b ] Nesting Depth

The FOR, WHILE, and IF constructs can contain other constructs. This is called nesting. The maximum depth of a nested structure that uses FOR, WHILE, and IF statements is limited to 8 levels.
If an instruction is preceded by a contact, it is treated like an IF construct and is included in the number of nesting levels.

# 5.5.10 IF-ELSE Construct (IF, ELSE, END_IF)

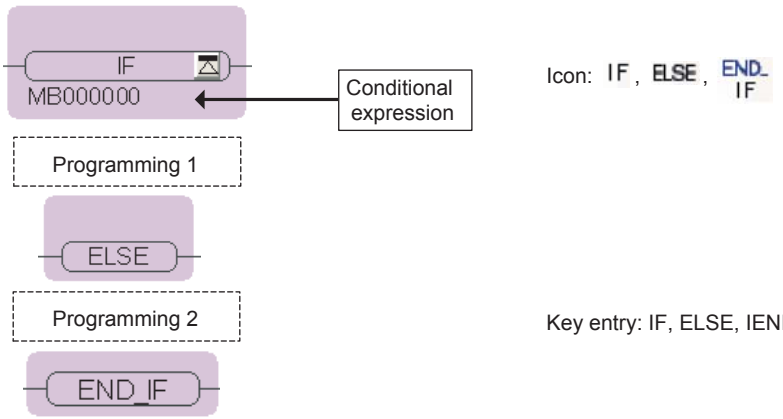
## ( 1 ) Operation

When the conditional expression for the IF instruction is satisfied, only programming 1 is executed. Programming 2 is not executed.
If the conditional expression is not satisfied, only programming 2 is executed. Programming 1 is not executed.




∗ 1. Programming 1 is executed and execution jumps to the next step.
∗ 2. Programming 2 is executed and execution jumps to the next step.

## ( 2 ) Format

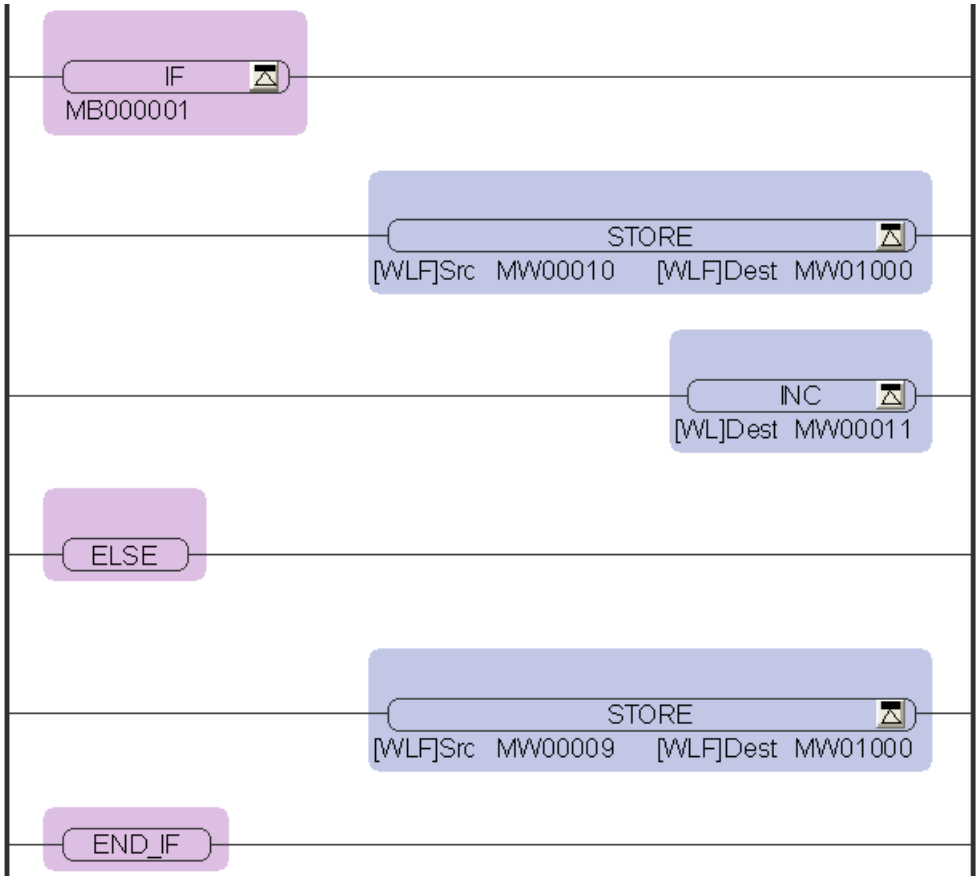



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Conditional expression | ○* | ○* | ○* | ○* | × | ○* | ○* |

∗ Write with the format for an EXPRESSION instruction.
Refer to *Appendix D Format for EXPRESSION Instruction* for details on the format used to write the expression.

## ( 3 ) Programming Example

When the conditional expression (MB000001) for the IF instruction turns ON, the value of MW00010 is set in MW01000 and MW00011 is incremented. When the conditional expression (MB000001) for the IF instruction is OFF, the value of MW00009 is set in MW01000.




## ( 4 ) Additional Information

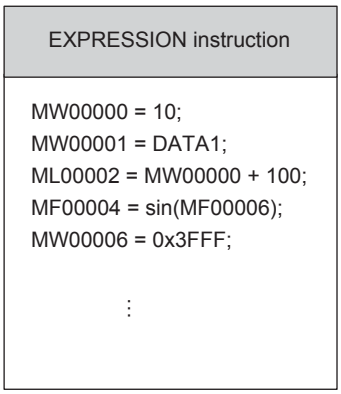
The conditional expressions that can be used, and the nesting depth is the same as for IF constructs.

# 5.5.11 Expression (EXPRESSION)

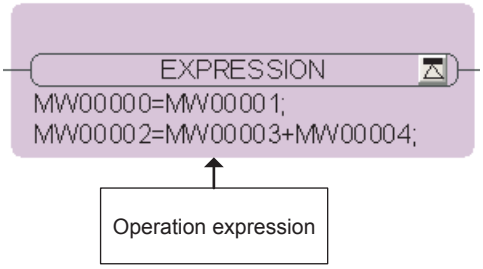
## ( 1 ) Operation

You can use the following elements in an EXPRESSION instruction:

- A variable name or structure in place of a register, similar to C language.
- Basic functions, such as the SIN and COS functions.
- Arithmetic operators, logical operators, comparison operators, and substitution operators.
- Arrays.




## ( 2 ) Format




Icon: Ex press

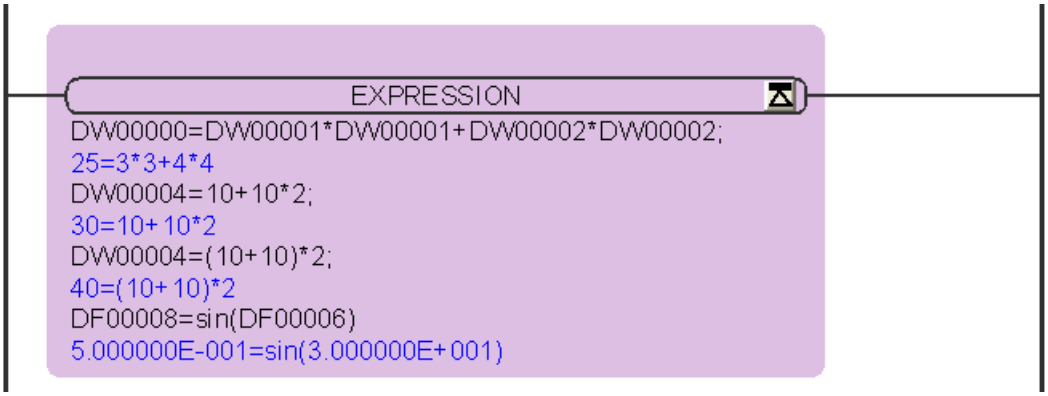
Key entry: EXPR

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Operation expression | ○* | ○* | ○* | ○* | × | ○* | ○* |

* Write with the format for an EXPRESSION instruction.
Refer to *Appendix D Format for EXPRESSION Instruction* for details on the format used to write the expression.

## ( 3 ) Programming Example

In the following programming example, multiple operations are programmed in a single EXPRESSION instruction.




## ( 4 ) Additional Information

The EXPRESSION instruction can be programmed with numeric expressions in addition to expressions that return Boolean TRUE or FALSE values.

| Expression Example | Notation | Remarks |
|---|---|---|
| MB000000 = true; | OK | TRUE: 1 (ON) |
| MW00000 = MW00001+10 | OK | – |
| MW00000 = 0x00FF; | OK | Prefix hexadecimal values with 0x. |
| MB000000 == true; | NG | – |
| MW00001 > MW00000; | NG | – |

- Refer to *Appendix D Format for EXPRESSION Instruction* for details on applicable instructions, operation order, and notation conventions.

# 5.6 Basic Function Instructions

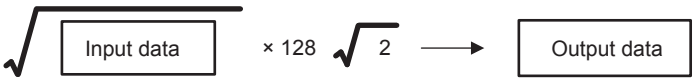
## 5.6.1 Square Root (SQRT)

### ( 1 ) Operation

The SQRT instruction calculates the square root of the integer or real number input data and stores the result in the output data.
Double-length integers cannot be used.

- If the input data is less than 0, the absolute value of the input data will be used to perform the operation and output the result.

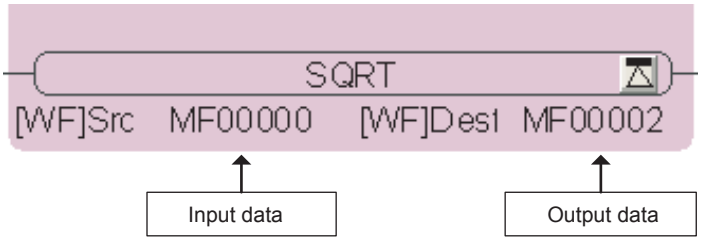
#### [ a ] Integer SQRT: When the Input Data and Output Data Are Integer Data.




- With an integer SQRT instruction, the result is calculated differently from the square root used in mathematics.

#### [ b ] Real Number SQRT: For Any Other Data Types

### ( 2 ) Format




Icon:

Key entry: SQRT

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | × | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

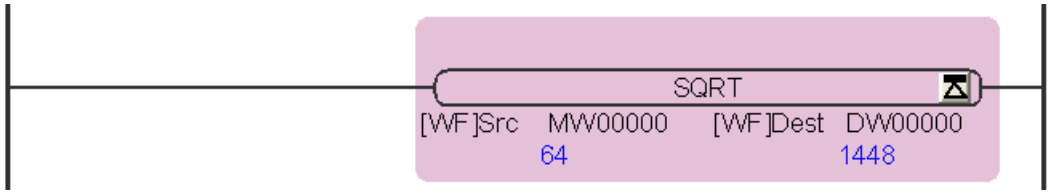
## ( 3 ) Programming Examples

The following programming examples demonstrate the SQRT instruction using integer and real number input data.

- Integer SQRT

  The square root of 64, an integer in the input data in MW00000, is multiplied by 128 $\sqrt{2}$ and the result is stored in the output data in DW00000.

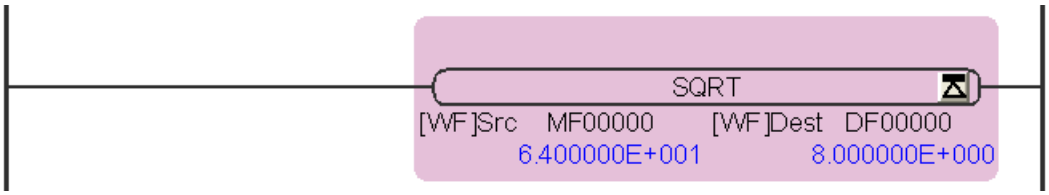
  $\sqrt{64} \times 128\ \sqrt{2} \rightarrow$ DW00000 = 1,448




- Real Number SQRT

  The square root of 64.0, a real number in the input data in MF00000, is calculated and the result is stored in the output data in DF00000.

  $\sqrt{64.0} \rightarrow$ DF00000 = 8.0



## 5.6.2 Sine (SIN)

### ( 1 ) Operation

The SIN instruction calculates the sine of the integer or real number input data and stores the result in the output data. Double-length integers cannot be used.

#### [ a ] Integer Input Data and Output Data

SIN ( Input data ) × 10000 ⟶ Output data

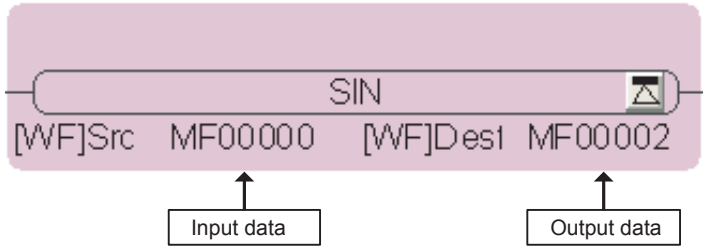
∗ 1. The input data is in degrees, where 1 = 0.01 degree.
∗ 2. The operation result is multiplied by 10,000 and stored in the output data.

#### [ b ] Real Number Input Data and Output Data

SIN ( Input data ) ⟶ Output data

- The input data is in degrees.

### ( 2 ) Format




Icon: sin

Key entry: SIN

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | × | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | × | ○* | × | ○ | × |

∗ C and # registers cannot be used.

#### [ a ] Integer

The input data is in degrees, where 1 = 0.01 degrees.
Therefore, the SIN function can operate on values between -327.78 and 327.67 degrees.
The output of the SIN function is multiplied by 10,000, so the output data will be output between -10,000 and 10,000.

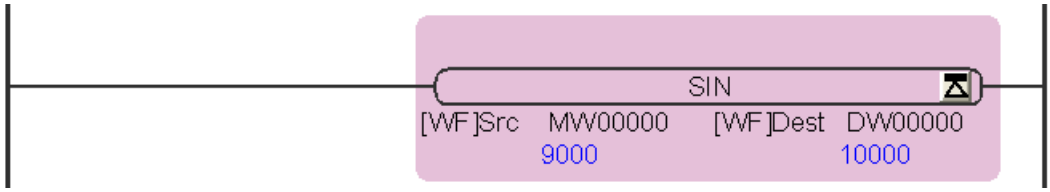
#### [ b ] Real Number

The input data is in degrees.

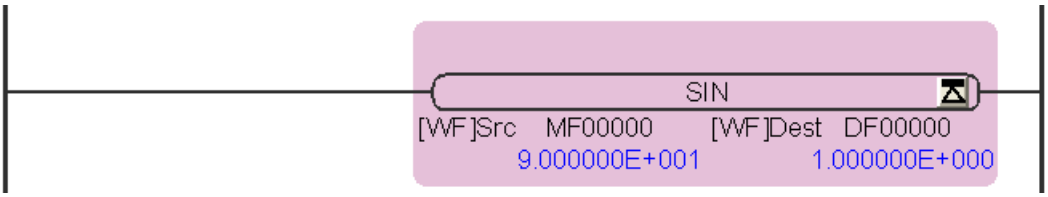
## ( 3 ) Programming Examples

The following programming examples demonstrate the SIN instruction using integer and real number input data.

- Integer SIN
  The sine of 9,000, an integer in the input data in MW00000, is calculated and the result is stored in the output data in DW00000.
  SIN (90.00 deg) × 10,000 → DW00000 = 10,000




- Real Number SIN
  The sine of 90.0, a real number in the input data in MF00000, is calculated and the result is stored in the output data in DF00000.
  SIN (90.0 deg) → DF00000 = 1.0



# 5.6.3 Cosine (COS)

## ( 1 ) Operation

The COS instruction calculates the cosine of the integer or real number input data and stores the result in the output data.
Double-length integers cannot be used.

### [ a ] Integer Input Data and Output Data

COS ( Input data ) × 10000 ⟶ Output data

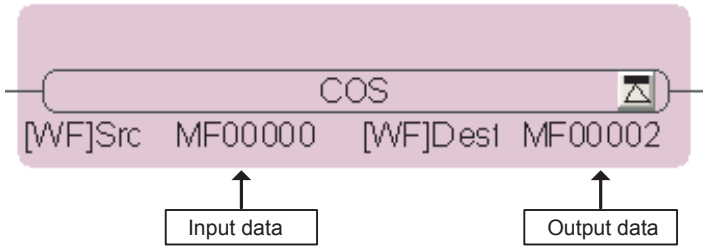
∗ 1. The input data is in degrees, where 1 = 0.01 degree.
∗ 2. The operation result is multiplied by 10,000 and stored in the output data.
∗ 3. The input data must be between -327.68 and 32.767 degrees. Any other number will not produce the correct result.

### [ b ] Real Number Input Data and Output Data

COS ( Input data ) ⟶ Output data

- The input data is in degrees.

## ( 2 ) Format




Icon: COS

Key entry: COS

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | ○ | × | ○ | × | ○ | ○ |
| Output data (Dest) | × | ○* | × | ○* | × | ○ | × |

∗ C and # registers cannot be used.

### [ a ] Integer

The input data is in degrees, where 1 = 0.01 degree.
Therefore, the COS function instruction can operate on values between -327.78 and 327.67 degrees.
The output of the COS function is multiplied by 10,000, so the data will be output between -10,000 and 10,000.

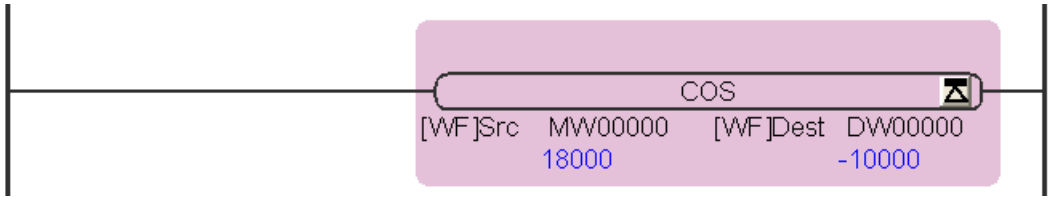
### [ b ] Real Number

The input data is in degrees.

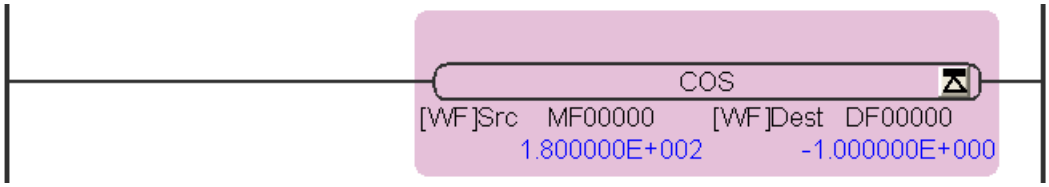
## ( 3 ) Programming Examples

The following programming examples demonstrate the COS instruction using integer and real number input data.

- Integer COS
  The cosine of 18,000, an integer in the input data in MW00000, is calculated and the result is stored in the output data in DW00000.
  COS (180.00 deg) × 10,000 → DW00000 = -10,000




- Real Number COS
  The cosine of 180.0, a real number in the input data in MF00000, is calculated and the result is stored in the output data in DF00000.
  COS (180.0 deg) → DF00000 = -1.0

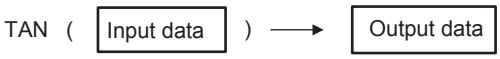


## 5.6.4 Tangent (TAN)

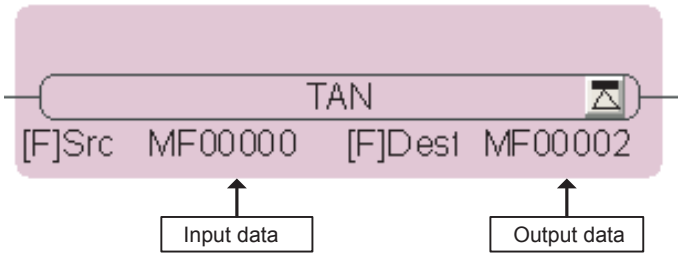
### ( 1 ) Operation

The TAN instruction calculates the tangent of the real number input data and stores the result in the output data.

- The input data is in degrees.




### ( 2 ) Format




Icon: tan

Key entry: TAN

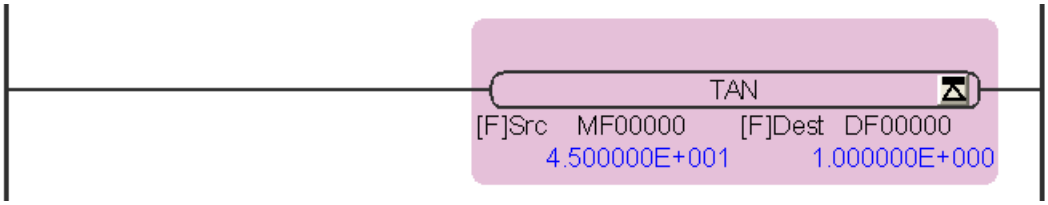
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | O | × | × | O |
| Output data (Dest) | × | × | × | O* | × | × | × |

* C and # registers cannot be used.

### ( 3 ) Programming Example

In the following programming example, the tangent of 45 in the input data in MW00000 is calculated and the result is stored in the output data in DF00000.

TAN (45.00 deg) → DF00000 = 1.0

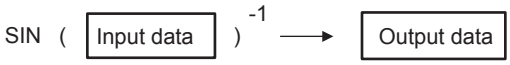


# 5.6.5 Arc Sine (ASIN)

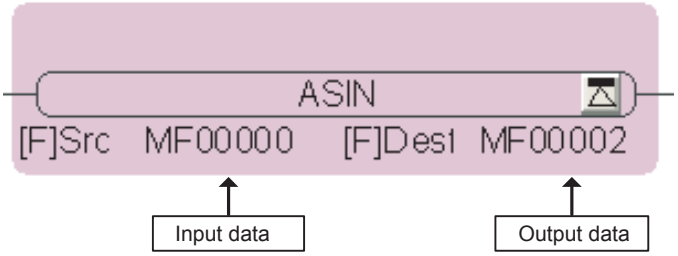
## ( 1 ) Operation

The ASIN instruction calculates the arc sine of the real number input data and stores the result in the output data.

- The output data is in degrees.




## ( 2 ) Format




Icon: sin-1

Key entry: ASIN

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | ○ | × | × | ○ |
| Output data (Dest) | × | × | × | ○* | × | × | × |

* C and # registers cannot be used.

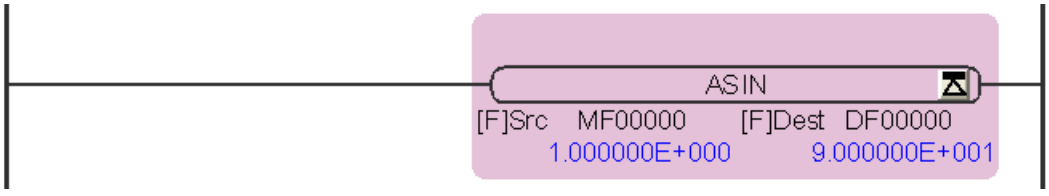
### ■ Input Data Range

Set the input data to a value between -1.0 and 1.0. The output is set to 0 if the input value is out of range.

## ( 3 ) Programming Example

In the following programming example, the arc sine of 1.0 in the input data in MF00000 is calculated and the result is stored in the output data in DF00000.

SIN $(1.0)^{-1}$ → DF00000 = 90.0 (degrees)

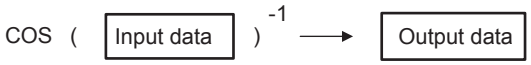


# 5.6.6 Arc Cosine (ACOS)

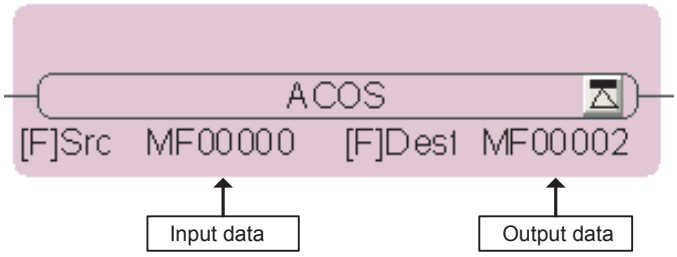
## ( 1 ) Operation

The ACOS instruction calculates the arc cosine of the real number input data and stores the result in the output data.

- The output data is in degrees.




## ( 2 ) Format




Icon:

Key entry: ACOS

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | O | × | × | O |
| Output data (Dest) | × | × | × | O* | × | × | × |

* C and # registers cannot be used.

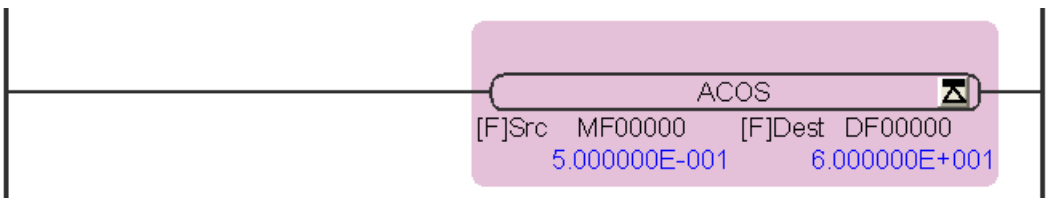
### ■ Input Data Range

Set the input data to a value between -1.0 and 1.0. The output is set to 0 if the input value is out of range.

## ( 3 ) Programming Example

In the following programming example, the arc sine of 0.5 in the input data in MF00000 is calculated and the result is stored in the output data in DF00000.

$\cos(0.5)^{-1} \rightarrow$ DF00000 = 60.0 (degrees)

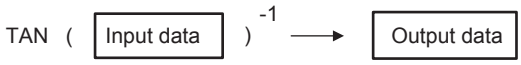


# 5.6.7 Arc Tangent (ATAN)

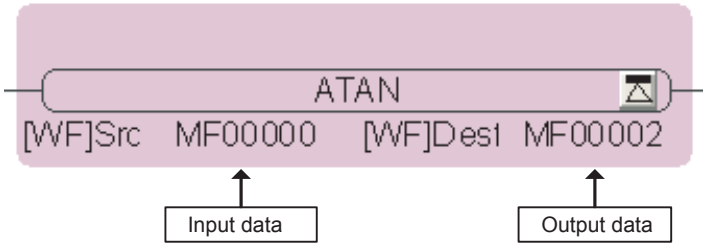
## ( 1 ) Operation

The ATAN instruction calculates the arc tangent of the real number input data and stores the result in the output data.

- The output data is in degrees.




## ( 2 ) Format




Icon: tan$^{-1}$

Key entry: ATAN

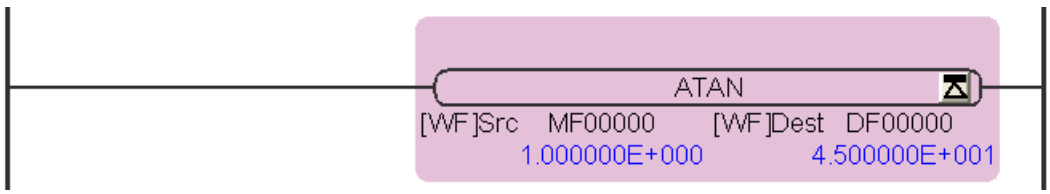
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | ○ | × | × | ○ |
| Output data (Dest) | × | × | × | ○* | × | × | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the arc tangent of 1.0 in the input data in MF00000 is calculated and the result is stored in the output data in DF00000.

TAN $(1.0)^{-1}$ → DF00000 = 45.0 (degrees)



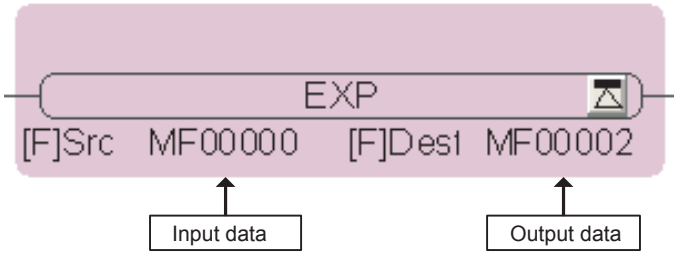
## 5.6.8 Exponential (EXP)

### ( 1 ) Operation

The EXP instruction calculates the value obtained by raising base *e* of the natural logarithm to the real number input data and stores the result in the output data.

- "e" is the base of the natural logarithm.

### ( 2 ) Format




Icon: $X^y$

Key entry: EXP

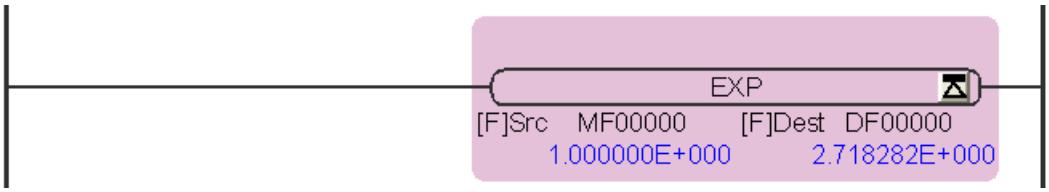
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | ○ | × | × | ○ |
| Output data (Dest) | × | × | × | ○* | × | × | × |

* C and # registers cannot be used.

### ( 3 ) Programming Example

The following programming example calculates base *e* of the natural logarithm raised to 1.0 in the input data in MF00000, and stores the result in the output data in DF00000.

$e^{1.0} \rightarrow$ DF00000 = 2.718282




INFO

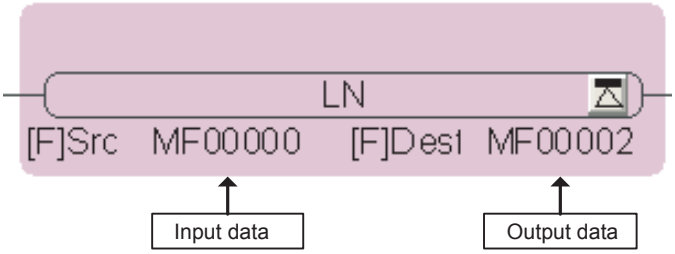
If the operation result overflows, the output data will be set to the maximum value 3.402E+38 and an operation error will not occur.

# 5.6.9 Natural Logarithm (LN)

## ( 1 ) Operation

The LN instruction calculates the natural logarithm of X ($\log_e X$), when input data X is a real number and stores the result in the output data.

## ( 2 ) Format




Icon: ln

Key entry: LN

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | ○ | × | × | ○ |
| Output data (Dest) | × | × | × | ○* | × | × | × |

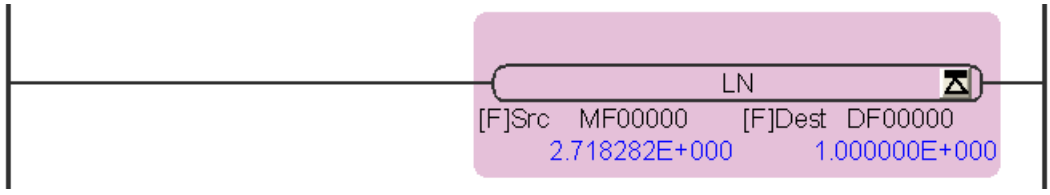
* C and # registers cannot be used.

### ■ When Input Data Is Less Than or Equal to 0

If the input data is less than 0, the absolute value of the input data will be used to perform the operation and output the result.
The output data is set to -∞ if the input value is 0.

## ( 3 ) Programming Example

The following programming example calculates the natural logarithm when the input data is 2.718282 (≈ e) in MF00000, and stores the result in the output data in DF00000.
$\text{Log}_e$ 2.718282 ≈ $\log_e$ e → DF00000 = 1.0

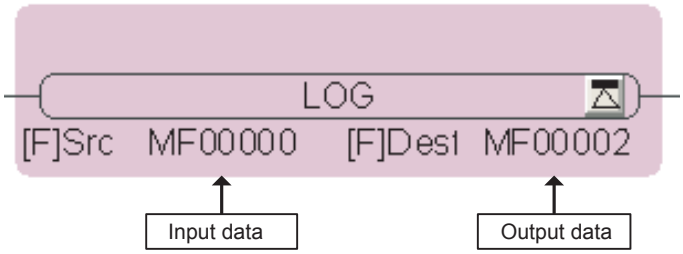


# 5.6.10 Common Logarithm (LOG)

## ( 1 ) Operation

The LOG instruction calculates the common logarithm of X ($\log_{10}$ X), when input data X is a real number and stores the result in the output data.

## ( 2 ) Format




Icon: log

Key entry: LOG

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input data (Src) | × | × | × | O | × | × | O |
| Output data (Dest) | × | × | × | O* | × | × | × |

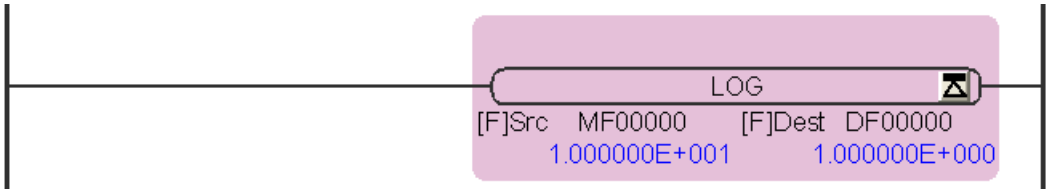
* C and # registers cannot be used.

### ■ When Input Data Is Less Than or Equal to 0

If the input data is less than 0, the absolute value of the input data will be used to perform the operation and output the result.
The output data is set to -∞ if the input value is 0.

## ( 3 ) Programming Example

The following programming example calculates the common logarithm when the input data is 10.0 in MF00000, and stores the result in the output data in DF00000.
$\log_{10}$ 10.0 → DF00000 = 1.0

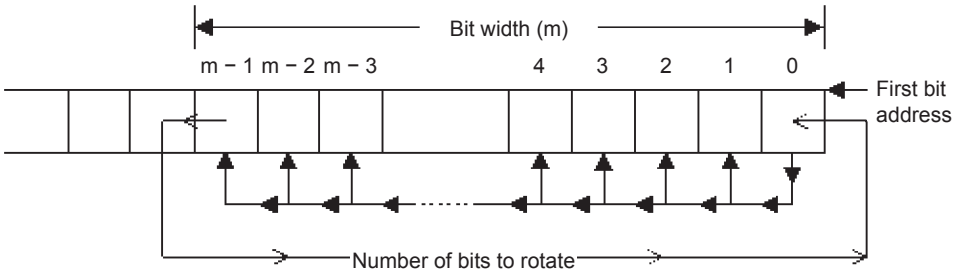


# 5.7 Data Shift Instructions

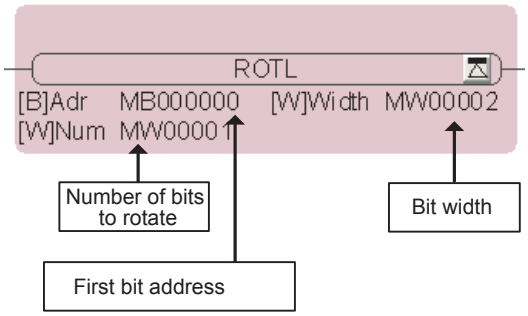
## 5.7.1 Bit Rotate Left (ROTL)

### ( 1 ) Operation

The ROTL instruction rotates the data specified by the first bit address and bit width to the left by the specified number of bits.




### ( 2 ) Format




Icon: ROT L

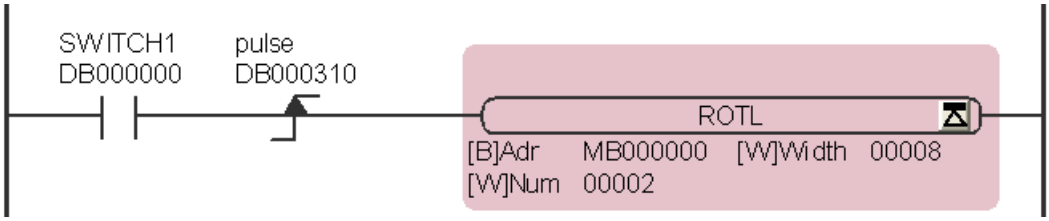
Key entry: ROTL

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First bit address (Adr) | ○* | × | × | × | × | × | × |
| Number of bits to rotate (Num) | × | ○ | × | × | × | ○ | ○ |
| Bit width (Width) | × | ○ | × | × | × | ○ | ○ |

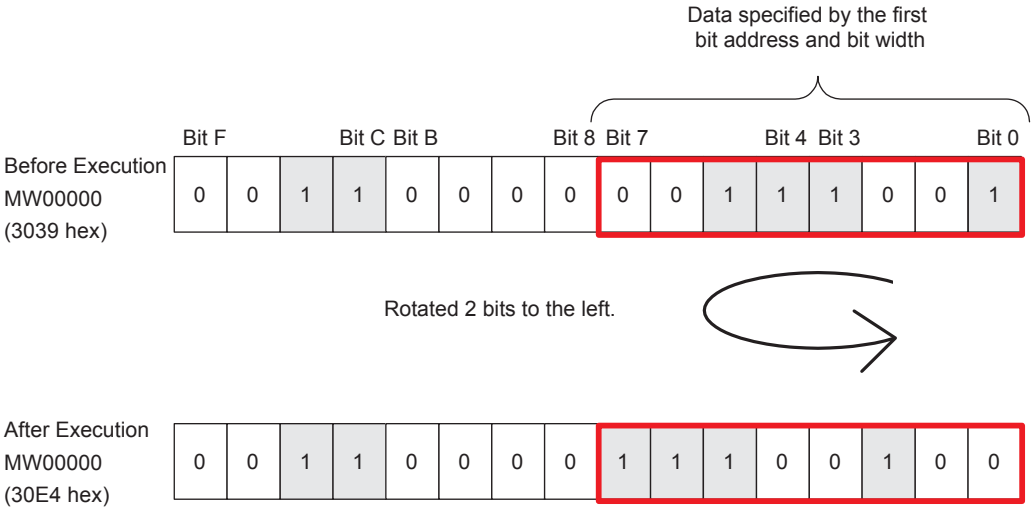
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the data specified as 8 bits wide from the first bit address at MB000000 is rotated two bits to the left.
The ROTL instruction is executed when switch 1 (DB000000) turns ON.



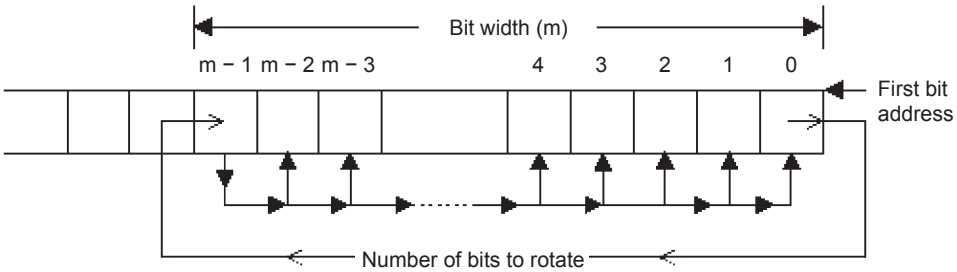

The following figure shows the operation when MW00000 is 12,345 (3039 hex).



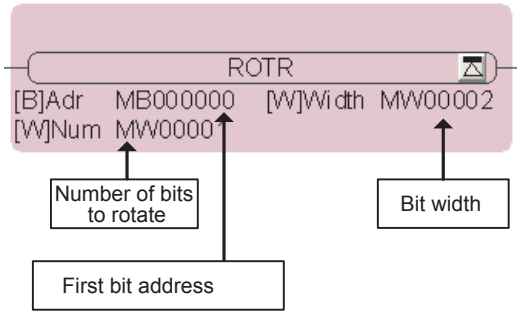
# 5.7.2 Bit Rotate Right (ROTR)

## ( 1 ) Operation

The ROTR instruction rotates the data specified by the first bit address and bit width to the right by the specified number of bits.




## ( 2 ) Format




Icon:

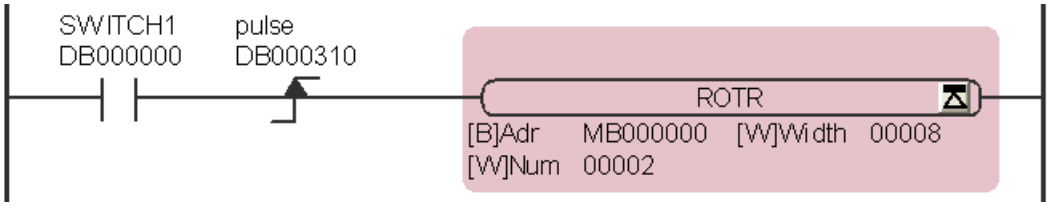

Key entry: ROTR

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First bit address (Adr) | ○* | × | × | × | × | × | × |
| Number of bits to rotate (Num) | × | ○ | × | × | × | ○ | ○ |
| Bit width (Width) | × | ○ | × | × | × | ○ | ○ |

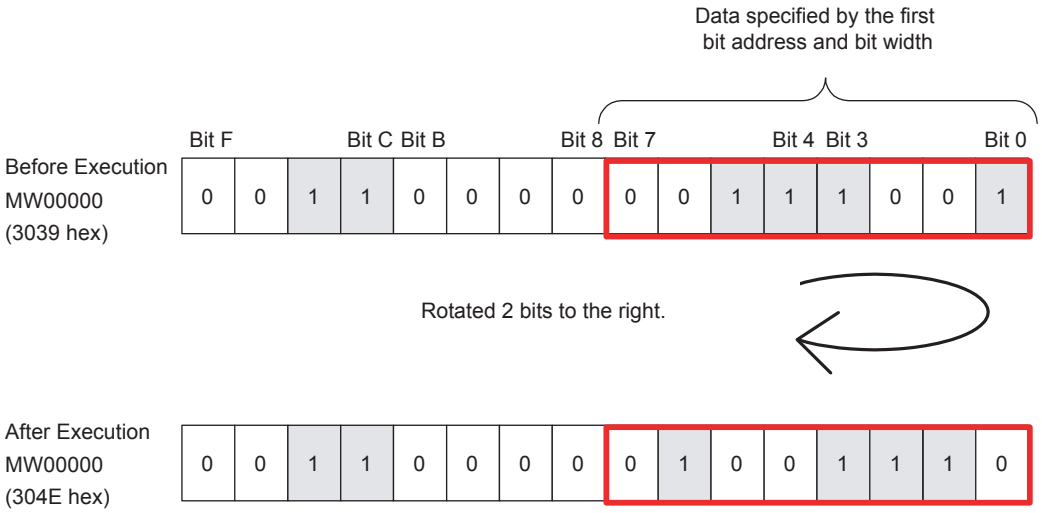
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the data specified as 8 bits wide from the first bit address at MB000000 is rotated two bits to the right.
The ROTR instruction is executed when switch 1 (DB000000) turns ON.



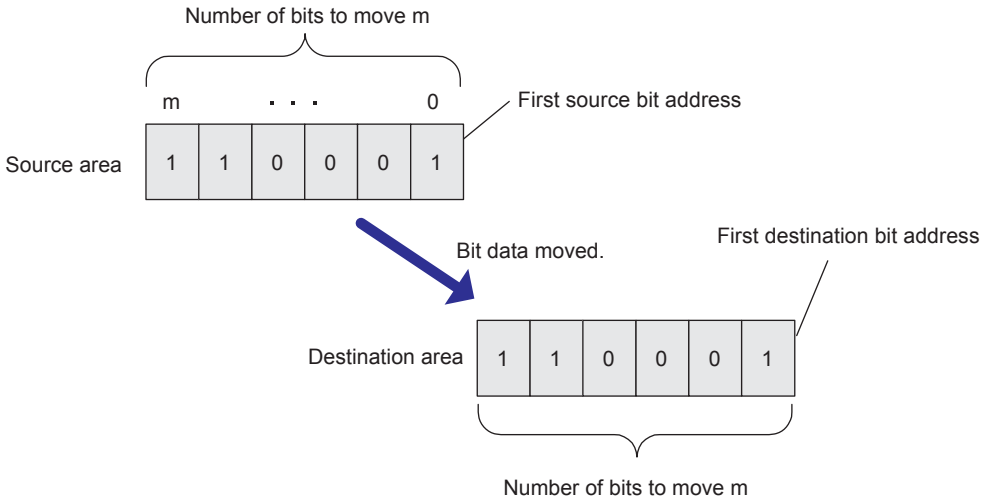

The following figure shows the operation when MW00000 is 12,345 (3039 hex).



# 5.7.3 Move Bit (MOVB)

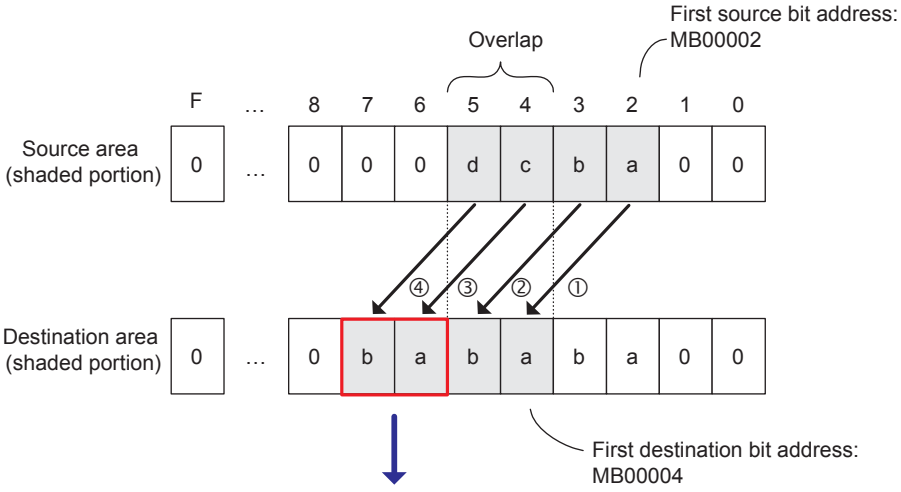
## ( 1 ) Operation

The MOVB instruction moves the designated number of bits of data from the area that starts with the first source bit address to the area that starts with the first destination bit address.



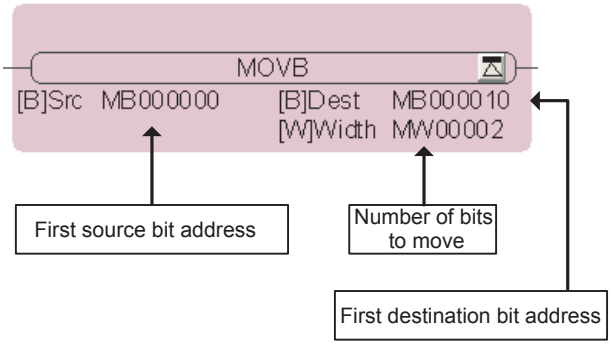

- The bits are moved one bit at a time from the lowest relay address.
  If the source area and destination area overlap, the source data that is actually moved may not be the data that was in the source area before the instruction was executed.

### ■ Example Where the Source Area and Destination Area Overlap




Bit status is moved in the following order: ① to ④. This means that the status of bits 2 and 3 are moved to bits 4 and 5 (① and ②) and then the status of bits 4 and 5 are moved (③ and ④).

## ( 2 ) Format




Icon: MOV B

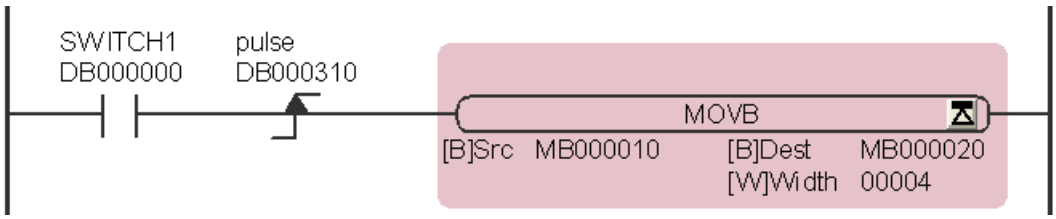
Key entry: MOVB

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source bit address (Src) | ○ | × | × | × | × | × | × |
| First destination bit address (Dest) | ○* | × | × | × | × | × | × |
| Number of bits to move | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four bits of data are moved from the area that starts with the first source bit address at MB000010 to the area that starts with the first destination bit address at MB000020.
The MOVB instruction is executed when switch 1 (DB000000) turns ON.



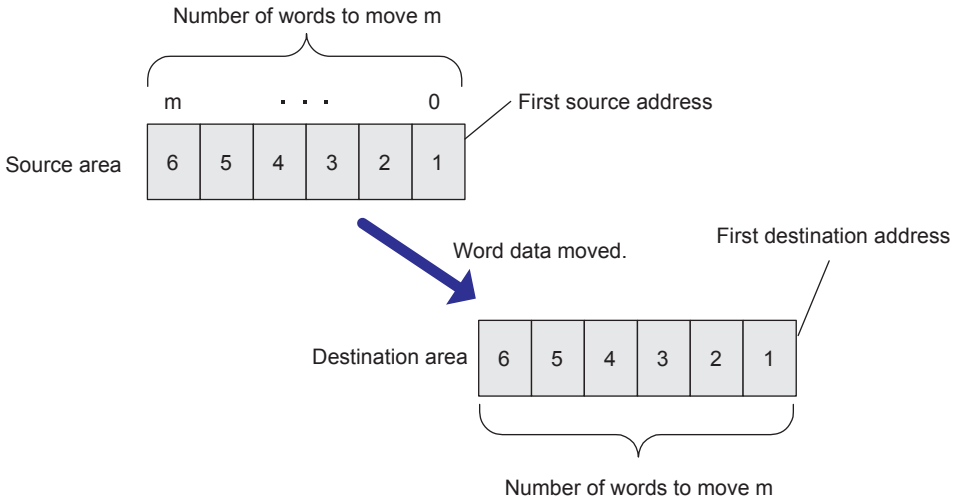

The following table illustrates how the data in the source area is moved to the destination area.

| Source area | | ⇒ | Destination area | | |
|---|---|---|---|---|---|
| Register | Data | | Register | Data before Execution of Instruction | Data after Execution of Instruction |
| MB000010 | 0 | | MB000020 | 0 | 0 |
| MB000011 | 1 | | MB000021 | 0 | 1 |
| MB000012 | 1 | | MB000022 | 0 | 1 |
| MB000013 | 1 | | MB000023 | 0 | 1 |

# 5.7.4 Move Word (MOVW)

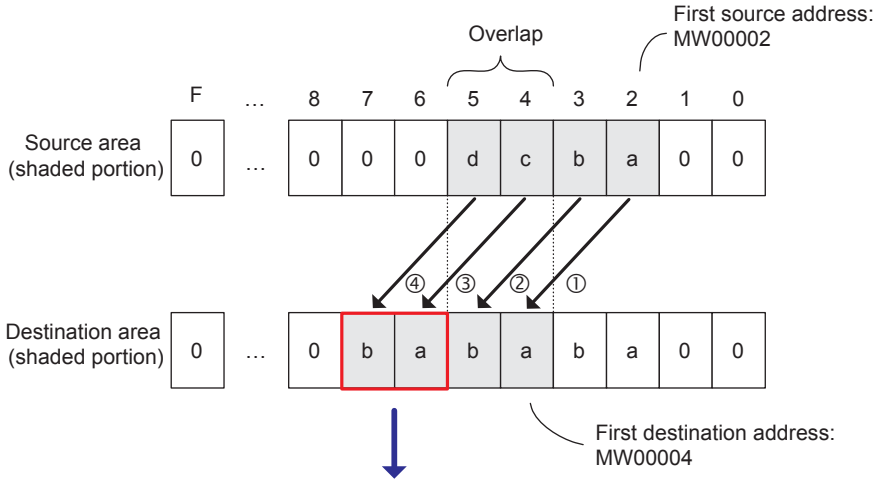
## ( 1 ) Operation

The MOVW instruction moves the specified number of words from the area that starts with the first source address to the area that starts with the first destination address.



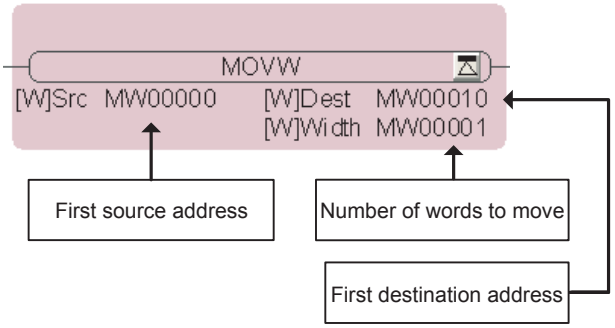

- The words are moved one at a time from the lowest register address.
  If the source area and destination area overlap, the source data that is actually moved may not be the data that was in the source area before the instruction was executed.

### ■ Example Where the Source Area and Destination Area Overlap




Word contents are moved in the following order: ① to ④. This means that the contents of MW00002 and MW00003 are moved to MW00004 and MW00005 (① and ②) and then the contents of MW00004 and MW00005 are moved (③ and ④).

## ( 2 ) Format




Icon: MOV W

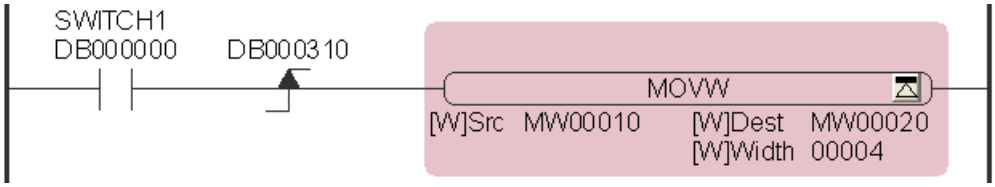
Key entry: MOVW

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source address (Src) | × | ○ | × | × | × | × | × |
| First destination address (Dest) | × | ○* | × | × | × | × | × |
| Number of words to move (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four words of data from the area that starts with the first source address at MW00010 are moved to the area that starts with the first destination address at MW00020.
The MOVW instruction is executed when switch 1 (DB000000) turns ON.



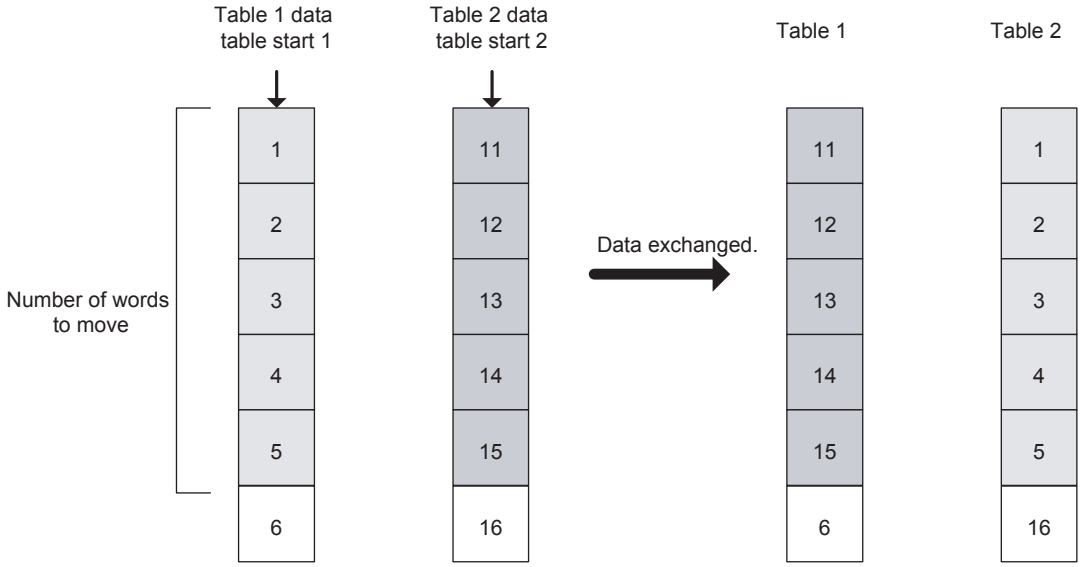

The following table illustrates how the data in the source area is moved to the destination area.

| Source area | | | Destination area | | |
|---|---|---|---|---|---|
| Register | Data | | Register | Data before Execution of Instruction | Data after Execution of Instruction |
| MW00010 | 10 | ⇒ | MW00020 | 0 | 10 |
| MW00011 | 20 | | MW00021 | 0 | 20 |
| MW00012 | 30 | | MW00022 | 0 | 30 |
| MW00013 | 40 | | MW00023 | 0 | 40 |

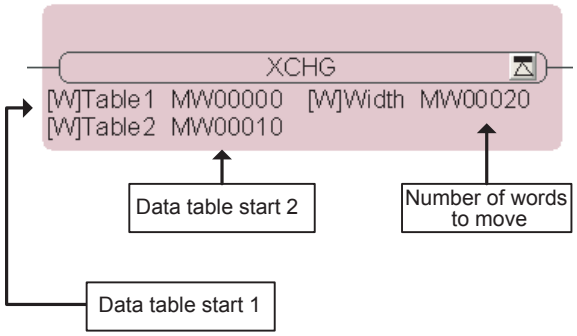
# 5.7.5 Exchange (XCHG)

## ( 1 ) Operation

The XCHG instruction exchanges the designated number of words to move between table 1 and table 2.
The data contents of table 1 and table 2 specified by data table start 1, data table start 2, and the number of words to move are exchanged.




## ( 2 ) Format




Icon: XCHG

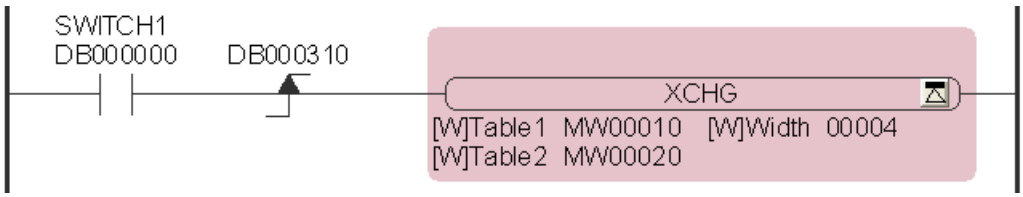
Key entry: XCHG

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Data table start 1 (Table1) | × | ○* | × | × | × | × | × |
| Data table start 2 (Table2) | × | ○* | × | × | × | × | × |
| Number of words to move (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four words of data are exchanged between table 1, which starts at MW00010, and table 2, which starts at MW00020.
The XCHG instruction is executed when switch 1 (DB000000) turns ON.




The following table illustrates how the data is exchanged between table 1 and table 2.

Table 1

| Register | Data before Execution of Instruction | Data after Execution of Instruction |
|---|---|---|
| MW00010 | 10 | 123 |
| MW00011 | 20 | 234 |
| MW00012 | 30 | 345 |
| MW00013 | 40 | 456 |

⇔

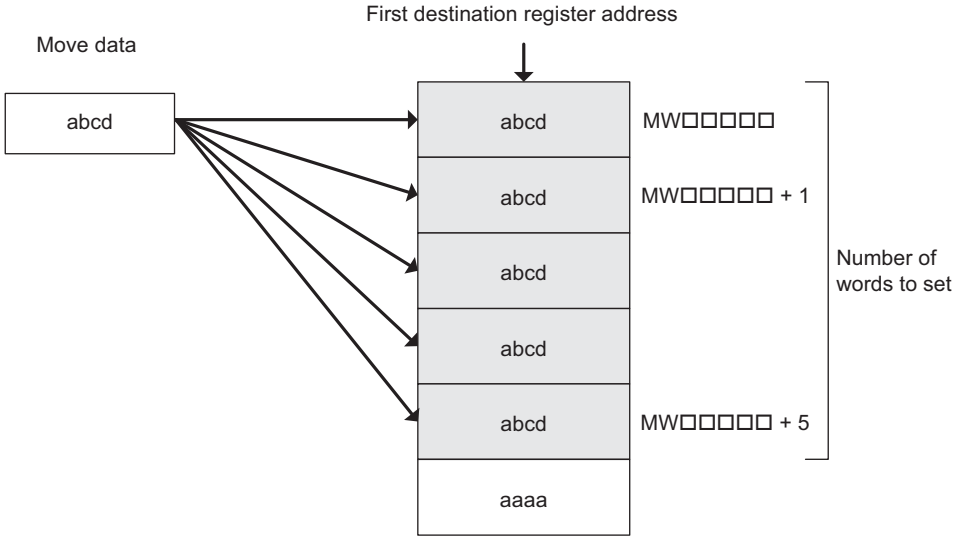
Table 2

| Register | Data before Execution of Instruction | Data after Execution of Instruction |
|---|---|---|
| MW00020 | 123 | 10 |
| MW00021 | 234 | 20 |
| MW00022 | 345 | 30 |
| MW00023 | 456 | 40 |

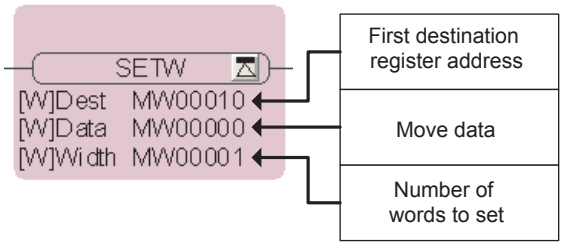
# 5.7.6 Table Initialization (SETW)

## ( 1 ) Operation

The SETW instruction stores the data designated by the move data in all registers in the area that starts from the first destination register address for the number of words to set. The data is stored one word at a time from the lowest register address to the highest.
The data is stored in order from the lowest register address to the highest.




## ( 2 ) Format




Icon: SET W

Key entry: SETW

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First destination register address (Dest) | × | ○* | × | × | × | × | × |
| Move data (Data) | × | ○ | × | × | × | ○ | ○ |
| Number of words to set (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the area of 1,000 words from MW00000 is initialized to the move data (0) on the first scan of the high-speed scan after the power is turned ON.

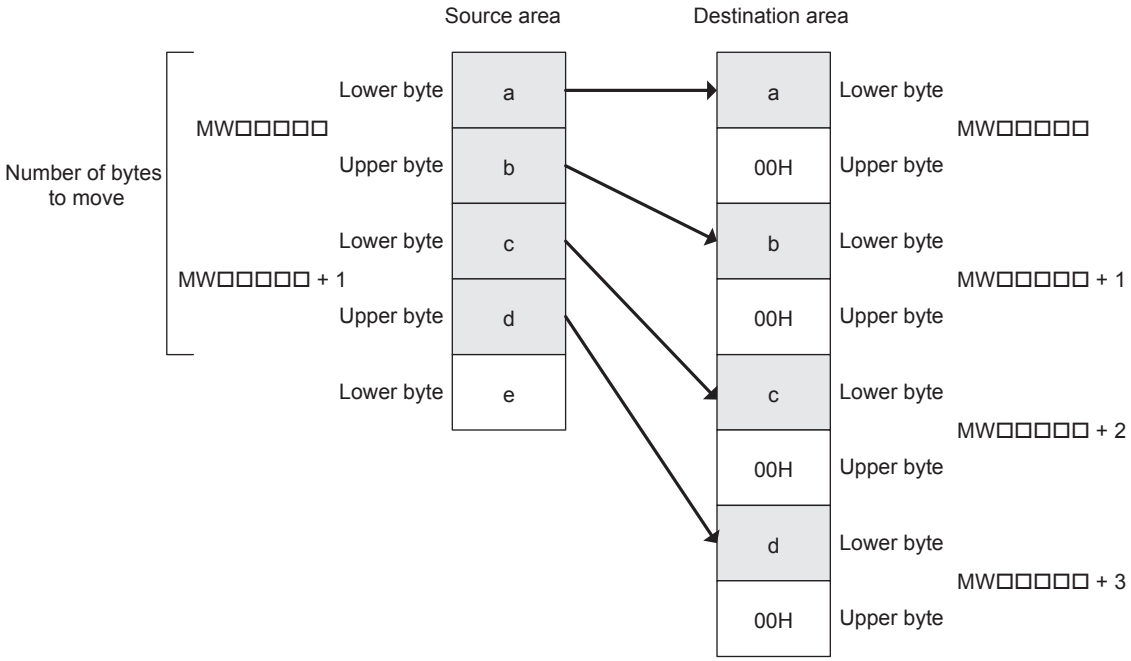
The following table illustrates how the registers are initialized to 0 after execution of the first scan of the high-speed scan when the power is turned ON.

| Register | Data |
|---|---|
| MW00000 | 0 |
| MW00001 | 0 |
| : | : |
| MW00998 | 0 |
| MW00999 | 0 |

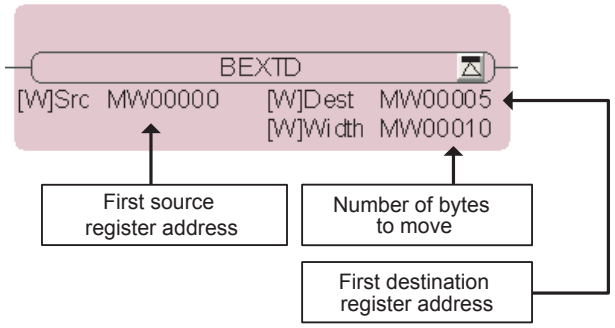
# 5.7.7 Byte-to-word Expansion (BEXTD)

## ( 1 ) Operation

The BEXTD instruction expands the byte data from an area designated by the number of bytes to move from the first source register address into individual word data, one byte at a time, and moves the word data to the area that starts with the first destination register address. When the byte is expanded into a word, the upper byte of the word is set to 0. The byte data from an area designated by the number of bytes to move from the first source register address is expanded into individual word data and moved to the area that starts with the first destination register address.




## ( 2 ) Format




Icon: B EXTD

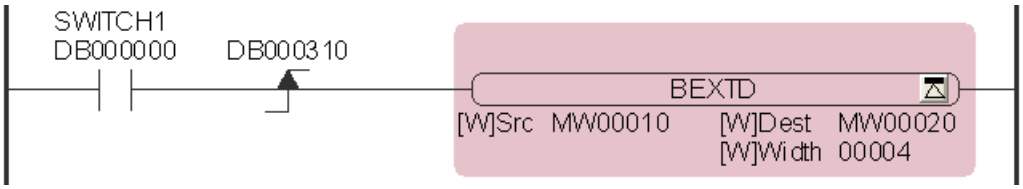
Key entry: BEXTD

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source register address (Src) | × | ○ | × | × | × | × | × |
| First destination register address (Dest) | × | ○* | × | × | × | × | × |
| Number of bytes to move (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four bytes of data are expanded and moved from the area that starts with the first source register address at MW00010 to the area of words that starts with the first destination register address at MW00020.
The BEXTD instruction is executed when switch 1 (DB000000) turns ON.




The following table illustrates how the byte data in the source area is expanded and moved into word data in the destination area.

| Source area | | |
|---|---|---|
| Register | | Data |
| MW00010 | Lower byte | 10 hex |
| | Upper byte | 20 hex |
| MW00011 | Lower byte | 30 hex |
| | Upper byte | 40 hex |

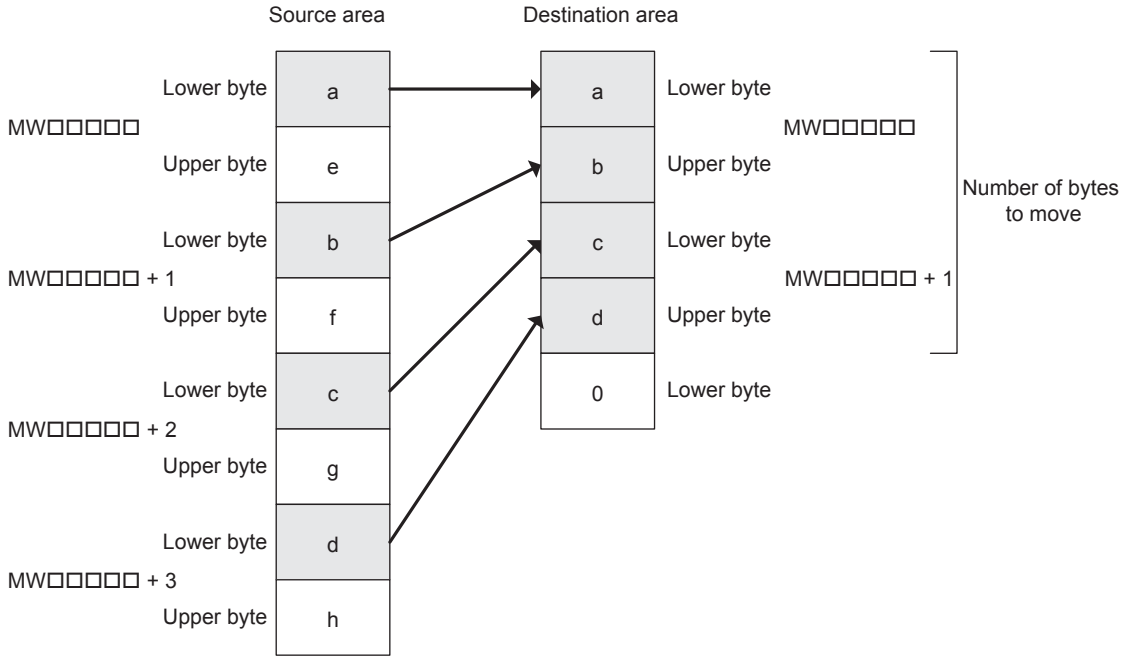
⇒

| Destination area | | |
|---|---|---|
| Register | | Data |
| MW00020 | Lower byte | 10 hex |
| | Upper byte | 00 hex |
| MW00021 | Lower byte | 20 hex |
| | Upper byte | 00 hex |
| MW00022 | Lower byte | 30 hex |
| | Upper byte | 00 hex |
| MW00023 | Lower byte | 40 hex |
| | Upper byte | 00 hex |

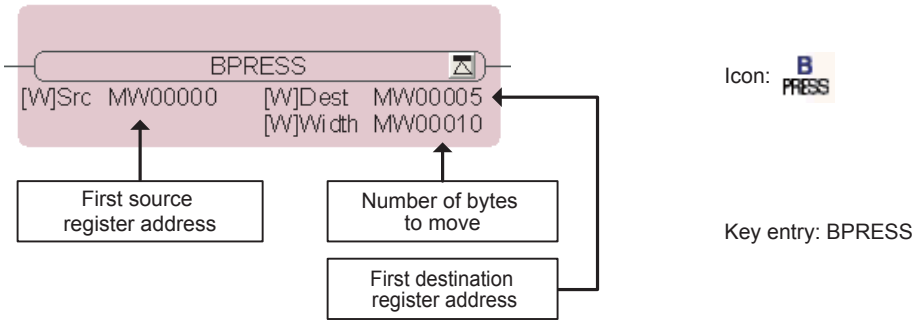
# 5.7.8 Word-to-byte Compression (BPRESS)

## ( 1 ) Operation

The BPRESS instruction stores the lower bytes of word data for the designated number of bytes to move starting from the first source register address, in the area that starts from the first destination register address, one byte at a time. This instruction performs the opposite operation of the BEXTD instruction.
The word data designated by the number of bytes to move is moved from the first source register address to the area that starts with the first destination register address.
The upper byte is discarded.




## ( 2 ) Format

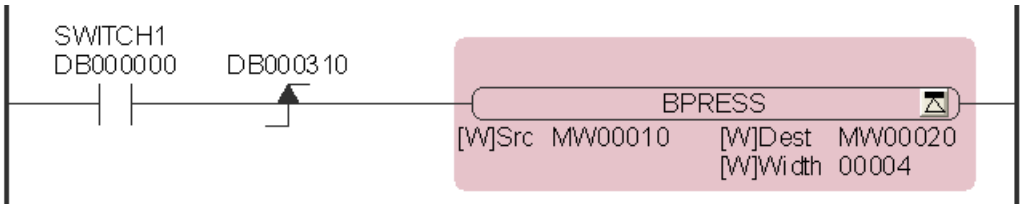



| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source register address (Src) | × | ○ | × | × | × | × | × |
| First destination register address (Dest) | × | ○* | × | × | × | × | × |
| Number of bytes to move (Width) | × | ○ | × | × | × | ○ | ○ |

＊ C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the lower bytes of data are moved from the area of four words that starts with the first source register address at MW00010 to the area of four bytes that starts with the first destination register address at MW00020.
The BPRESS instruction is executed when switch 1 (DB000000) turns ON.




The following table illustrates how the word data in the source area is compressed and moved into byte data in the destination area.

| Source area | | |
|---|---|---|
| Register | | Data |
| MW00010 | Lower byte | 12 hex |
| | Upper byte | 23 hex |
| MW00011 | Lower byte | 34 hex |
| | Upper byte | 45 hex |
| MW00012 | Lower byte | 56 hex |
| | Upper byte | 67 hex |
| MW00013 | Lower byte | 78 hex |
| | Upper byte | 89 hex |

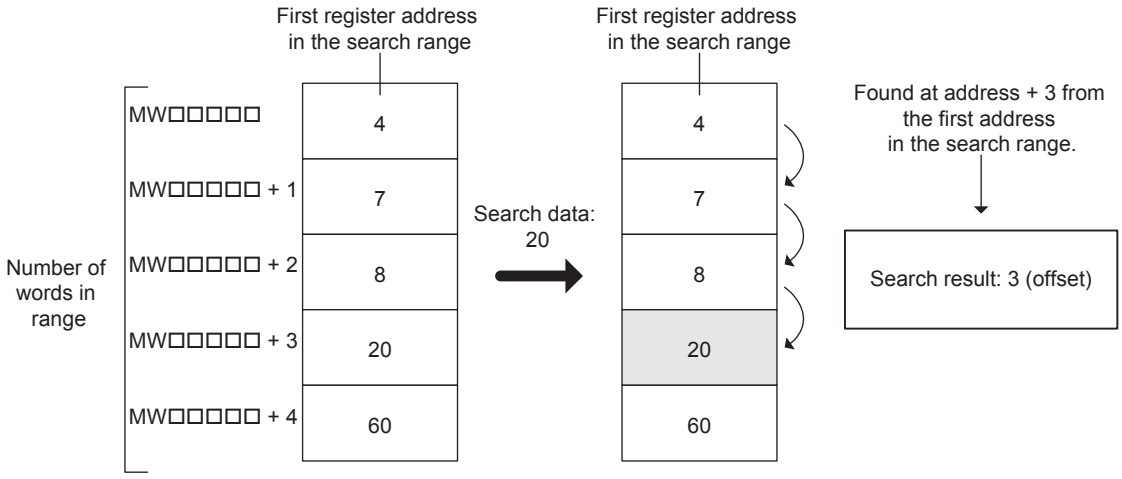
⇒

| Destination area | | |
|---|---|---|
| Register | | Data |
| MW00020 | Lower byte | 12 hex |
| | Upper byte | 34 hex |
| MW00021 | Lower byte | 56 hex |
| | Upper byte | 78 hex |

# 5.7.9 Binary Search (BSRCH)

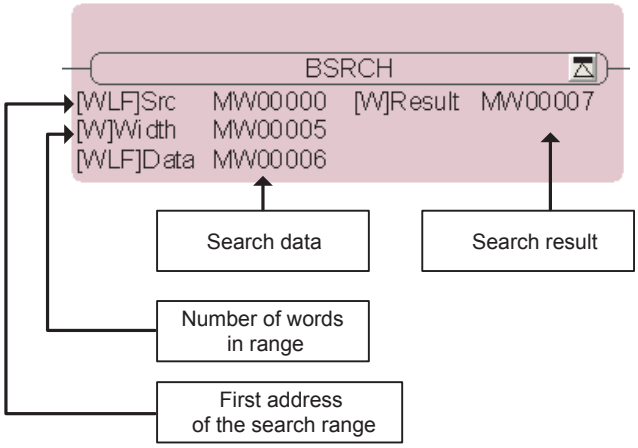
## ( 1 ) Operation

The BSRCH instruction searches for the search data using a binary search method in the area designated by the number of words from the first address in the search range. The search result is output as the offset word number of the data that matches the search data from the first register in the search range.




∗ 1. Always sort the search area in ascending order before executing the BSRCH instruction.
∗ 2. The conceptual diagram shown here is for integers. The instruction operates in the same way for double-length integers and real numbers.
∗ 3. If the search data is not found, the instruction sets the search result to -1.

## ( 2 ) Format




Icon: B SRCH

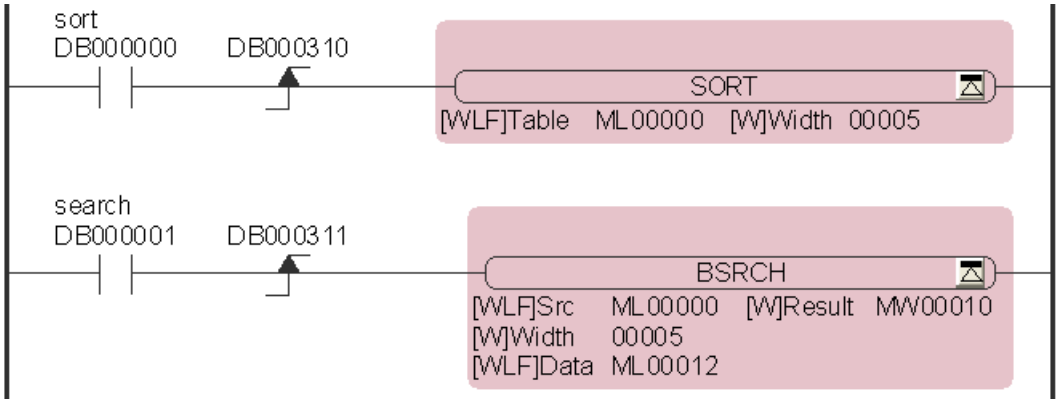
Key entry: BSRCH

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of search range (Src) | × | ○ | ○ | ○ | × | × | × |
| Number of words in range (Width) | × | ○ | × | × | × | ○ | ○ |
| Search data (Data) | × | ○ | ○ | ○ | × | ○ | ○ |
| Search result (Result) | × | ○* | × | × | × | × | × |

∗ C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the data from ML00000 to ML00008 is sorted when the sort command (DB000000) turns ON.
Then, if the search command (DB000001) turns ON, the search data in ML00012 is searched for in the sorted data area.



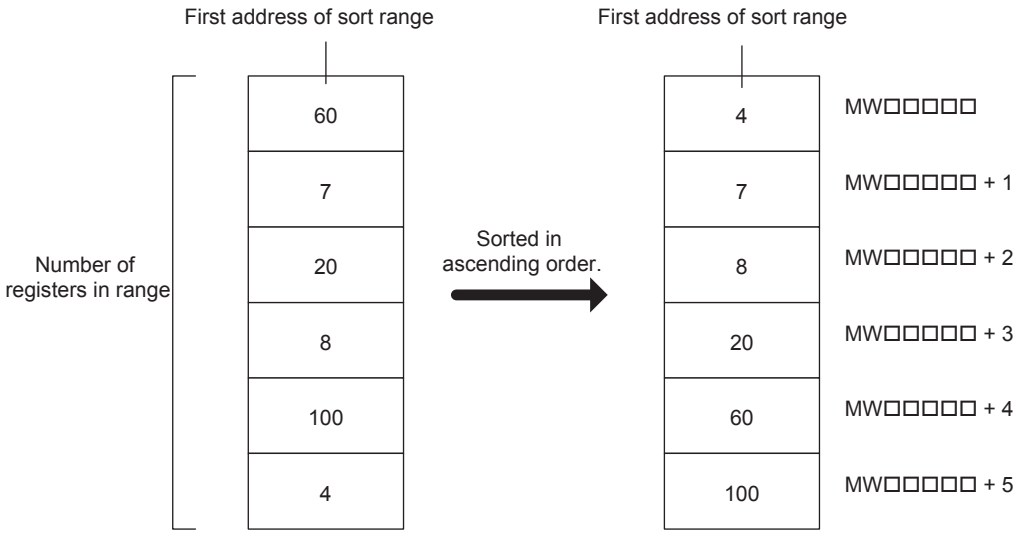

The following table shows how the sort is processed when the first line is executed. Here, the data from ML00000 to ML00008 is as listed below, and the search data in ML00012 is 70. When the second line is executed, the search result in MW00010 is set to 4 as the result of finding 70.

| Register | Data before Execution of 1st Line | Data after Execution of 1st Line | Execution Result of 2nd Line |
|---|---|---|---|
| ML00000 | 100 | 15 | ML00004 = 70, so MW00010 = 4 |
| ML00002 | 30 | 30 | |
| ML00004 | 90 | 70 | |
| ML00006 | 15 | 90 | |
| ML00008 | 70 | 100 | |

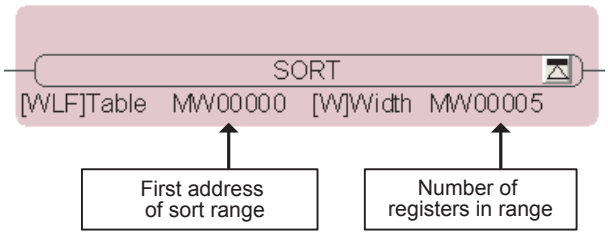
# 5.7.10 Sort (SORT)

## ( 1 ) Operation

The SORT instruction sorts the data in the range of registers from the first address of the sort range in ascending order. The following diagram describes the operation using integers as an example. The sort is performed in the same way for double-length integers and real numbers.




## ( 2 ) Format




Icon: SORT

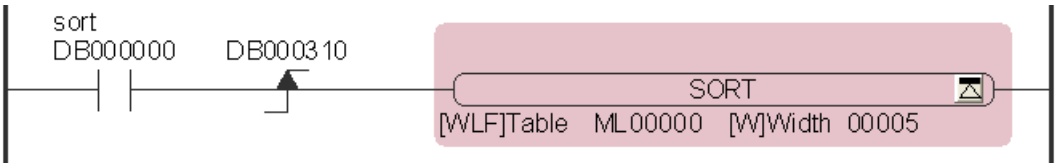
Key entry: SORT

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of sort range (Table) | × | ○* | ○* | ○* | × | × | × |
| Number of registers in range (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the data from ML00000 to ML00008 is sorted in ascending order when the sort command (DB000000) turns ON.



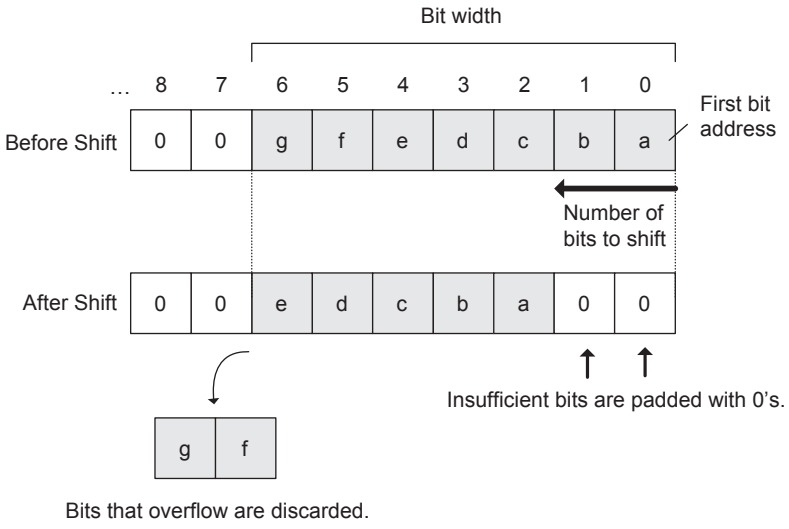

The following table shows how the data from ML00000 to ML00008 is sorted when the SORT instruction is executed.

| Register | Data before Execution of Instruction | Data after Execution of Instruction |
|---|---|---|
| ML00000 | 100 | 15 |
| ML00002 | 30 | 30 |
| ML00004 | 90 | 70 |
| ML00006 | 15 | 90 |
| ML00008 | 70 | 100 |

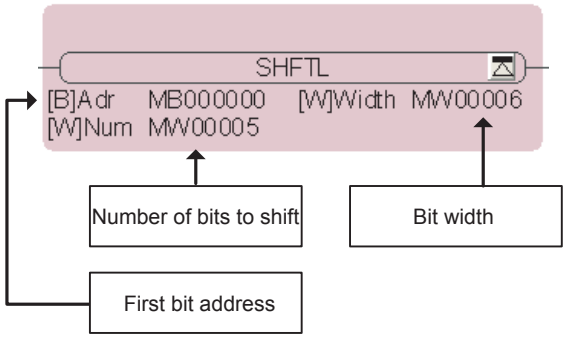
# 5.7.11 Bit Shift Left (SHFTL)

## ( 1 ) Operation

The SHFTL instruction shifts the bits specified by the first bit address and bit width to the left by the specified number of bits to shift.
Data that overflows from the bit width is discarded and insufficient bits are padded with 0's.




## ( 2 ) Format




Icon: SHFT L

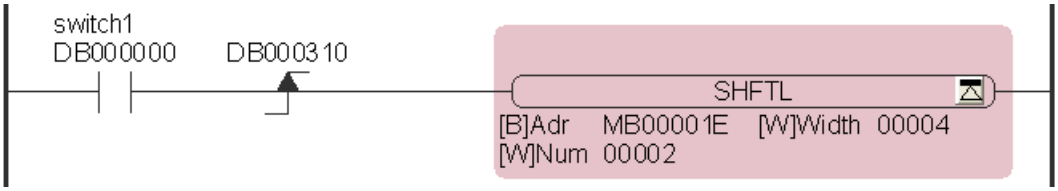
Key entry: SHFTL

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First bit address (Adr) | ○* | × | × | × | × | × | × |
| Number of bits to shift (Num) | × | ○ | × | × | × | ○ | ○ |
| Bit width (Width) | × | ○ | × | × | × | ○ | ○ |

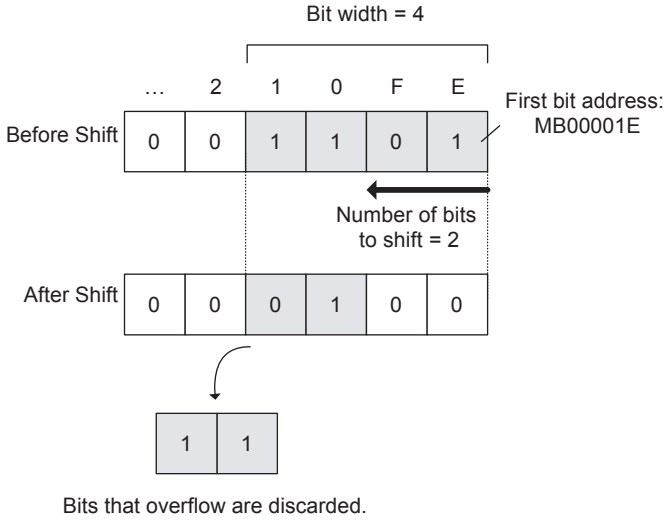
* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four bits from the first bit address at MB00001E are shifted two bits to the left when switch 1 (DB000000) turns ON.



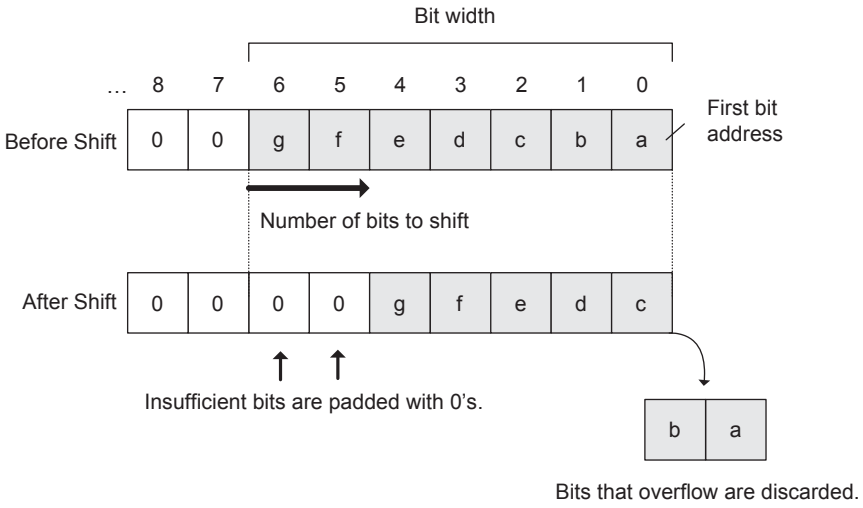

The following figure illustrates the result when the above instructions are executed.



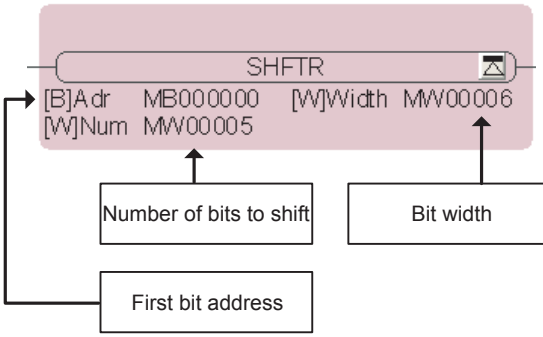
# 5.7.12 Bit Shift Right (SHFTR)

## ( 1 ) Operation

The SHFTR instruction shifts the bits specified by the first bit address and bit width to the right by the specified number of bits to shift.
Data that overflows from the bit width is discarded and insufficient bits are padded with 0's.




## ( 2 ) Format




Icon: SHFT R

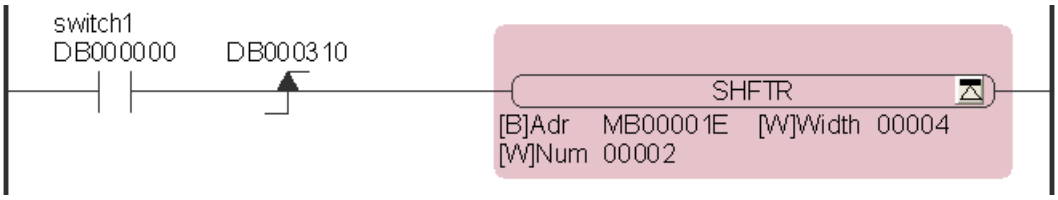
Key entry: SHFTR

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First bit address (Adr) | ○* | × | × | × | × | × | × |
| Number of bits to shift (Num) | × | ○ | × | × | × | ○ | ○ |
| Bit width (Width) | × | ○ | × | × | × | ○ | ○ |

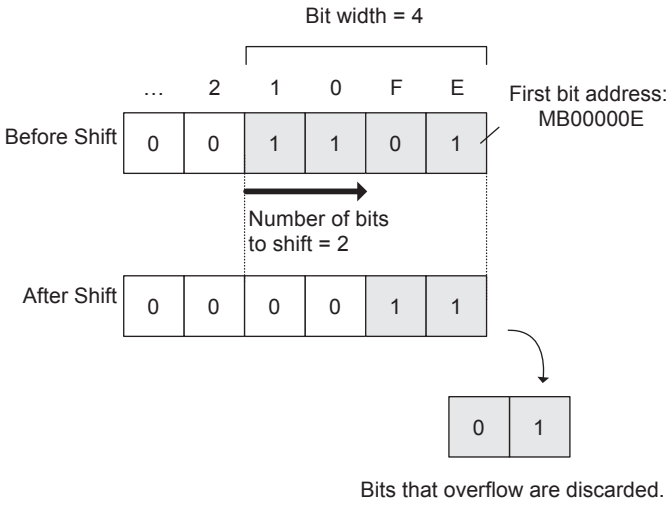
＊ C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, four bits from the first bit address at MB00001E are shifted two bits to the right when switch 1 (DB000000) turns ON.



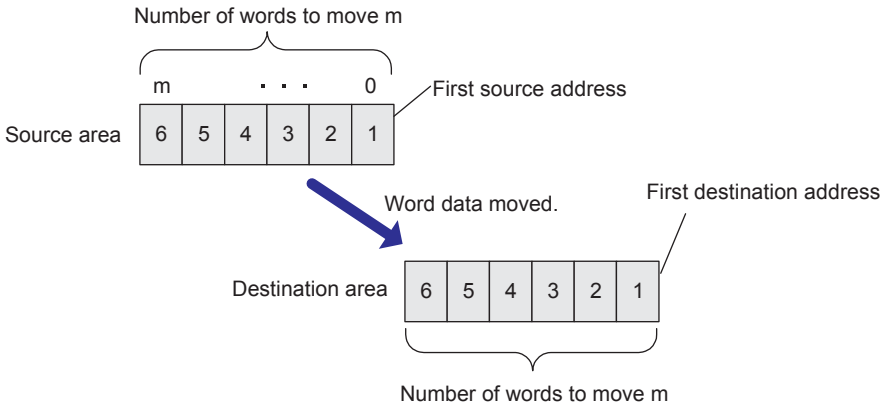

The following figure illustrates the result when the above instructions are executed.



# 5.7.13 Copy Word (COPYW)

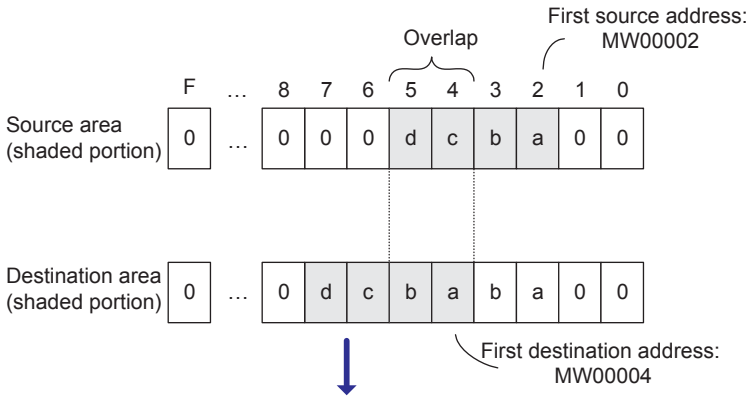
## ( 1 ) Operation

The COPY instruction copies the specified number of words to move from the area that starts with the first source address to the area that starts with the first destination address.
The data is copied as a block from the source to the destination. Unlike the MOVW instruction, the data is copied to the destination as is, even if the source and destination overlap.



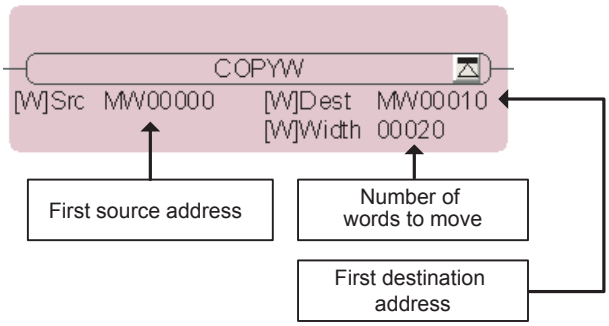

- This instruction differs from the MOVW instruction by the way it handles overlap between the source and destination areas.

### ■ Example Where the Source Area and Destination Area Overlap




Unlike the MOVW instruction, all of the data in the source area is moved to the destination area, even if the two areas overlap.

## ( 2 ) Format




Icon: COPY W

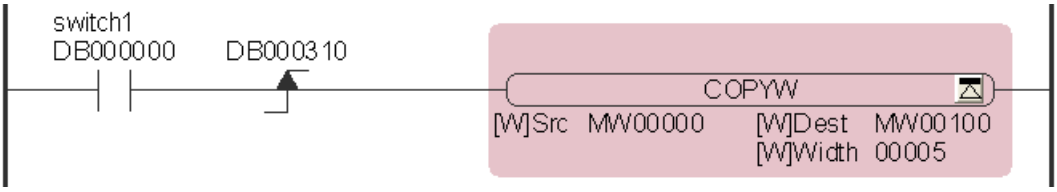
Key entry: COPYW

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source address (Src) | × | ○ | × | × | × | × | × |
| First destination address (Dest) | × | ○* | × | × | × | × | × |
| Number of words to move (Width) | × | ○ | × | × | × | ○ | ○ |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, five words of data are copied from the area that starts with the first source address at MW00000 to the area that starts with the first destination address at MW00100 when switch 1 (DB000000) turns ON.




The following figure illustrates the result when the above instructions are executed.

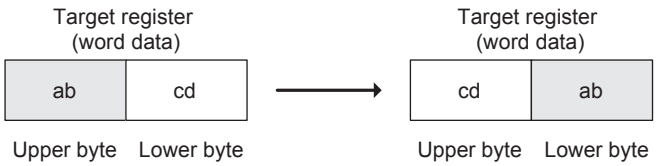
| Register | Data |
|---|---|
| MW00000 | 1 |
| MW00001 | 2 |
| MW00002 | 3 |
| MW00003 | 4 |
| MW00004 | 5 |

| Register | Data before Execution of Instruction | Data after Execution of Instruction |
|---|---|---|
| MW00100 | 123 | 1 |
| MW00101 | 234 | 2 |
| MW00102 | 345 | 3 |
| MW00103 | 456 | 4 |
| MW00104 | 567 | 5 |

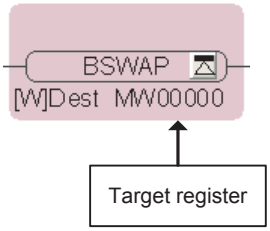
# 5.7.14 Byte Swap (BSWAP)

## ( 1 ) Operation

The BSWAP instruction swaps the upper byte and lower byte of the target register.




## ( 2 ) Format




Icon:


Key entry: BSWAP

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Target register (Dest) | × | O* | × | × | × | × | × |

* C and # registers cannot be used.

## ( 3 ) Programming Example

In the following programming example, the upper byte and lower byte of MW00000 are swapped when switch 1 (DB000000) turns ON.

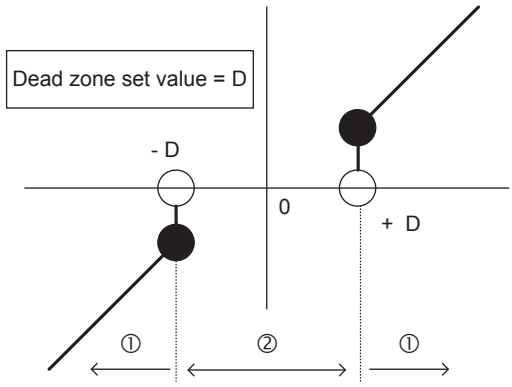
- If MW00000 is 00FF hex, MW00000 will be FF00 hex after execution of the BSWAP instruction.

# 5.8 DDC Instructions

## 5.8.1 Dead Zone A (DZA)

### ( 1 ) Operation

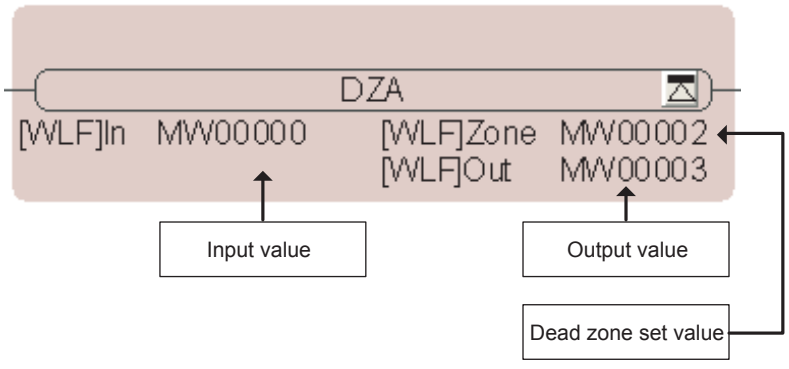
The DZA instruction calculates the output value by comparing the input value against a predefined dead zone.
As shown in the following figure, if the absolute value of the input value is greater than or equal to the absolute value of D, the input value is outside of the dead zone, so it becomes the output value.
If the absolute value of the input value is less than the absolute value of D, the input value is inside of the dead zone, so the output is set to 0.




① If | Input value | ≥ | D |
Output value = Input value

② If | Input value | < | D |
Output value = 0

### ( 2 ) Format




Icon:

Key entry: DZA

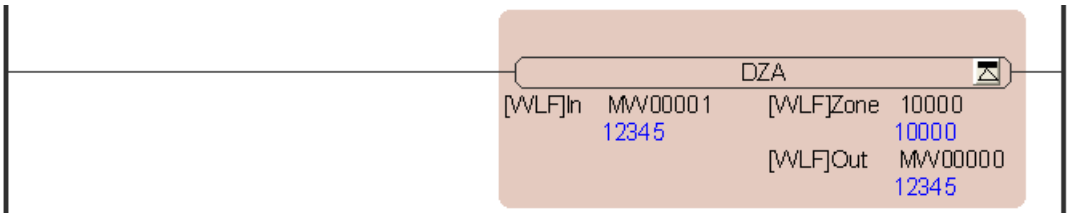
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| Dead zone set value (Zone) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output value (Out) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

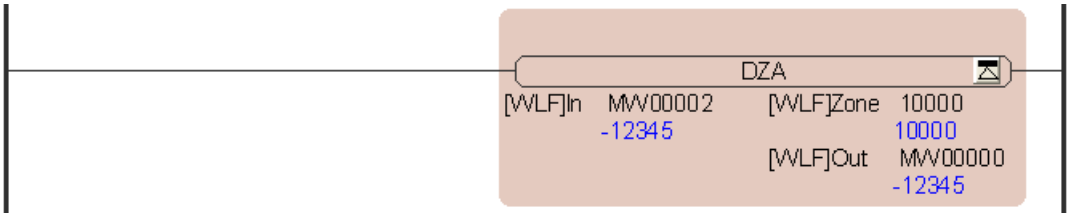
## ( 3 ) Programming Examples

In the following programming examples, the dead zone set value is set to 10,000 and the output value is stored in MW00000.
The output values are calculated with respect to the input values in MW00001 to MW00003 as shown below.

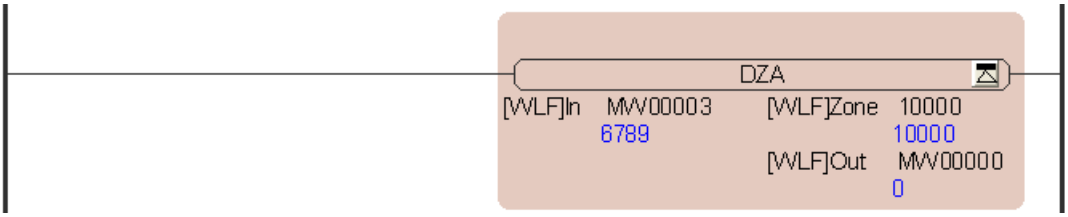
- Outside of the Dead Zone
  | MW00001 (12,345) | ≥ | 10,000 |, so MW00000 is 12,345.




  | MW00002 (-12,345) | ≥ | 10,000 |, so MW00000 is -12,345.



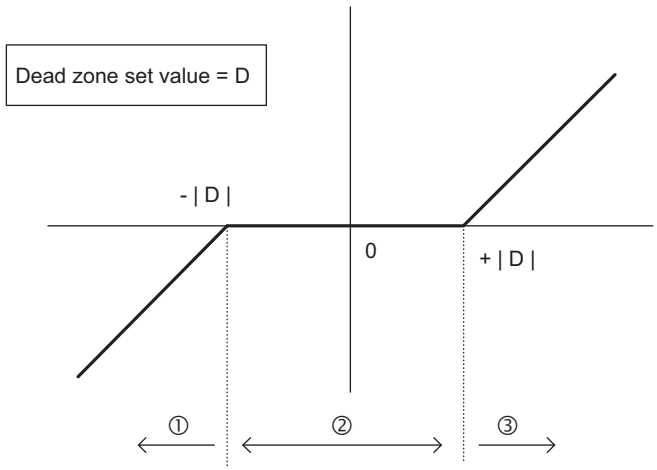

- Inside of the Dead Zone
  | MW00003 (6,789) | < | 10,000 |, so MW00000 is 0.



## 5.8.2 Dead Zone B (DZB)

### ( 1 ) Operation

The DZB instruction calculates the output value by comparing the input value against a predefined dead zone.
As shown in the following figure, if the absolute value of the input value is less than the absolute value of D, the input value is inside of the dead zone, so the output is set to 0.
Unlike the DZA instruction, when the input value is outside of the dead zone, the sign of the input value determines whether the output value is obtained by adding the absolute value of D to or subtracting it from the input value.

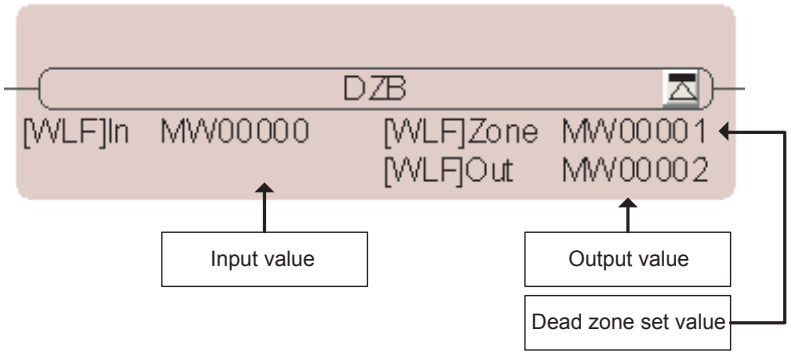



① If Input value < 0 and | Input value | ≥ | D |
Output value = Input value + | D |

② If | Input value | < | D |
Output value = 0

③ If Input value > 0 and | Input value | ≥ | D |
Output value = Input value – | D |

### ( 2 ) Format




Icon:

Key entry: DZB

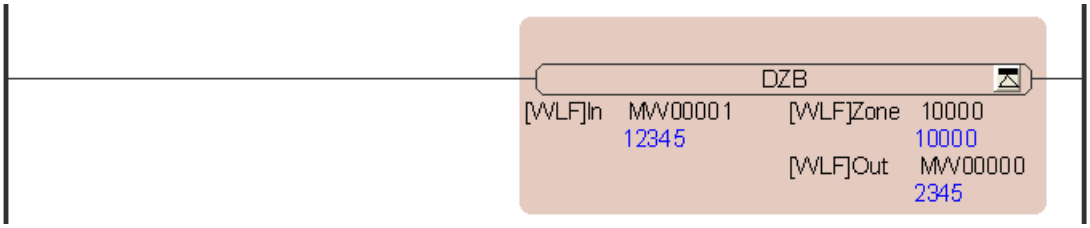
| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| Dead zone set value (Zone) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output value (Out) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

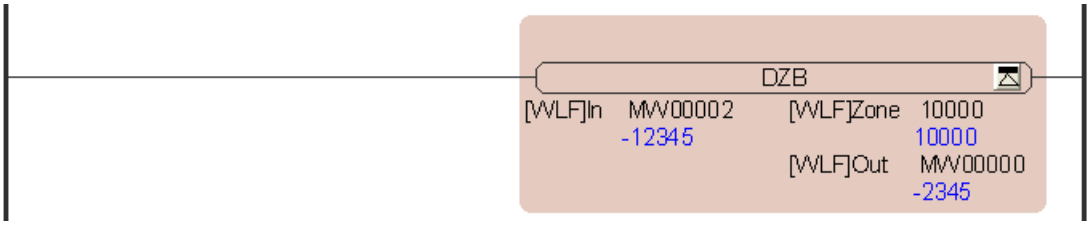
## ( 3 ) Programming Examples

In the following programming examples, the dead zone set value is set to 10,000 and the output value is stored in MW00000.
The output values are calculated with respect to the input values in MW00001 to MW00003 as shown below.

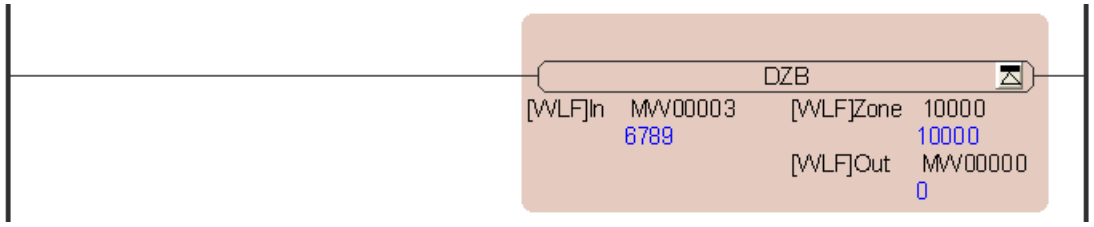
- Outside of the Dead Zone
  Because MW00001 (12,345) > 0 and | MW00001 (12,345) | ≥ | 10,000 |, so MW00000 = 12,345 – | 10,000 | = 2,345.




MW00002 (-12,345) < 0, | MW00002 (-12,345) | ≥ | 10,000 |, so MW00000 = -12,345 + | 10,000 | = -2,345.



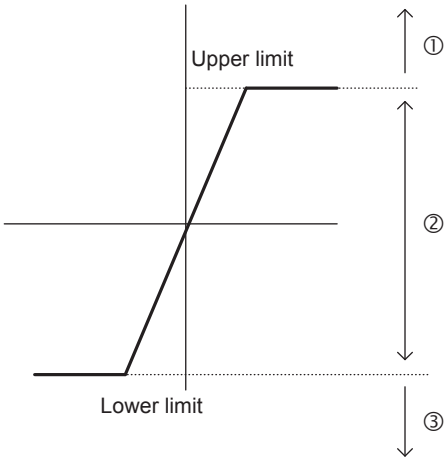

- Inside of the Dead Zone
  | MW00003 (6,789) | < | 10,000 |, so MW00000 is 0.



## 5.8.3 Upper/Lower Limit (LIMIT)

### ( 1 ) Operation

The LIMIT instruction controls the output value so that it does not exceed the specified upper and lower limits for the input value.
As shown in the following figure, if the input value is within the upper and lower limits, the input value is output unaltered.
The upper limit is output when the input value is greater than upper limit. The lower limit is output when the input value is less than the lower limit.

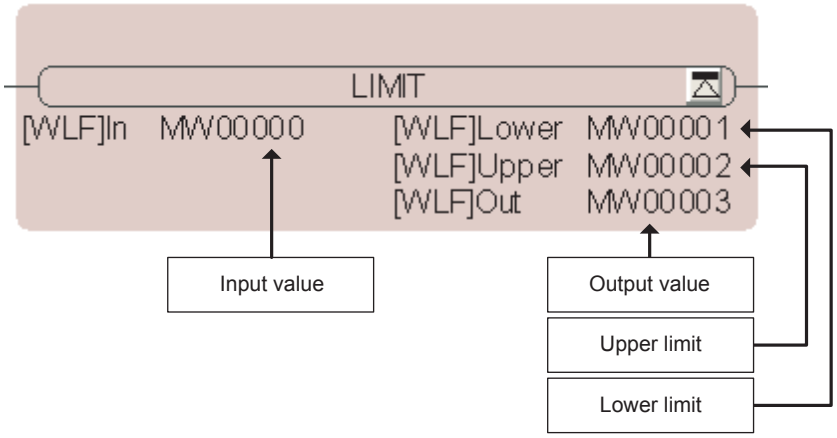



① If Input value > Upper limit
Output value = Upper limit

② If Lower limit ≤ Input value ≤ Upper value
Output value = Input value

③ Input value < Lower limit
Output value = Lower limit

### ( 2 ) Format




Icon:

Key entry: LIMIT

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| Lower limit (Lower) | × | ○ | ○ | ○ | × | ○ | ○ |
| Upper limit (Upper) | × | ○ | ○ | ○ | × | ○ | ○ |
| Output value (Out) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

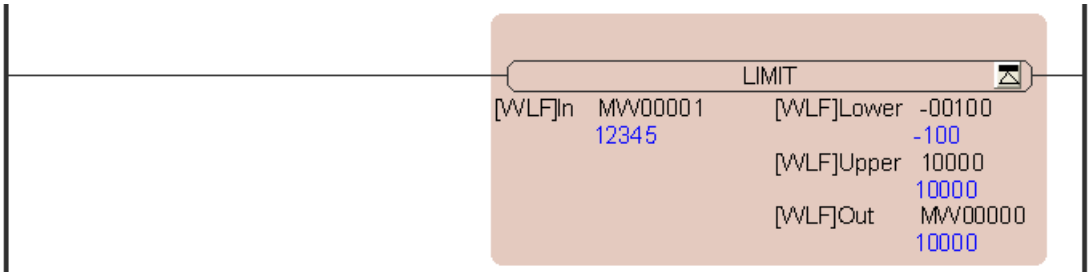
INFO

Always set the lower limit to a value that is less than or equal to the upper limit.

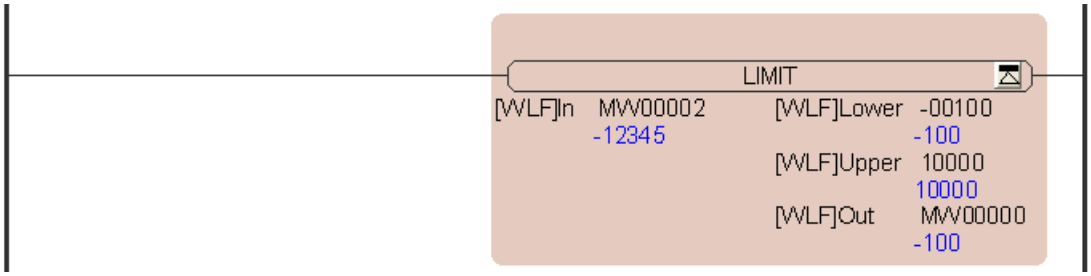
## ( 3 ) Programming Examples

In the following programming examples, the operation results are stored as the output value (MW00000) when the lower limit is -100 and the upper limit is 10,000.
The output values are calculated with respect to the input values in MW00001 to MW00003 as shown below.

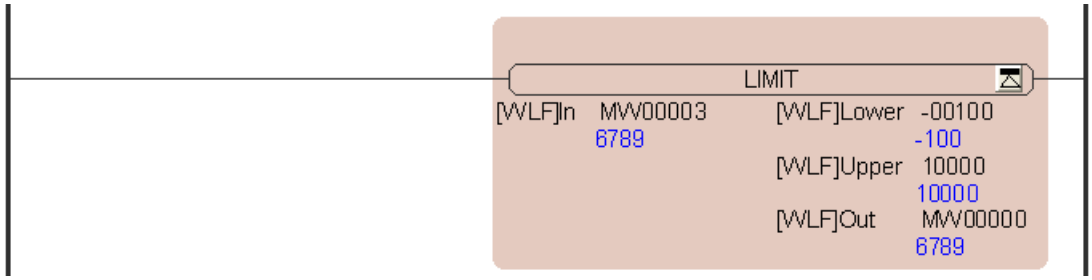
- The Input Value Is Outside of the Upper and Lower Limits
  Because MW00001 (12,345) is greater than the upper limit (10,000), MW00000 becomes the upper limit (10,000).




Because MW00002 (-12,345) is less than the lower limit (-100), MW00000 becomes the lower limit (-100).




- The Input Value Is Within the Upper and Lower Limits
  Because the lower limit (-100) is less than MW00003 (6,789), which is less than the upper limit (10,000), MW00000 becomes 6,789.



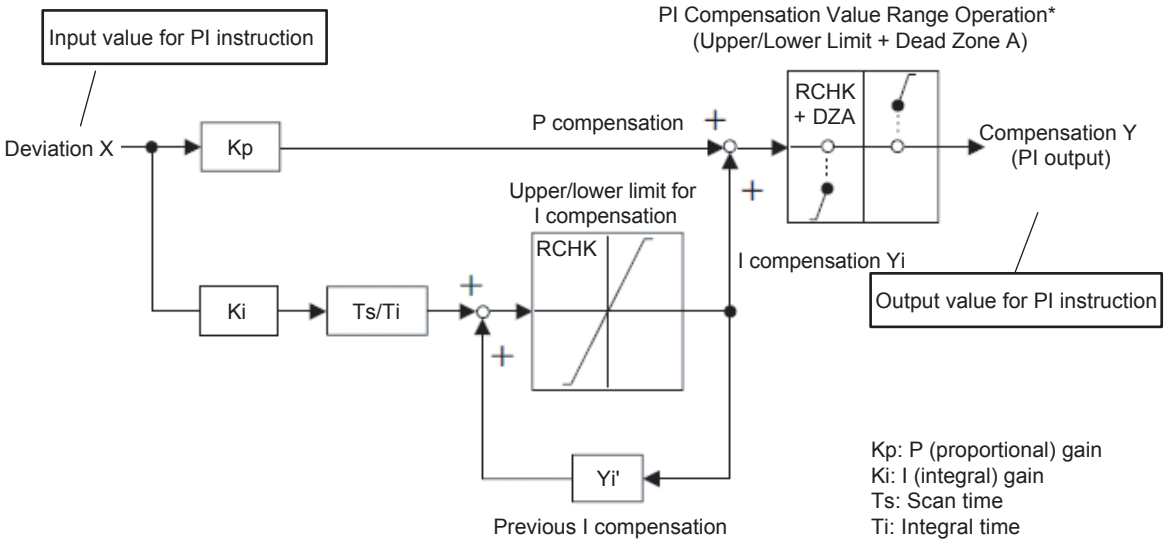
## 5.8.4 PI Control (PI)

### ( 1 ) Operation

When deviation X is input, the PI instruction performs P and I operations and a range operation based on predefined parameters in a parameter table, and outputs the result as compensation Y.
When the reset integration bit in the parameter table is closed (turned ON), the PI compensation is calculated using an I compensation value of 0.
The input value to the PI instruction can be an integer or a real number. Double-length integers cannot be used.
The structure of the parameter table is different for integers and real numbers.

**IMPORTANT**

If using an integer, set an integral multiple of 1 ms for the scan time.

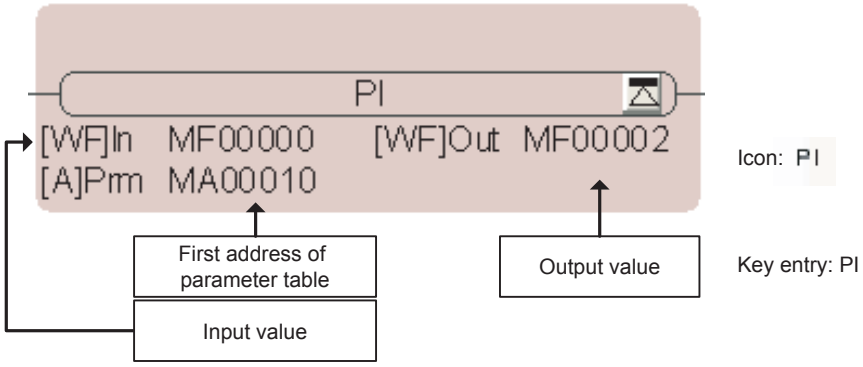



* The range operation for the PI compensation is processed as follows if the P + I compensation crosses the PI upper or lower limit (UL or LL), or the PI dead zone (DB):
  - If the P compensation and I compensation have the same sign (divergence) → The previous value is retained for the I compensation value.
  - If the P compensation and I compensation have different signs (convergence to 0) → The I compensation value is updated to a new value.

The operation of the PI instruction can be expressed by the following formula, where X (s) is the input value and Y (s) is the output value.

$$\frac{Y(s)}{X(s)} = Kp + Ki \times \frac{1}{Ti \times s}$$

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○* | ○ | ○ |
| Output value (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for PI Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 100.) | IN |
| 2 | W | Ki | I gain | Gain for the input to the integration circuit (A gain of 1 is equivalent to 100.) | IN |
| 3 | W | Ti | Integral time | Integral time (ms) | IN |
| 4 | W | IUL | Upper integration limit | Upper limit for the I compensation | IN |
| 5 | W | ILL | Lower integration limit | Lower limit for the I compensation | IN |
| 6 | W | UL | PI upper limit | Upper limit for the P + I compensation | IN |
| 7 | W | LL | PI lower limit | Lower limit for the P + I compensation | IN |
| 8 | W | DB | PI output dead zone | Dead zone width for the P + I compensation | IN |
| 9 | W | Y | PI output | PI compensation output (output to Out) | OUT |
| 10 | W | Yi | I compensation | I compensation storage | OUT |
| 11 | W | IREM | I remainder | I remainder storage | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | IRST | Reset integration bit | This input is closed to reset the integration operation. | IN |
| 1 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 to F | – | (Reserved.) | Spare output relays | OUT |

### [ b ] Parameter Table Configuration for PI Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 1.0.) | IN |
| 4 | F | Ki | I gain | Gain for the input to the integral circuit (A gain of 1 is equivalent to 1.0.) | IN |
| 6 | F | Ti | Integral time | Integral time (s) | IN |
| 8 | F | IUL | Upper integration limit | Upper limit for the I compensation | IN |
| 10 | F | ILL | Lower integration limit | Lower limit for the I compensation | IN |
| 12 | F | UL | PI upper limit | Upper limit for the P + I compensation | IN |
| 14 | F | LL | PI lower limit | Lower limit for the P + I compensation | IN |
| 16 | F | DB | PI output dead zone | Dead zone width for the P + I compensation | IN |
| 18 | F | Y | PI output | PI compensation output (output to Out) | OUT |
| 20 | F | Yi | I compensation | I compensation storage | OUT |

* The relay input and output bit assignments are the same as for integers.

### [ c ] Internal Operation of the Instruction

The deviation X input is used to calculate the output value (PI compensation) as shown below.
In the formula shown below, Yi' is the previous I compensation of Yi and Ts is the scan time set value.

- When IRST (reset integration) is closed, the PI compensation is calculated with the I compensation set to 0.

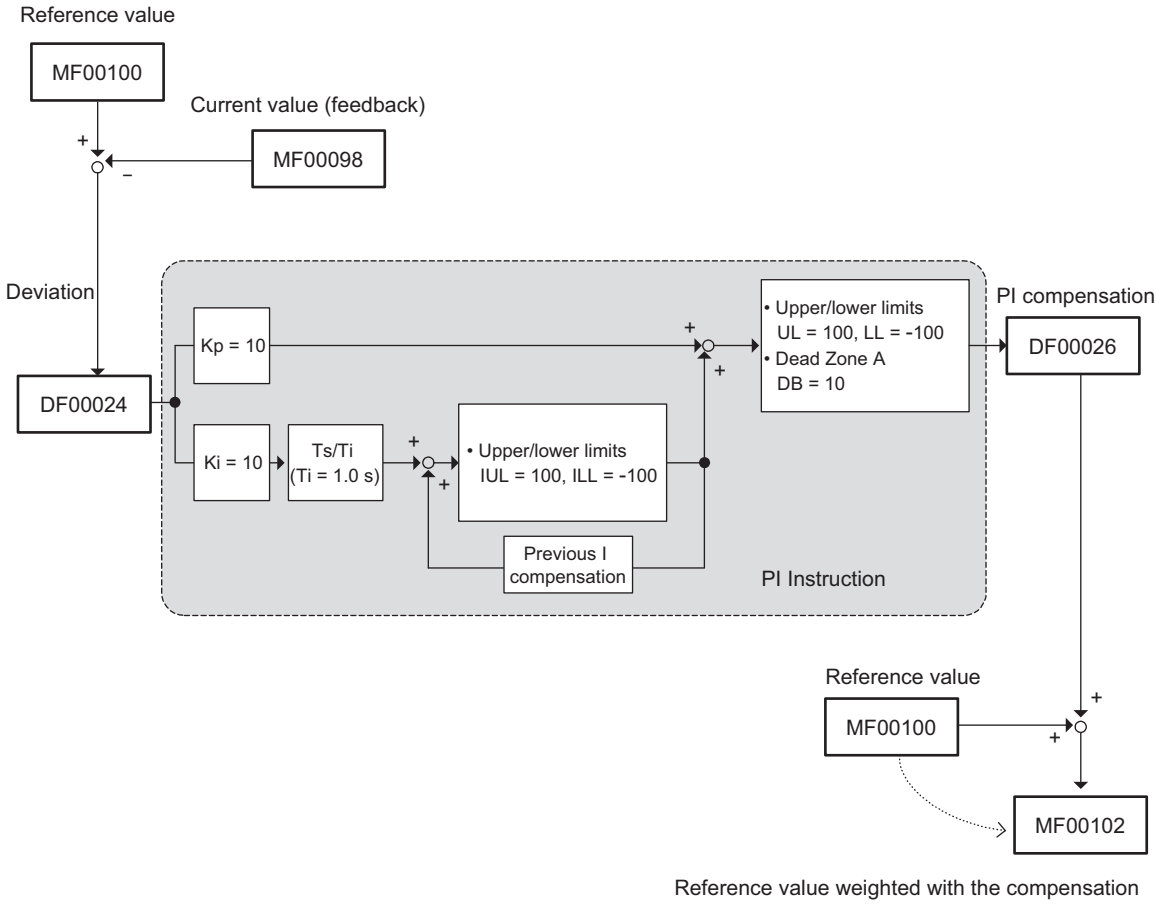
P compensation = Upper/lower limit (UL or LL) of (Kp × X)

$$\text{Yi (I compensation)} = \text{Upper/lower limit (IUL or ILL) of } \{ (Ki \times X + IREM) / \frac{Ti}{Ts} + Yi' \}$$

Y (PI compensation) = P compensation + Upper/lower limit (UL or LL) and Dead zone A (Width DB) of the I compensation

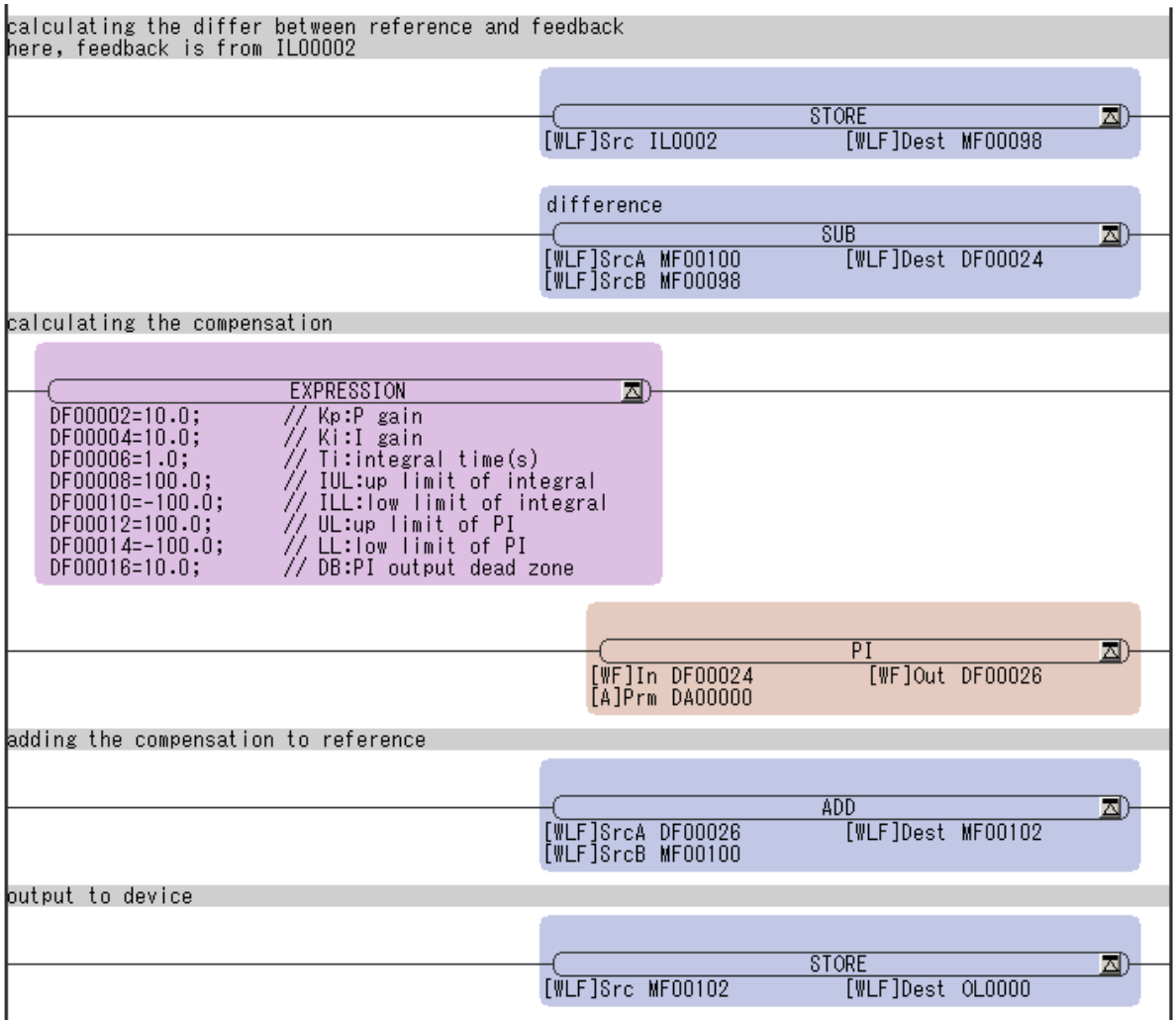
## ( 3 ) Programming Example

This programming example calculates the reference value in MF00102 weighted with the PI compensation.
The deviation in DF00024 is obtained from the reference value in MF00100 and the current value in MF00098 and it is used as the input to the PI instruction.
The reference value to output is obtained by adding the original reference value in MF00100 to the PI compensation output in DF00026.
The following block diagram illustrates the programming example.



The programming example is shown below.

- The OL00000 (reference value) and IL00002 (feedback value) registers are assigned to external devices.



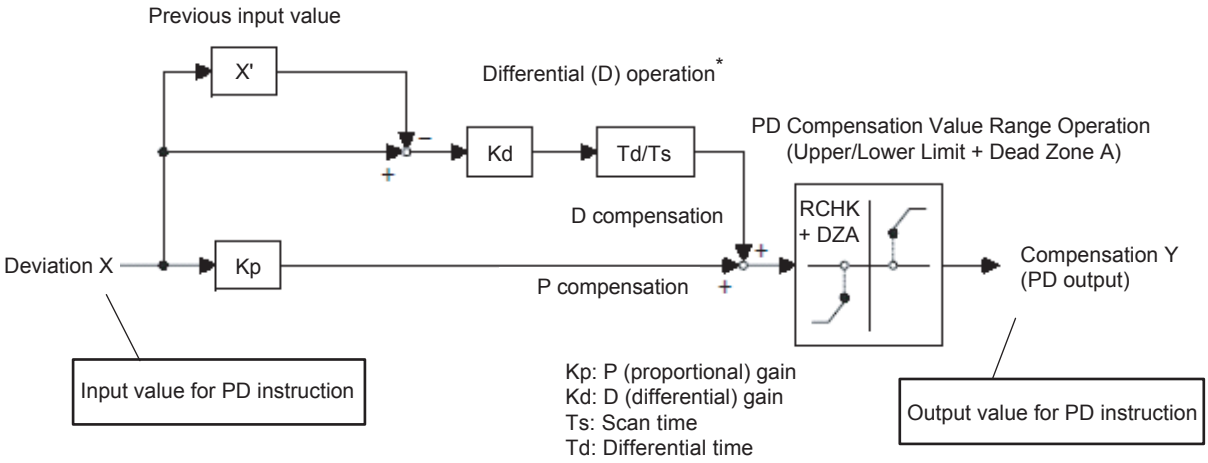
## 5.8.5 PD Control (PD)

### ( 1 ) Operation

When deviation X is input, the PD instruction performs P and D operations and a range operation based on predefined parameters in a parameter table, and outputs the result as compensation Y.
The input value to the PD instruction can be an integer or a real number. Double-length integers cannot be used.
The structure of the parameter table is different for integers and real numbers.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.

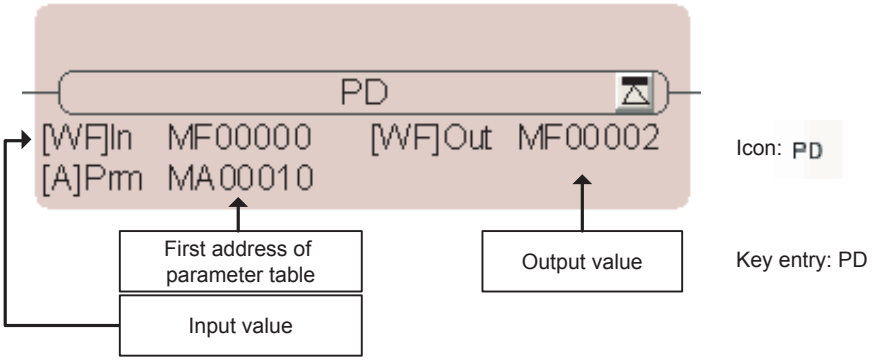



* The differential time (Td) changes based on the relationship between the change in the deviation input (X – X') and the previous deviation input (X') as follows:
  - If the change in the deviation input (X – X') and the previous deviation input (X') have the same sign (divergence)
    → Td = Td1 (differential time for divergence)
  - If the change in the deviation input (X – X') and the previous deviation input (X') have different signs (convergence)
    → Td = Td2 (differential time for convergence)

The operation of the PD instruction can be expressed by the following formula, where X (s) is the input value and Y (s) is the output value.

$$\frac{Y(s)}{X(s)} = Kp + Kd \times Td \times S$$

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○* | ○ | ○ |
| Output value (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for PD Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 100.) | IN |
| 2 | W | Kd | D gain | Gain for the input to the differential circuit (A gain of 1 is equivalent to 100.) | IN |
| 3 | W | Td1 | Differential time for divergence | Differential time used when the input diverges (ms) | IN |
| 4 | W | Td2 | Differential time for convergence | Differential time used when the input converges (ms) | IN |
| 5 | W | UL | PD upper limit | Upper limit for the P + D compensation | IN |
| 6 | W | LL | PD lower limit | Lower limit for the P + D compensation | IN |
| 7 | W | DB | PD output dead zone | Dead zone width for the P + D compensation | IN |
| 8 | W | Y | PD output | PD compensation output (output to Out) | OUT |
| 9 | W | X | Input value storage | Storage of current input value | OUT |

* The relay input and output bits are assigned as given below.

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 to F | – | (Reserved.) | Spare output relays | OUT |

### [ b ] Parameter Table Configuration for PD Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 1.0.) | IN |
| 4 | F | Kd | D gain | Gain for the input to the differential circuit (A gain of 1 is equivalent to 1.0.) | IN |
| 6 | F | Td1 | Differential time for divergence | Differential time used when the input diverges (s) | IN |
| 8 | F | Td2 | Differential time for convergence | Differential time used when the input converges (s) | IN |
| 10 | F | UL | PD upper limit | Upper limit for the P + D compensation | IN |
| 12 | F | LL | PD lower limit | Lower limit for the P + D compensation | IN |
| 14 | F | DB | PD output dead zone | Dead zone width for the P + D compensation | IN |
| 16 | F | Y | PD output | PD compensation output (output to Out) | OUT |
| 18 | F | X | Input value storage | Storage of current input value | OUT |

* The relay input and output bit assignments are the same as for integers.

### [ c ] Internal Operation of the Instruction

The deviation X input is used to calculate the PD compensation output as shown below.
In the formula shown below, X' is the previous input value of X, Ts is the scan time set value, and Td* is the differential time.

* The differential time (Td) is Td1 when X – X' and X' have the same sign, and Td2 when X – X' and X' have different signs.

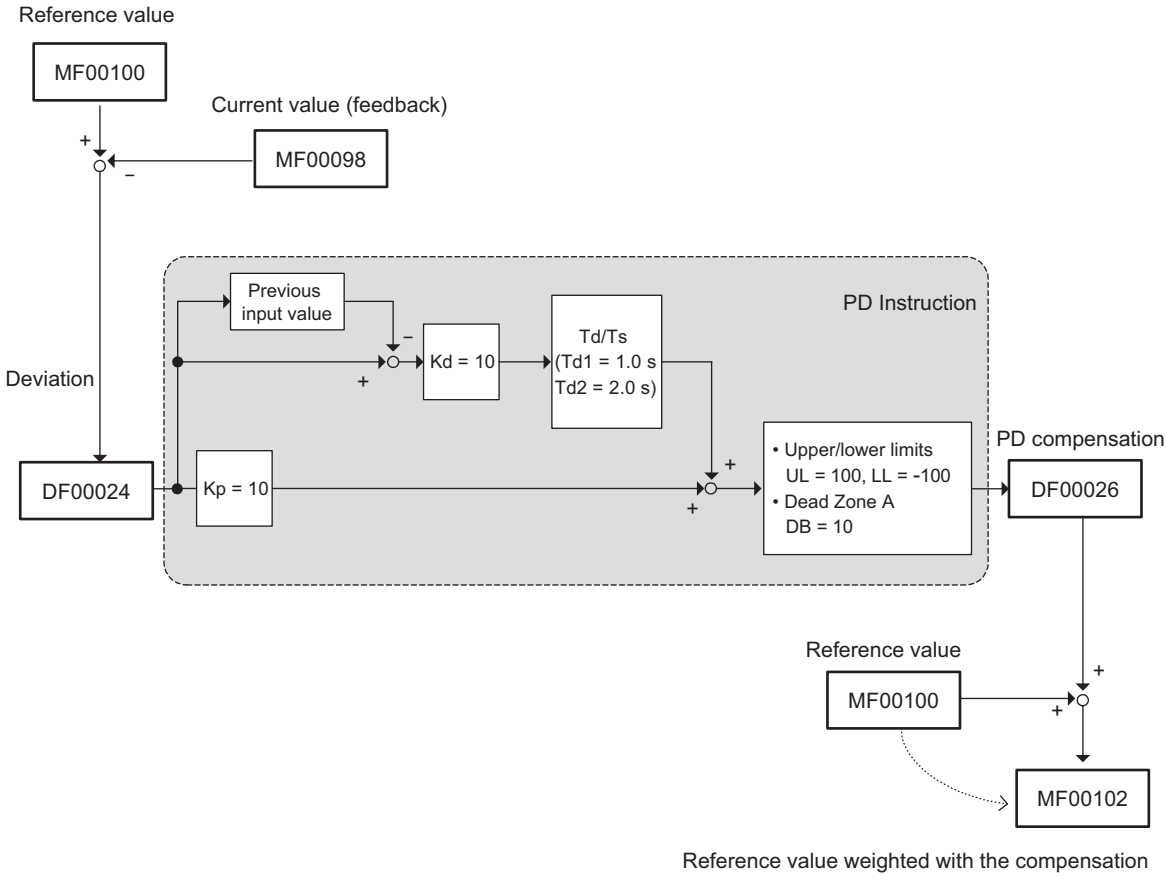
P compensation = Upper/lower limit (UL or LL) of (Kp × X)

D compensation = Kd × (X – X') × Upper/lower limit (IUL or ILL) of $\frac{Td}{Ts}$

PD compensation = Upper/lower limit (UL or LL) of (P compensation + D compensation) and Dead zone A (Width DB)

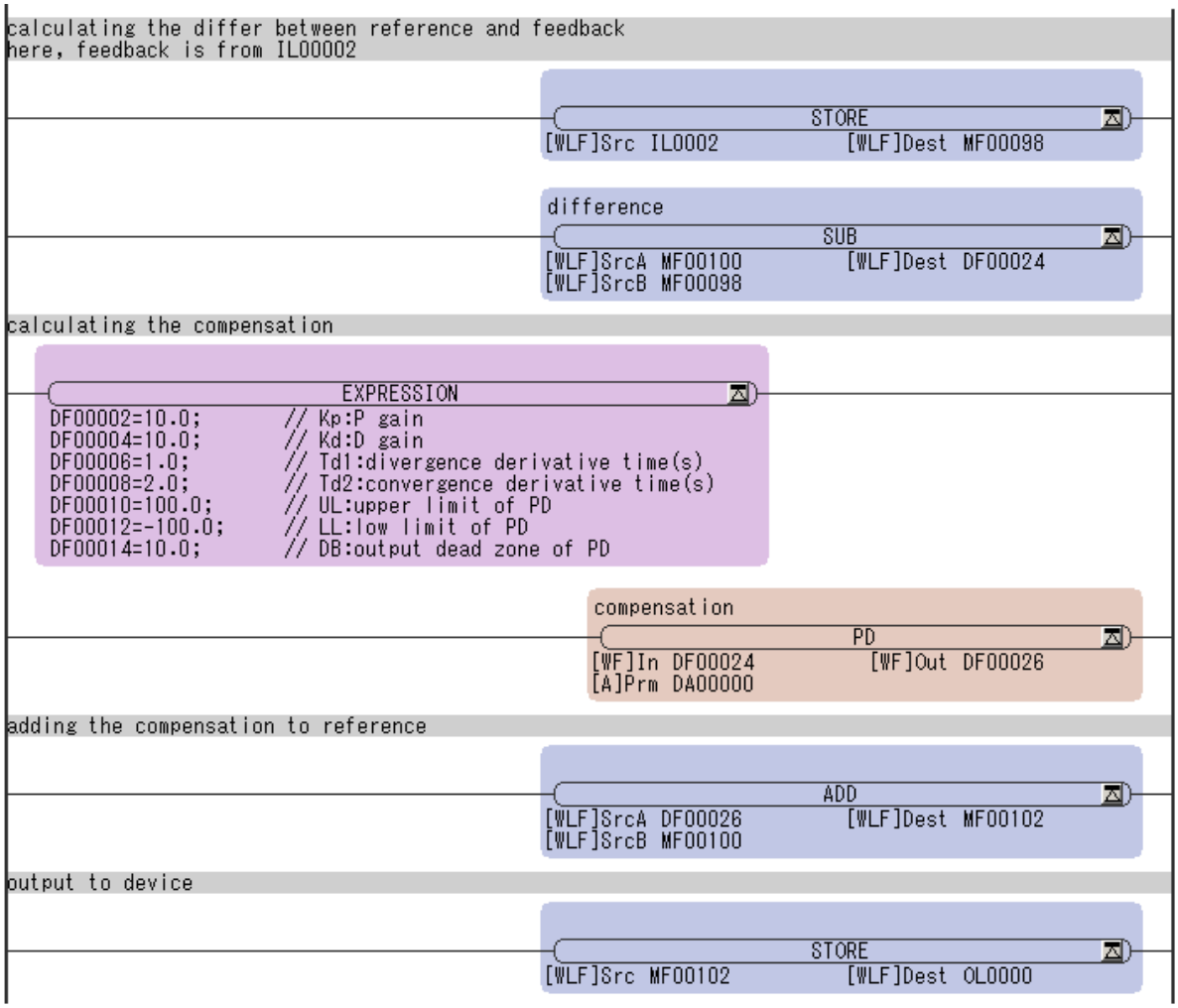
## ( 3 ) Programming Example

This programming example calculates the reference value in MF00102 weighted with the PD compensation.
The deviation in DF00024 is obtained from the reference value in MF00100 and the current value in MF00098 and it is used as the input to the PD instruction.
The reference value to output is obtained by adding the original reference value in MF00100 to the PD compensation output in DF000026.
The following block diagram illustrates the programming example.



The programming example is shown below.

- The OL00000 (reference value) and IL00002 (feedback value) registers are assigned to external devices.



## ( 4 ) Additional Information

### [ a ] Transfer Functions

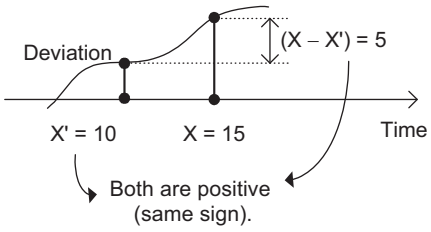
The transfer function of the P and D operations can be expressed by the formula shown below.
In this formula, X (s) is the input value and Y (s) is the output value.

$$\frac{Y(s)}{X(s)} = Kp + Kd \times Td \times S$$

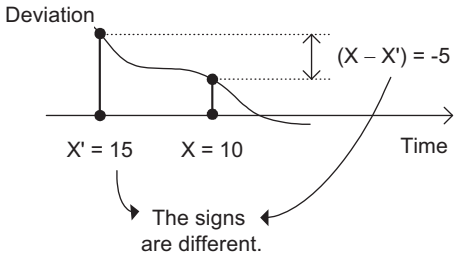
### [ b ] Divergence and Convergence

The following figure shows the relation between the current deviation X and previous deviation X' on the divergence and convergence sides.

• Example of a Diverging Deviation




• Example of a Converging Deviation



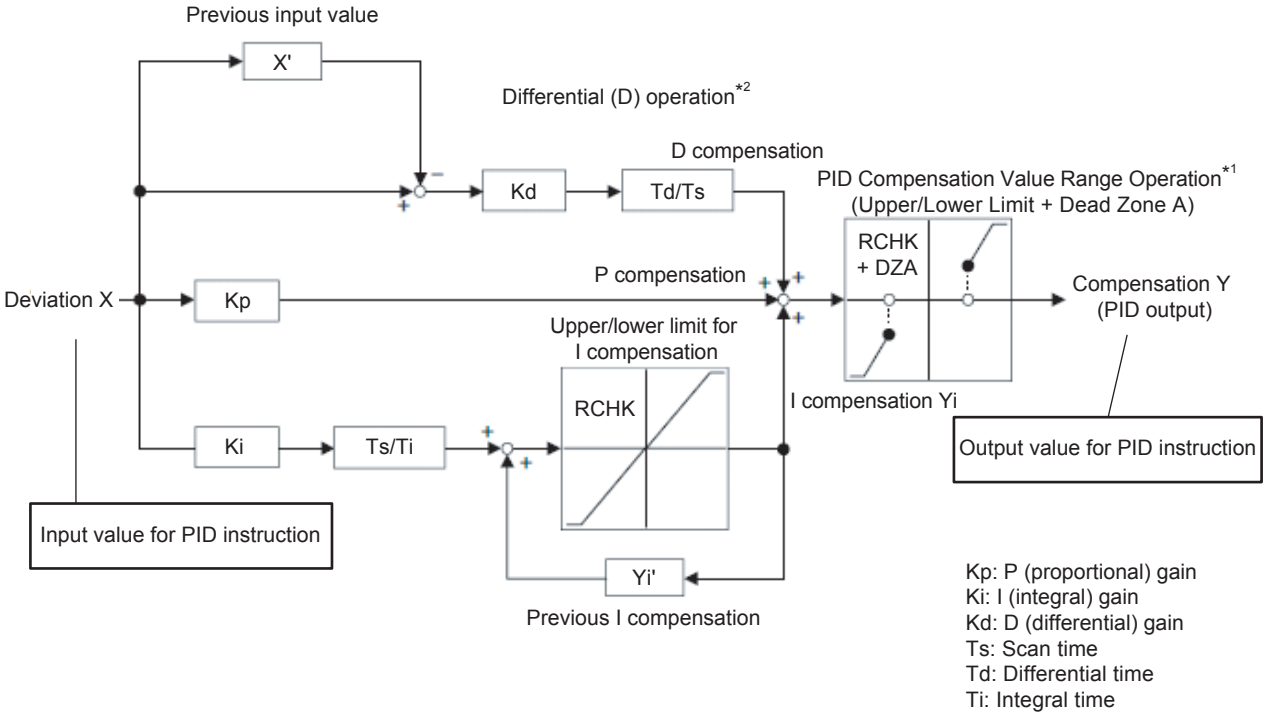
# 5.8.6 PID Control (PID)

## ( 1 ) Operation

When deviation X is input, the PID instruction performs P, I, and D operations and a range operation based on pre-defined parameters in a parameter table, and outputs the result as compensation Y.
When the reset integration bit in the parameter table is closed (turned ON), the PI compensation is calculated using an I compensation value of 0.
The input value to the PID instruction can be an integer or a real number. Double-length integers cannot be used.
The structure of the parameter table is different for integers and real numbers.

**IMPORTANT**

If using an integer, set an integral multiple of 1 ms for the scan time.



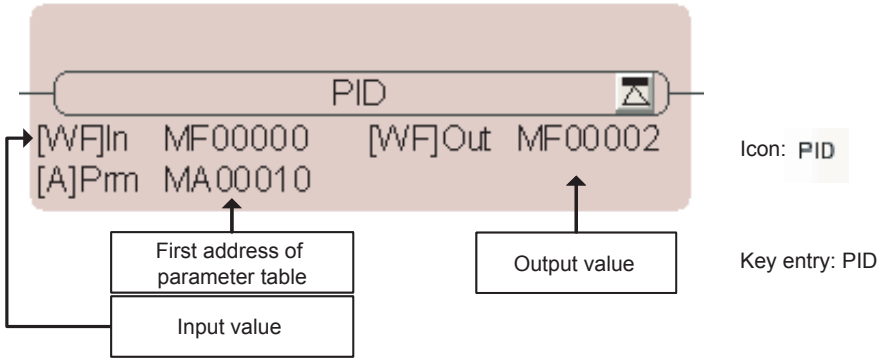

∗ 1. If the P + I + D compensation crosses the UL or LL (PID upper or lower limit), or DB (PI dead zone), the following processing is performed,
- If the P compensation and I compensation have the same sign (divergence) → The previous value is retained for the I compensation value.
- If the P compensation and I compensation have different signs (convergence to 0) → The I compensation value is updated to a new value.

∗ 2. The differential time (Td) changes based on the relationship between the change in the deviation input (X – X') and the previous deviation input (X') as follows:
- If the change in the deviation input (X – X') and the previous deviation input (X') have the same sign (divergence)
  → Td = Td1 (differential time for divergence)
- If the change in the deviation input (X – X') and the previous deviation input (X') have different signs (convergence)
  → Td = Td2 (differential time for convergence)

The operation of the PID instruction can be expressed by the following formula, where X (s) is the input value and Y (s) is the output value.

$$\frac{Y(s)}{X(s)} = Kp + Ki \times \frac{1}{Ti \times s} + Kd \times Td \times S$$

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○* | ○ | ○ |
| Output value (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for PID Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 100.) | IN |
| 2 | W | Ki | I gain | Gain for the input to the integration circuit (A gain of 1 is equivalent to 100.) | IN |
| 3 | W | Kd | D gain | Gain for the input to the differential circuit (A gain of 1 is equivalent to 100.) | IN |
| 4 | W | Ti | Integral time | Integral time (ms) | IN |
| 5 | W | Td1 | Differential time for divergence | Differential time used when the input diverges (ms) | IN |
| 6 | W | Td2 | Differential time for convergence | Differential time used when the input converges (ms) | IN |
| 7 | W | IUL | Upper integration limit | Upper limit for the I compensation | IN |
| 8 | W | ILL | Lower integration limit | Lower limit for the I compensation | IN |
| 9 | W | UL | PID upper limit | Upper limit for the P + I compensation | IN |
| 10 | W | LL | PID lower limit | Lower limit for the P + I compensation | IN |
| 11 | W | DB | PID output dead zone | Dead zone width for the P + I compensation | IN |
| 12 | W | Y | PID output | PI compensation output (output to Out) | OUT |
| 13 | W | Yi | I compensation | I compensation storage | OUT |
| 14 | W | IREM | I remainder | I remainder storage | OUT |
| 15 | W | X | Input value storage | Storage of current input value | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | IRST | Reset integration bit | This input is closed to reset the integration operation. | IN |
| 1 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 to F | – | (Reserved.) | Spare output relays | OUT |

### [ b ] Parameter Table Configuration for PID Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | IN |
| 2 | F | Kp | P gain | Gain for the P compensation (A gain of 1 is equivalent to 1.0.) | IN |
| 4 | F | Ki | I gain | Gain for the input to the integral circuit (A gain of 1 is equivalent to 1.0.) | IN |
| 6 | F | Kd | D gain | Gain for the input to the differential circuit (A gain of 1 is equivalent to 1.0.) | IN |
| 8 | F | Ti | Integral time | Integral time (s) | IN |
| 10 | F | Td1 | Differential time for divergence | Differential time used when the input diverges (s) | IN |
| 12 | F | Td2 | Differential time for convergence | Differential time used when the input converges (s) | IN |
| 14 | F | IUL | Upper integration limit | Upper limit for the I compensation | IN |
| 16 | F | ILL | Lower integration limit | Lower limit for the I compensation | IN |
| 18 | F | UL | PID upper limit | Upper limit for the P + I + D compensation | IN |
| 20 | F | LL | PID lower limit | Lower limit for the P + I + D compensation | IN |
| 22 | F | DB | PID output dead zone | Dead zone width for the P + I + D compensation | IN |
| 24 | F | Y | PID output | PID compensation output (output to Out) | OUT |
| 26 | F | Yi | I compensation | I compensation storage | OUT |
| 28 | F | X | Input value storage | Storage of current input value | OUT |

* The relay input and output bit assignments are the same as for integers.

### [ c ] Internal Operation of the Instruction

The deviation X input is used to calculate the PID compensation output as shown below.
In the formula shown below, X' is the previous input value of X, Y' is the previous I compensation, Ts is the scan time set value, and Td* is the differential time.

* The differential time (Td) is Td1 when X – X' and X' have the same sign, and Td2 when X – X' and X' have different signs.
* When IRST (reset integration) is closed, the PID compensation is calculated with the I compensation set to 0.

P compensation = Upper/lower limit (UL or LL) of (Kp × X)

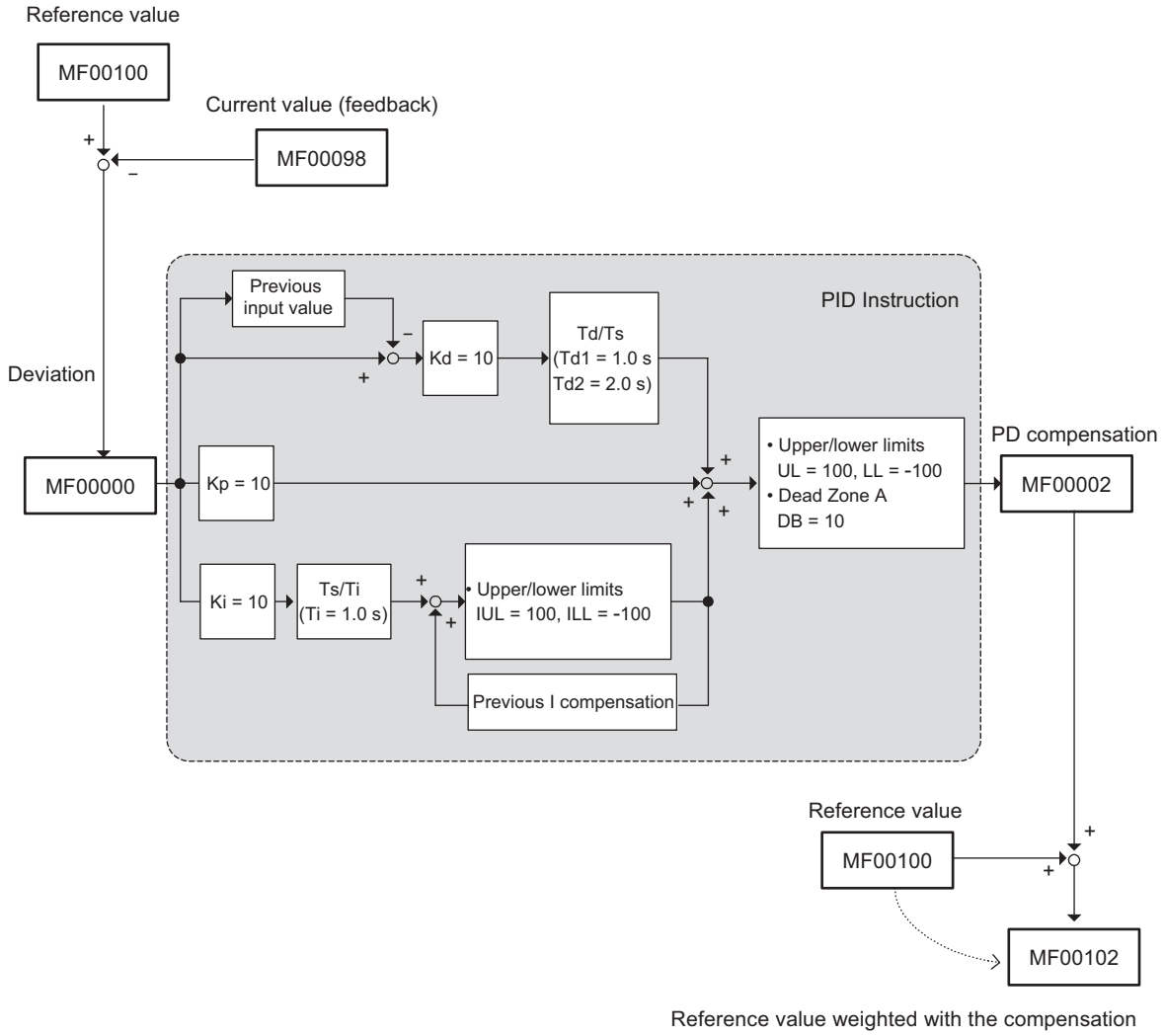
Yi (I compensation) = Upper/lower limit (IUL or ILL) of $\{ (Ki \times X + IREM) / \frac{Ti}{Ts} + Yi' \}$

D compensation = Kd × (X – X') × Upper/lower limit (IUL or ILL) of $\frac{Td}{Ts}$

Y (PID compensation) = Upper/lower limits (UL or LL) of P + I + D compensation values and dead zone A (Width DB)

## ( 3 ) Programming Example

This programming example calculates the reference value in MF00102 weighted with the PID compensation.
The deviation in MF00000 is obtained from the reference value in MF00100 and the current value in MF00098 and it is used as the input to the PID instruction.
The reference value to output is obtained by adding the original reference value in MF00100 to the PID compensation output in MF00002.
The following block diagram illustrates the programming example.



The programming example is shown below.

- The OL00000 (reference value) and IL00002 (feedback value) registers are assigned to external devices.

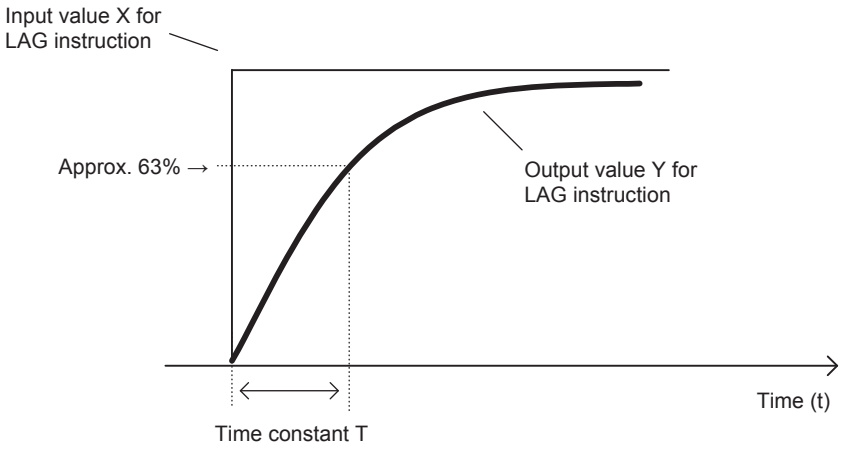
## 5.8.7 First-order Lag (LAG)

### ( 1 ) Operation

The LAG instruction calculates the first-order lag according to predefined parameters in a parameter table.
The input value to the LAG instruction can be an integer or a real number. Double-length integers cannot be used.
The structure of the parameter table is different for integers and real numbers.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.




The LAG operation in the figure shown above can be expressed by the formula shown below.

$$\frac{Y(s)}{X(s)} = \frac{1}{1 + T \times s}$$

Therefore,

$$T \times \frac{dY}{dt} + Y = X$$

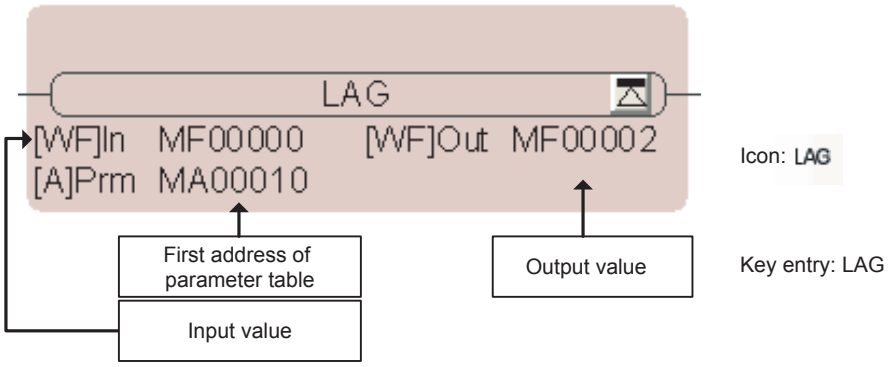
The following operation is performed internally by the LAG instruction, where dt = Ts and dY = Y – Y'.
In the formula shown below, Y' is the previous output value, Ts is the scan time set value,* and REM is the remainder.

* The unit for Ts is the same as the unit for T.

$$Y = \frac{T \times Y' + Ts \times X + REM}{T + Ts}$$

- When IRST (LAG reset) is closed, Y outputs 0 and REM outputs 0.
- The symbols in the figure correspond to those in the parameter table.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○* | ○ | ○ |
| Output value (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for LAG Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | T | First-order lag time constant | First-order lag time constant (ms) | IN |
| 2 | W | Y | LAG output | LAG output (output to Out) | OUT |
| 3 | W | REM | Remainder | Remainder storage | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | IRST | LAG reset bit | This input is closed to reset the LAG operation. | IN |
| 1 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 to F | – | (Reserved.) | Spare output relays | OUT |

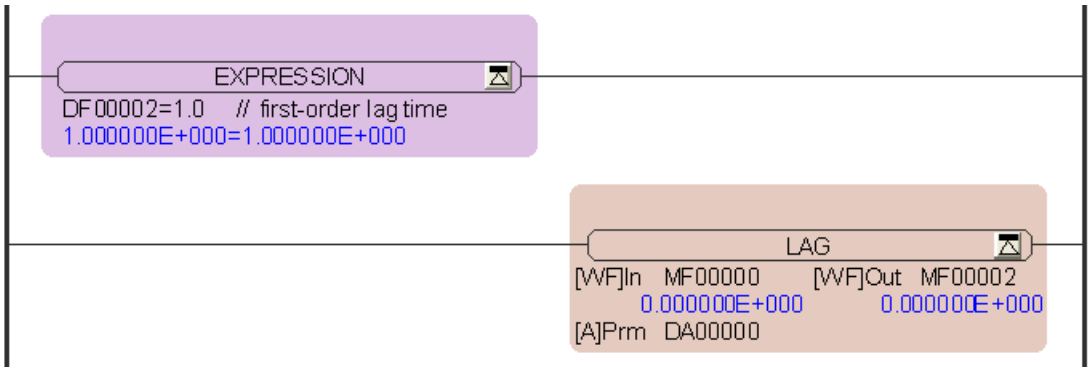
### [ b ] Parameter Table Configuration for LAG Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | T | First-order lag time constant | First-order lag time constant (s) | IN |
| 4 | F | Y | LAG output | LAG output (output to Out) | OUT |

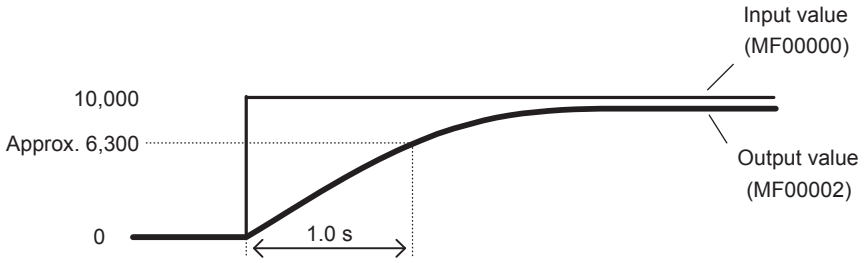
* The relay input and output bit assignments are the same as for integers.

## ( 3 ) Programming Example

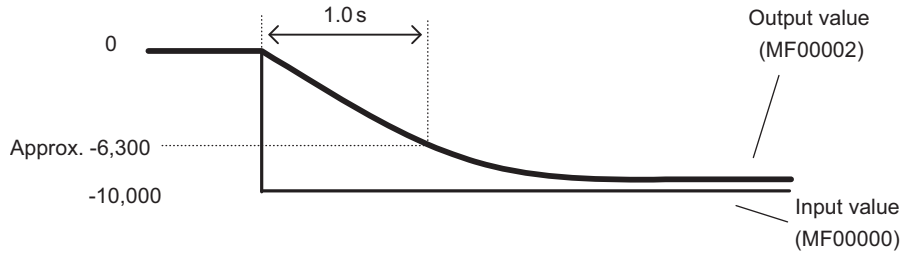
In the following programming example, the LAG instruction is executed where MF00000 is the input value in the parameter table, MF00002 is the output value, and the first-order lag time constant is set to 1.0.




MF00002 changes as shown below when the input value (MF00000) changes from 0 to 10,000.




MF00002 changes as shown below when the input value (MF00000) changes from 0 to -10,000.



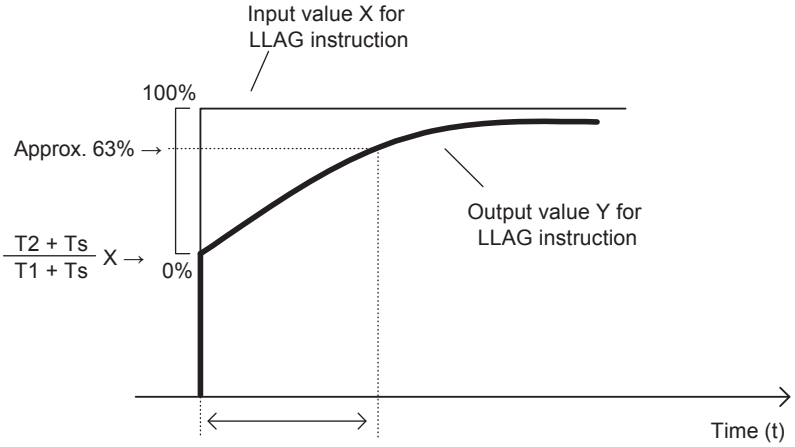
# 5.8.8 Phase Lead Lag (LLAG)

## ( 1 ) Operation

The LLAG instruction calculates the phase lead and lag according to predefined parameters in a parameter table. The input value to the LLAG instruction can be an integer or real number. Double-length integers cannot be used.
The structure of the parameter table is different for integers and real numbers.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.




Phase lag time constant T1

The LLAG operation in the figure shown above can be expressed by the formula shown below.

$$\frac{Y(s)}{X(s)} = \frac{1 + T2 \times s}{1 + T1 \times s}$$

Therefore,

$$T1 \times \frac{dY}{dt} + Y = T2 \times \frac{dX}{dt} + X$$

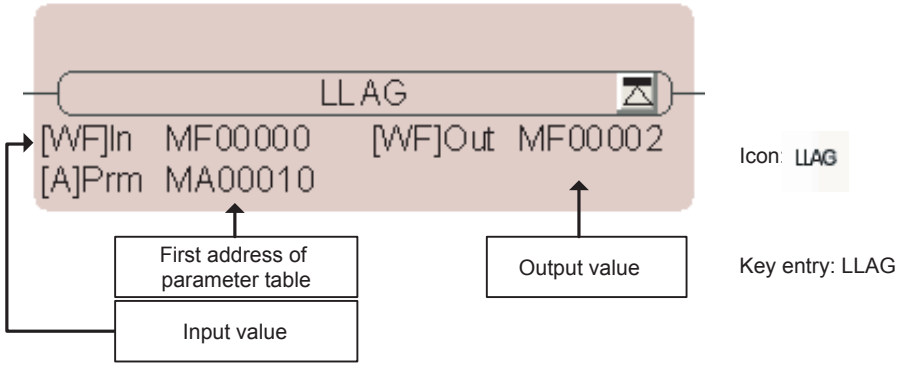
The following operation is performed internally by the LLAG instruction, where dt = Ts, dY = Y – Y', and dX = X – X'.
In the formula shown below, Y' is the previous output value, X' is the previous input value, Ts is the scan time set value*, and REM is the remainder.

* The unit for Ts is the same as the unit for T1.

$$Y = \frac{T1 \times Y' + (T2 + Ts) \times X - T2 \times X' + REM}{T1 + Ts}$$

- When IRST (LLAG reset) is closed, Y outputs 0, REM outputs 0, and X outputs 0.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○* | ○ | ○ |
| Output value (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for LLAG Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | T2 | Phase lead time constant | Phase lead time constant (ms) | IN |
| 2 | W | T1 | Phase lag time constant | Phase lag time constant (ms) | IN |
| 3 | W | Y | LLAG output | LLAG output (output to Out) | OUT |
| 4 | W | REM | Remainder | Remainder storage | OUT |
| 5 | W | X | Input value storage | Input value storage | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | IRST | LLAG reset bit | This input is closed to reset the LLAG operation. | IN |
| 1 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 to F | – | (Reserved.) | Spare output relays | OUT |

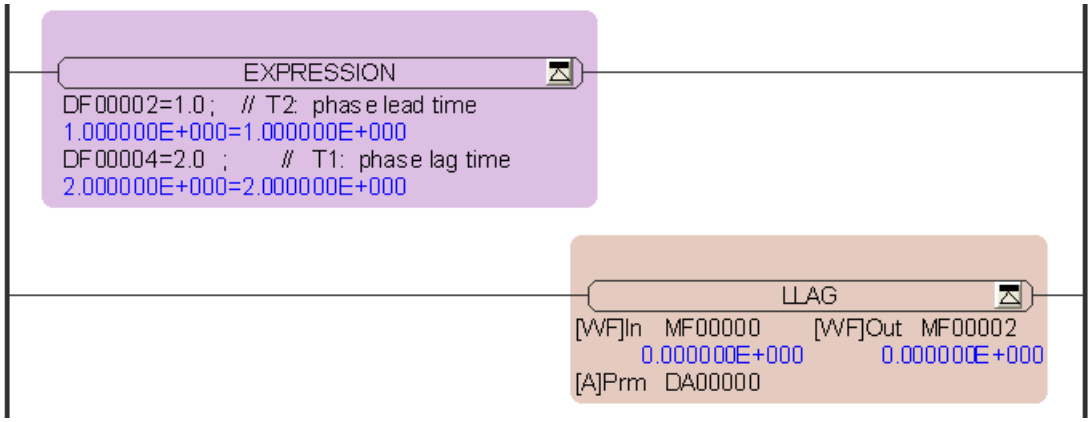
### [ b ] Parameter Table Configuration for LLAG Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | T2 | Phase lead time constant | Phase lead time constant (s) | IN |
| 4 | F | T1 | Phase lag time constant | Phase lag time constant (s) | IN |
| 6 | F | Y | LLAG output | LLAG output (output to Out) | OUT |
| 8 | F | X | Input value storage | Input value storage | OUT |

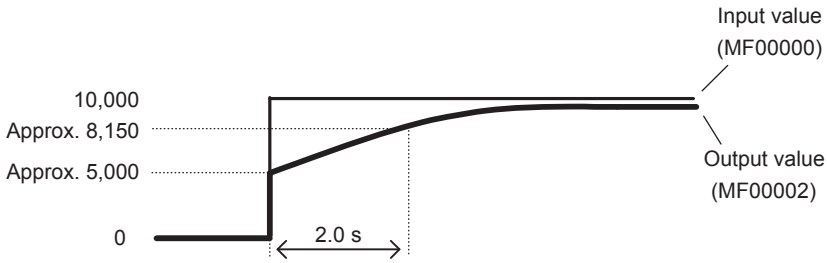
* The relay input and output bit assignments are the same as for integers.

## ( 3 ) Programming Example

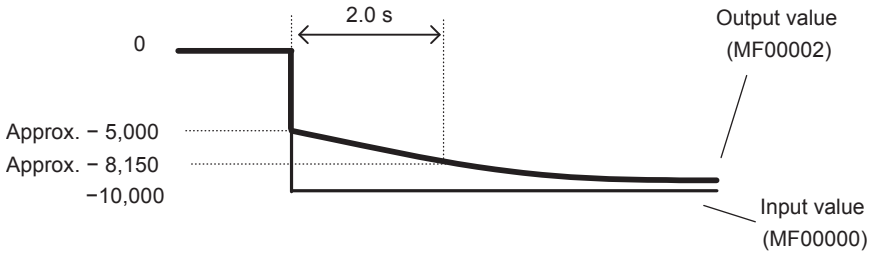
In the following programming example, the LLAG instruction is executed where MF00000 is the input value, MF00002 is the output value, the phase lead time constant is set to 1.0 seconds, and the phase lag time constant is set to 2.0 seconds.




MF00002 changes as shown below when the input value (MF00000) changes from 0 to 10,000.



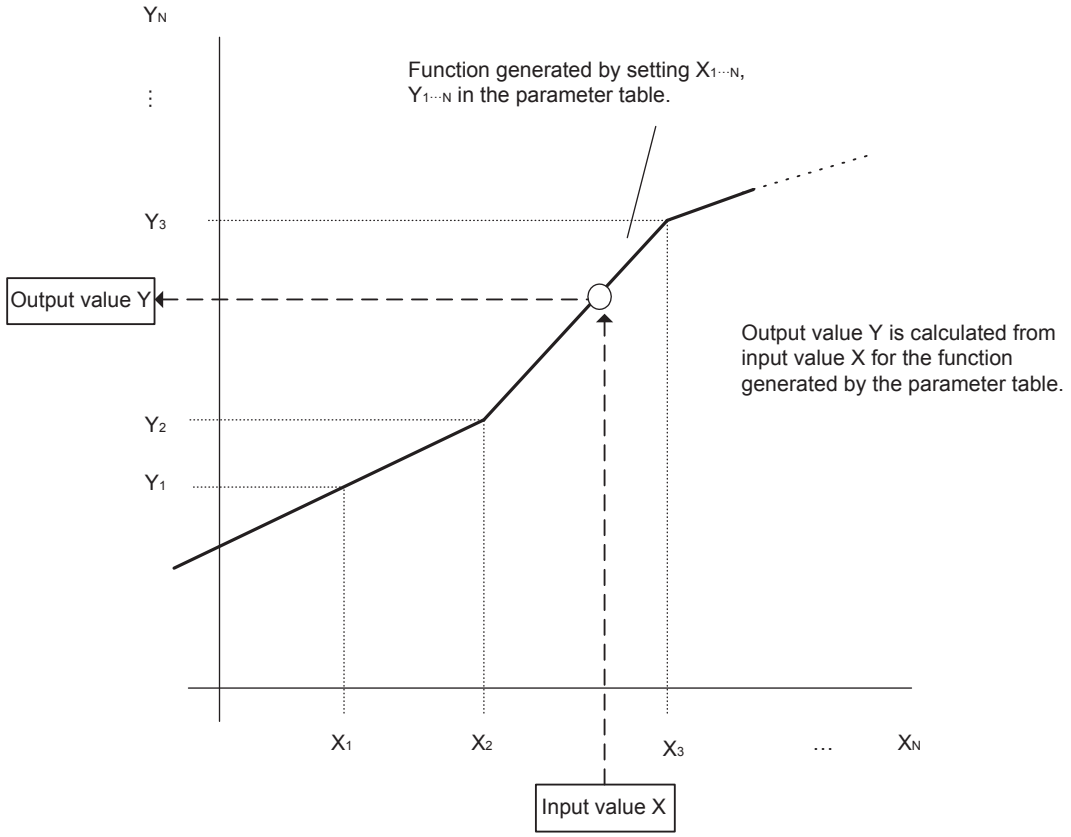

MF00002 changes as shown below when the input value (MF00000) changes from 0 to -10,000.



# 5.8.9 Function Generator (FGN)

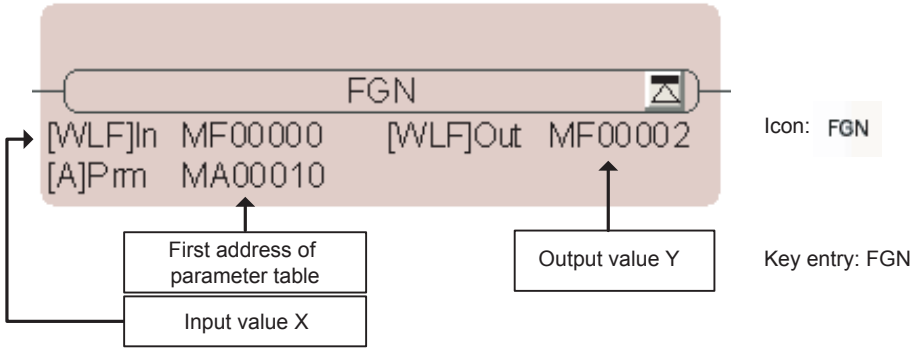
## ( 1 ) Operation

The FGN instruction generates a function based on the parameters specified in the parameter table. It then uses the function to calculate output value Y based on the value of input X.
The FGN instruction will be for integers, double-length integers, or real numbers, depending on the data type of input value X. The structure of the parameter table changes accordingly.




- Create the parameter table so that $X_1 < X_2 < ... < X_N$.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value X (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output value Y (Out) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for FGN Instruction with Integers

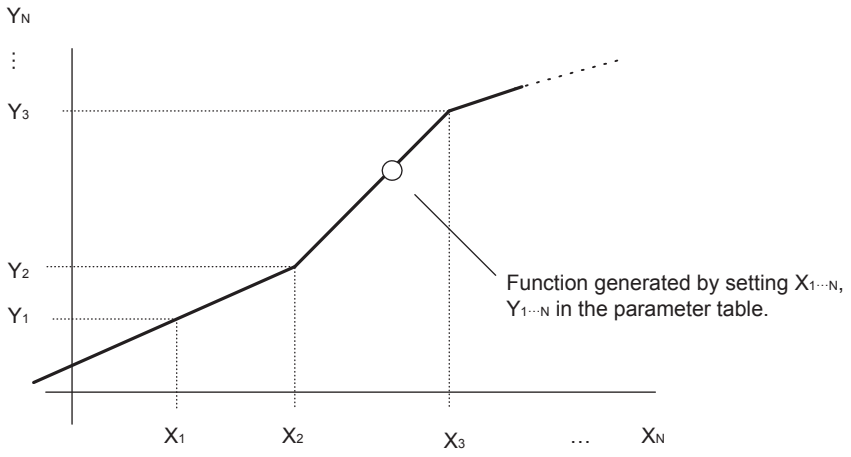
If input value X is an integer, the FGN instruction will be for integers.
Create the parameter table as shown below.

| Address | Data Type | Symbol | Name |
|---|---|---|---|
| 0 | W | N | Number of pairs of X and Y |
| 1 | W | $X_1$ | Data $X_1$ |
| 2 | W | $Y_1$ | Data $Y_1$ |
| 3 | W | $X_2$ | Data $X_2$ |
| 4 | W | $Y_2$ | Data $Y_2$ |
| : | : | : | : |
| 2N - 1 | W | $X_N$ | Data $X_N$ |
| 2N | W | $Y_N$ | Data $Y_N$ |

### [ b ] Parameter Table Configuration for FGN Instruction with Double-length Integers or Real Numbers

If input value X is a double-length integer, the FGN instruction will be for double-length integers. If input value X is a real number, the FGN instruction will be for real numbers.
Create the parameter table as shown below.

| Address | Data Type | Symbol | Name |
|---|---|---|---|
| 0 | W | N | Number of pairs of X and Y |
| 1 | W | – | Reserved. |
| 2 | L/F | $X_1$ | Data $X_1$ |
| 4 | L/F | $Y_1$ | Data $Y_1$ |
| 6 | L/F | $X_2$ | Data $X_2$ |
| 8 | L/F | $Y_2$ | Data $Y_2$ |
| : | : | : | : |
| 4N - 2 | L/F | $X_N$ | Data $X_N$ |
| 4N | L/F | $Y_N$ | Data $Y_N$ |



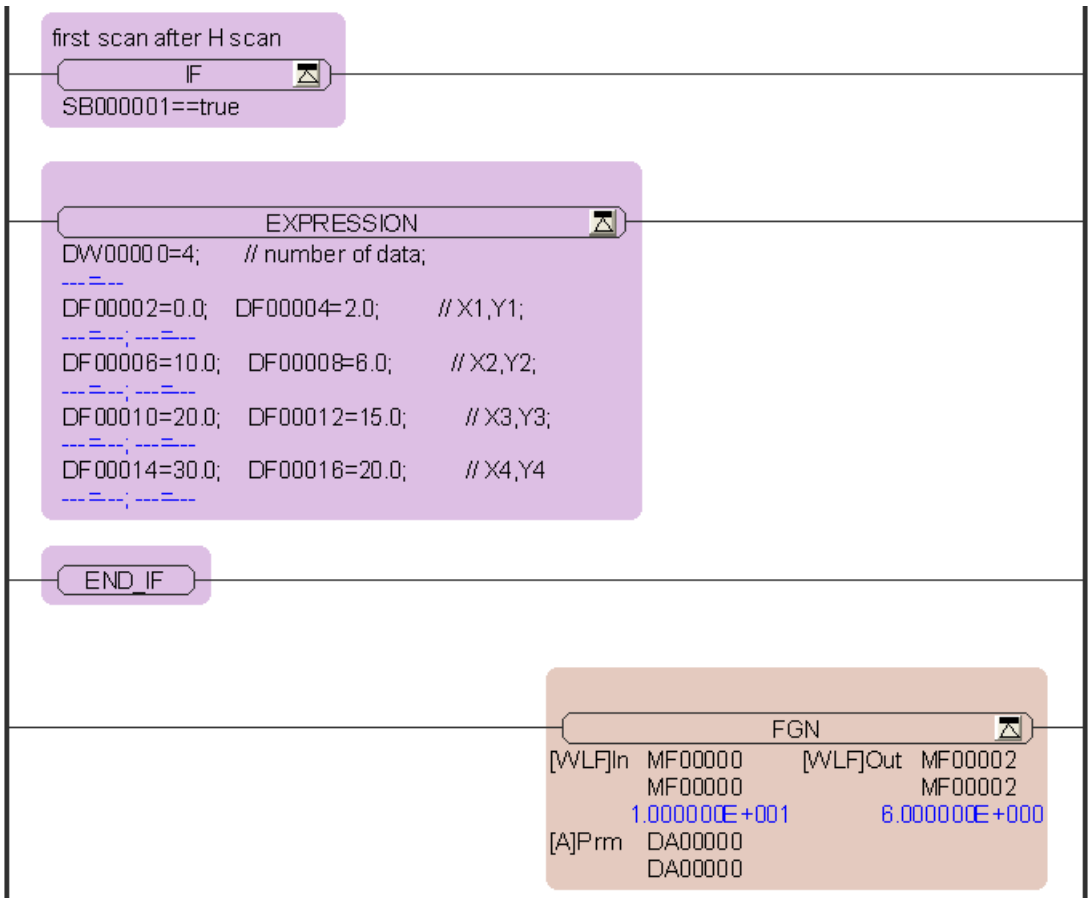

INFO

For the FGN instruction, make sure to set the data so that $X_1 < X_2 < \cdots < X_N$, regardless of whether the parameter table is for integer data, double-length integer data, or real number data.

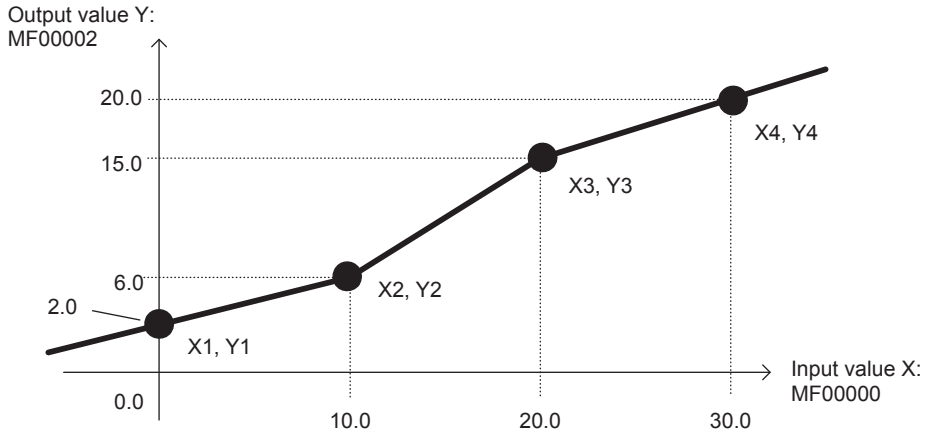
## ( 3 ) Programming Example

In the following programming example, the function is generated using the FGN instruction for real numbers with the parameter table given below.

| Number of Pairs | 4 |
|---|---|
| X1, Y1 | 0.0, 2.0 |
| X2, Y2 | 10.0, 6.0 |
| X3, Y3 | 20.0, 15.0 |
| X4, Y4 | 30.0, 20.0 |




The following figure shows the relationship between input value X in MF00000 and output value Y in MF00002.



## ( 4 ) Additional Information

The FGN instruction searches for the pair $X_n$ and $Y_n$ where $X_n \leq$ Input $X \leq X_n + 1$ to calculate output value Y.

$$\boxed{\text{Output value Y}} = Y_n + \frac{Y_{n+1} - Y_n}{X_{n+1} - X_n} \times \left( \boxed{\text{Input value X}} - X_n \right) \quad (1 \leq n \leq N - 1)$$

If the pair $X_n$ and $Y_n$, where $X_n \leq$ Input $X \leq X_{n+1}$, does not exist, the calculation is as follows:

- If Input value $X < X_1$,

$$\boxed{\text{Output value Y}} = Y_1 + \frac{Y_2 - Y_1}{X_2 - X_1} \times \left( \boxed{\text{Input value X}} - X_1 \right)$$

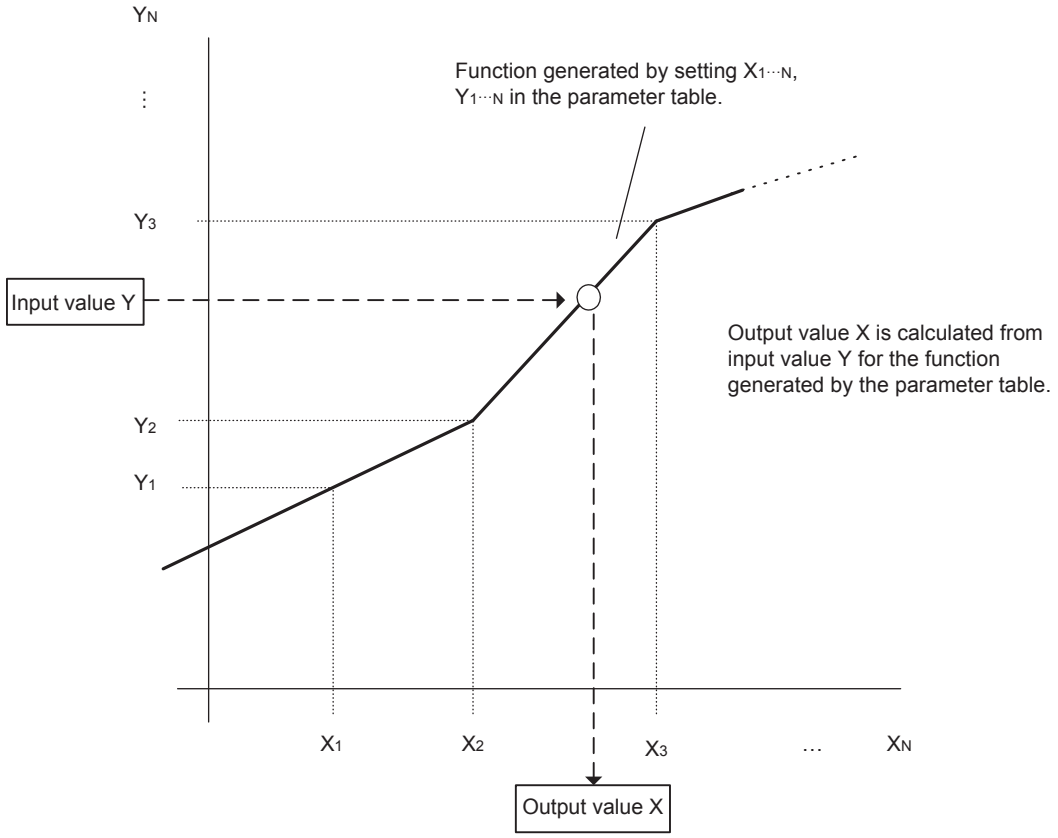
- If Input value $X > X_N$,

$$\boxed{\text{Output value Y}} = Y_N + \frac{Y_N - Y_{N-1}}{X_N - X_{N-1}} \times \left( \boxed{\text{Input value X}} - X_N \right)$$

# 5.8.10 Inverse Function Generator (IFGN)

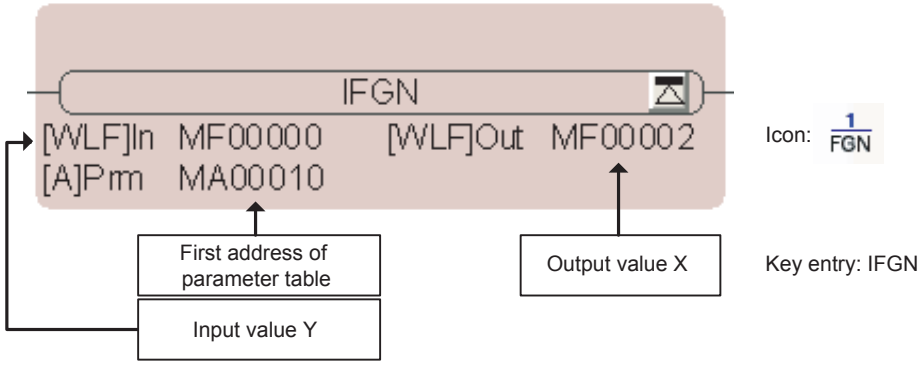
## ( 1 ) Operation

The IFGN instruction generates a function based on the parameters specified in the parameter table in the same way as the FGN instruction.
It then uses the function to calculate output value X based on the value of input Y, i.e., the opposite direction from the FGN instruction.
The structure of the parameter table is the same as for the FGN instruction.




- Create the parameter table so that $Y_1 < Y_2 < ... < Y_N$.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value Y (In) | × | ○ | ○ | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output value X (Out) | × | ○* | ○* | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for IFGN Instruction with Integers

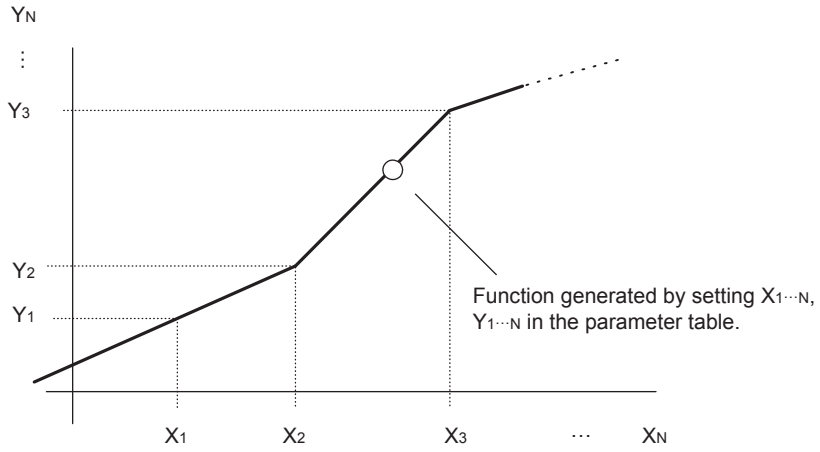
If input value Y is an integer, the IFGN instruction will be for integers.
Create the parameter table as shown below.

| Address | Data Type | Symbol | Name |
|---|---|---|---|
| 0 | W | N | Number of pairs of X and Y |
| 1 | W | $X_1$ | Data $X_1$ |
| 2 | W | $Y_1$ | Data $Y_1$ |
| 3 | W | $X_2$ | Data $X_2$ |
| 4 | W | $Y_2$ | Data $Y_2$ |
| : | : | : | : |
| 2N - 1 | W | $X_N$ | Data $X_N$ |
| 2N | W | $Y_N$ | Data $Y_N$ |

### [ b ] Parameter Table Configuration for IFGN Instruction with Double-length Integers or Real Numbers

If input value Y is a double-length integer, the IFGN instruction will be for double-length integers. If input value Y is a real number, the IFGN instruction will be for real numbers.
Create the parameter table as shown below.

| Address | Data Type | Symbol | Name |
|---|---|---|---|
| 0 | W | N | Number of pairs of X and Y |
| 1 | W | – | Reserved. |
| 2 | L/F | $X_1$ | Data $X_1$ |
| 4 | L/F | $Y_1$ | Data $Y_1$ |
| 6 | L/F | $X_2$ | Data $X_2$ |
| 8 | L/F | $Y_2$ | Data $Y_2$ |
| : | : | : | : |
| 4N - 2 | L/F | $X_N$ | Data $X_N$ |
| 4N | L/F | $Y_N$ | Data $Y_N$ |



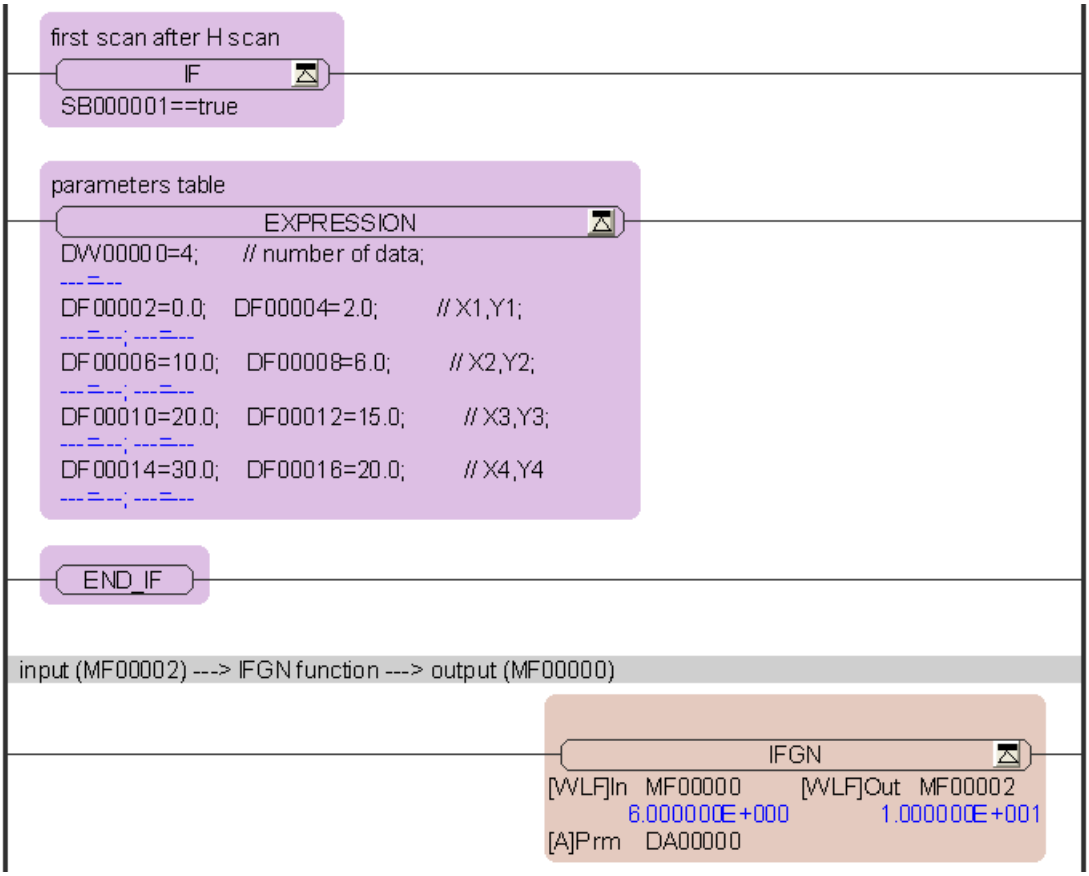

INFO

For the IFGN instruction, make sure to set the data so that $Y_1 < Y_2 < \cdots < Y_N$, regardless of whether the parameter table is for integer data, double-length integer data, or real number data.

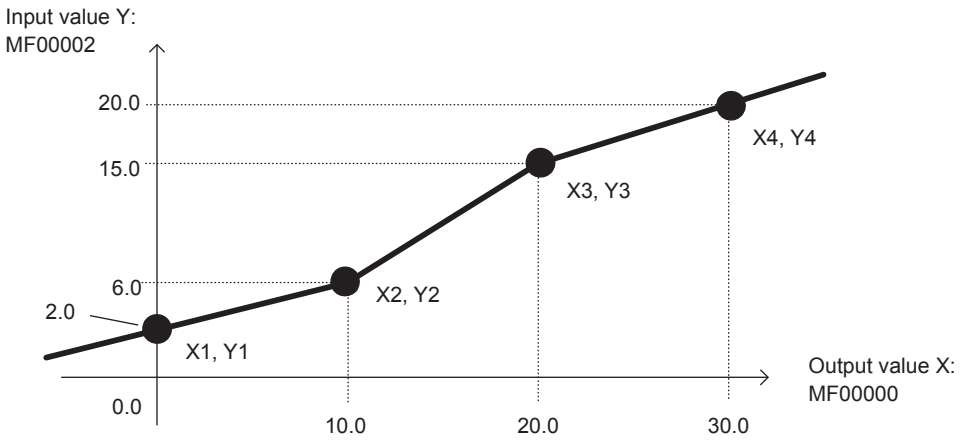
## ( 3 ) Programming Example

In the following programming example, the function is generated using the IFGN instruction for real numbers with the parameter table given below.

| Number of Pairs | 4 |
|---|---|
| X1, Y1 | 0.0, 2.0 |
| X2, Y2 | 10.0, 6.0 |
| X3, Y3 | 20.0, 15.0 |
| X4, Y4 | 30.0, 20.0 |




The following figure shows the relationship between input value Y in MF00002 and output value X in MF00000.



## ( 4 ) Additional Information

The IFGN instruction searches for the pair $X_n$ and $Y_n$ where $Y_n \leq$ Input $Y \leq Y_n + 1$ to calculate output value X.

$$\boxed{\text{Output value X}} = X_n + \frac{X_{n+1} - X_n}{Y_{n+1} - Y_n} \times (\boxed{\text{Input value Y}} - Y_n) \; (1 \leq n \leq N - 1)$$

If the pair $X_n$ and $Y_n$, where $Y_n \leq$ Input $Y \leq Y_{n+1}$, does not exist, the calculation is as follows:

- If Input value $Y < Y_1$,

$$\boxed{\text{Output value X}} = X_1 + \frac{X_2 - X_1}{Y_2 - Y_1} \times (\boxed{\text{Input value Y}} - Y_1)$$

- If Input value $Y > Y_N$,

$$\boxed{\text{Output value X}} = X_N + \frac{X_N - X_{N-1}}{Y_N - Y_{N-1}} \times (\boxed{\text{Input value Y}} - Y_N)$$

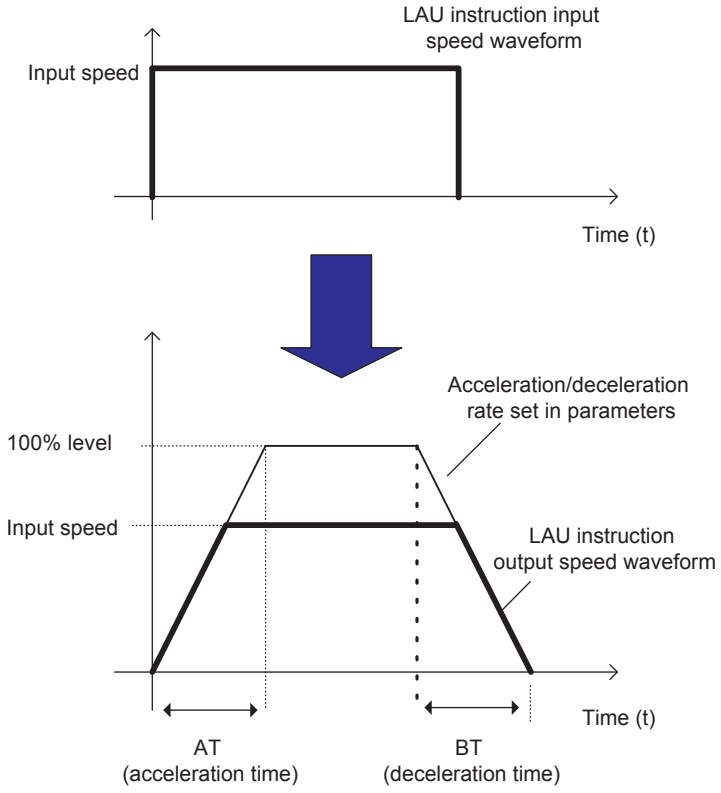
## 5.8.11 Linear Accelerator/Decelerator 1 (LAU)

### ( 1 ) Operation

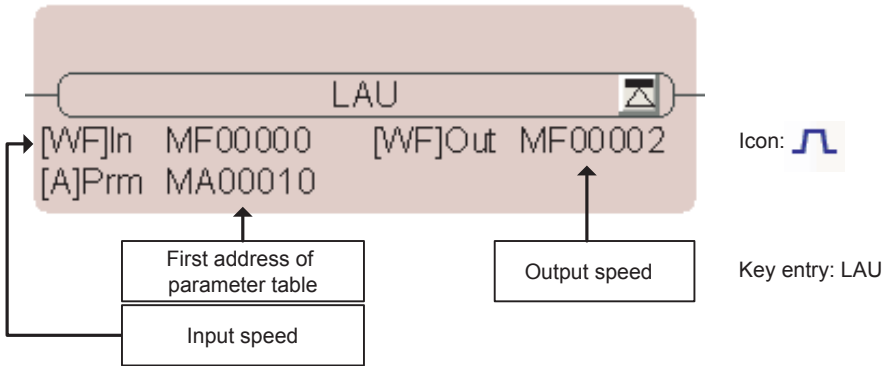
The LAU instruction outputs the speed that results from applying a constant acceleration or deceleration rate to the input speed. The acceleration or deceleration rate is applied according to predefined parameters in a parameter table. The input value to the LAU instruction can be an integer or a real number. Double-length integers cannot be used. The structure of the parameter table is different for integers and real numbers.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.



## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input speed (In) | × | ○ | × | ○ | × | ○ | ○ |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output speed (Out) | × | ○* | × | ○* | × | ○ | × |

* C and # registers cannot be used.

### [ a ] Parameter Table Configuration for LAU Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | LV | 100% level of input | Scale for 100% input | IN |
| 2 | W | AT | Acceleration time | Time to accelerate from 0% to 100% (0.1 s) | IN |
| 3 | W | BT | Deceleration time | Time to decelerate from 100% to 0% (0.1 s) | IN |
| 4 | W | QT | Quick stop time | Time to make a quick stop from 100% to 0% (0.1 s) | IN |
| 5 | W | V | Current speed | LAU output (output to Out) | OUT |
| 6 | W | DVDT | Current acceleration/ deceleration rate | Scaling with the normal acceleration rate set to 5,000 | OUT |
| 7 | W | – | (Reserved.) | Spare register | – |
| 8 | W | VIM | Previous speed reference | For storage of the previous speed reference input value | OUT |
| 9 | W | DVDTK | DVDT coefficient | Scaling factor for DVDT (Current Acceleration Rate) | IN |
| 10 | L | REM | Remainder | Remainder of the acceleration/deceleration rate | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | RN | Line running | This input is closed to run the line. | IN |
| 1 | QS | Quick stop | This input is opened to execute a quick stop. | IN |
| 2 | DVDTF | Skip execution of DVDT operation | This input is closed to skip execution of the DVDT operation. | IN |
| 3 | DVDTS | DVDT operation selection | Selects the method for calculating DVDT | IN |
| 4 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | ARY | Accelerating | This output is closed during acceleration. | OUT |
| 9 | BRY | Decelerating | This output is closed during deceleration. | OUT |
| A | LSP | Zero speed | This output is closed during zero speed. | OUT |
| B | EQU | Equal | This output is closed when the input speed equals the output speed. | OUT |
| C to F | – | (Reserved.) | Spare output relays | OUT |

- If QS (quick stop) is opened, QT (quick stop time) is used as the acceleration/deceleration time.

## [ b ] Parameter Table Configuration for LAU Instruction with Real Numbers

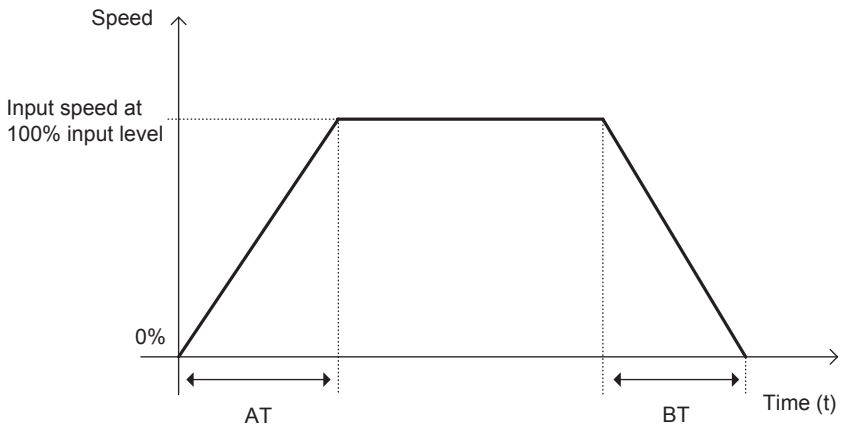
| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | LV | 100% level of input | Scale for 100% input | IN |
| 4 | F | AT | Acceleration time | Time to accelerate from 0% to 100% (s) | IN |
| 6 | F | BT | Deceleration time | Time to decelerate from 100% to 0% (s) | IN |
| 8 | F | QT | Quick stop time | Time to make a quick stop from 100% to 0% (s) | IN |
| 10 | F | V | Current speed | LAU output (output to Out) | OUT |
| 12 | F | DVDT | Current acceleration/ deceleration rate | The current acceleration or deceleration rate is output. | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | RN | Line running | This input is closed to run the line. | IN |
| 1 | QS | Quick stop | This input is opened to execute a quick stop. | IN |
| 2 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | ARY | Accelerating | This output is closed during acceleration. | OUT |
| 9 | BRY | Decelerating | This output is closed during deceleration. | OUT |
| A | LSP | Zero speed | This output is closed during zero speed. | OUT |
| B | EQU | Equal | This output is closed when the input speed equals the output speed. | OUT |
| C to F | – | (Reserved.) | Spare output relays | OUT |

- If QS (quick stop) is opened, QT (quick stop time) is used as the acceleration/deceleration time.

Output Speed Waveform When Input Speed Is at the 100% Input Level (LV)




The acceleration time (AT) is the time from the 0% speed to the 100% speed. The deceleration time (BT) is the time from the 100% speed to the 0% speed. The 100% speed is set as the 100% level of input. The setting of this parameter determines the acceleration/deceleration rate. When the input speed is applied, operation is performed at the acceleration/deceleration rate.
Therefore, the ratio between the set value for LV (input at 100% level) and the input speed determines the actual acceleration/deceleration time.
Refer to *( 4 ) Additional Information* for details on the processing that is performed internally by the LAU instruction.

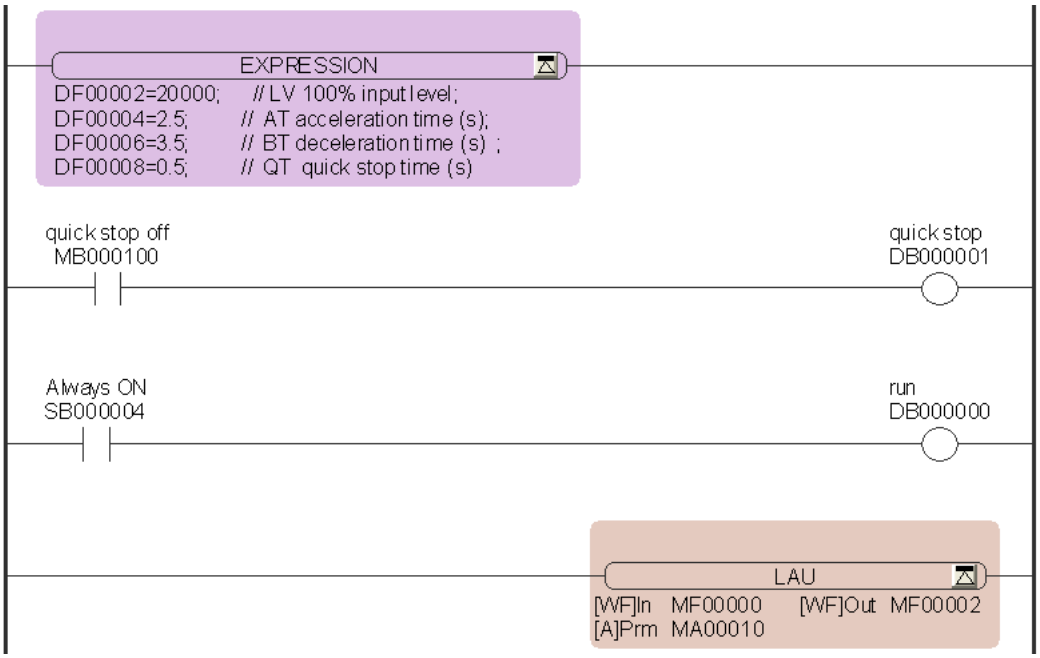
INFO

When QS (quick stop) opens (OFF), the acceleration/deceleration time is set to the QT (quick stop time).
To execute a quick stop, open (OFF) QS (quick stop) and set the input speed to 0 at the same time.

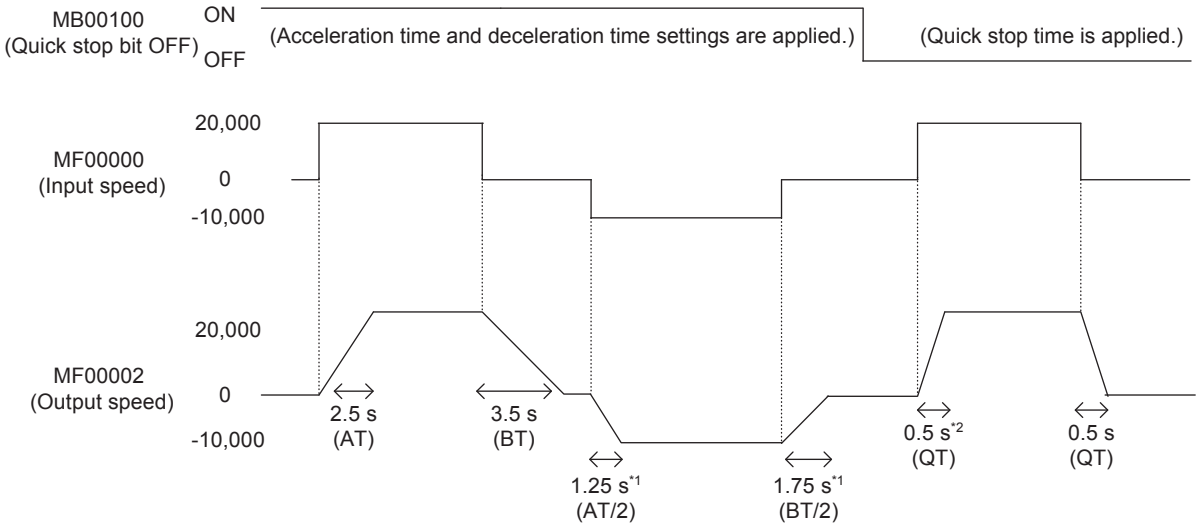
## ( 3 ) Programming Example

In the following programming example, the LAU instruction for real numbers is executed with the specified acceleration and deceleration rates where MF00000 is the input speed and MF00002 is the output speed.
The following parameters are set with an EXPRESSION instruction to create the acceleration or deceleration rate.

- 100% level of acceleration/deceleration rate input = 20,000
- Acceleration time = 2.5 s
- Deceleration time = 3.5 s
- Quick stop time = 0.5 s




The following figure shows how each register operates.




∗ 1. The acceleration time is applied when moving away from 0, and the deceleration time is applied when moving toward 0.
∗ 2. The quick stop time is also applied as the acceleration time.

## ( 4 ) Additional Information

This information applies when the LAU instruction is used for integer or real number data.

### [ a ] LAU Instruction with Integers

The LAU instruction for integers calculates the speed output value during acceleration, deceleration, and quick stops, and the current acceleration or deceleration rates using the formula shown below based on predefined parameters.
In this formula, V is the speed output value, V' is the previous speed output value, VI is the input value for the speed reference, and Ts is the scan time set value.

#### ■ Speed Output Value during Acceleration

The speed output value during acceleration is calculated as follows:

If VI > V' (V' ≥ 0), then V = V' + ADV.
If VI < V' (V' ≤ 0), then V = V' − ADV.

$$\text{ADV (acceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM}}{\text{AT (0.1 s)} \times 1{,}000}$$

#### ■ Speed Output Value during Deceleration

The speed output value during deceleration is calculated as follows:

If VI > V' (V' < 0), then V = V' + BDV.
If VI < V' (V' > 0), then V = V' − BDV.

$$\text{BDV (deceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM}}{\text{BT (0.1 s)} \times 1{,}000}$$

#### ■ Speed Output Value during a Quick Stop

The speed output value during a quick stop is calculated as follows:

If QS = OFF (VI > V', V' < 0), then V = V' + QDV.
If QS = OFF (VI < V', V' > 0), then V = V' − QDV.

$$\text{QDV (Quick Stop Rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM}}{\text{QT (0.1 s)} \times 1{,}000}$$

#### ■ Current Acceleration/Deceleration Rate

If DVDTF is ON, DVDT (current acceleration/deceleration rate) will be calculated according to the setting of DVDTS (DVDT operation selection) using one of the following formulas. If DVDTF is OFF, DVDT is set to 0.

$$\text{If DVDTS = ON, DVDT} = \frac{(\text{V} - \text{V}') \times 5{,}000}{\text{ADV}}$$

$$\text{If DVDTS = ON, DVDT} = (\text{V} - \text{V}') \times \text{DVDTK}$$

INFO

ARY (accelerating) turns ON when V' ≥ 0 and ADV > 0, or when V' ≤ 0 and ADV < 0.
BRY (decelerating) turns ON at the following times:
- When V' < 0 and BDV > 0, or when V' > 0 and BDV < 0
- When V' < 0 and QDV > 0, or when V' > 0 and QDV < 0

LSP (zero speed) turns ON when V equals 0. EQU (equal) turns ON when VI equals V.
If RN (line running) is opened (OFF), the outputs for V, DVDT, and REM are set to 0.

### [ b ] LAU Instruction for Real Numbers

The LAU instruction for real numbers calculates the speed output value during acceleration, deceleration, and quick stops, and the current acceleration or deceleration rates using the formula shown below based on predefined parameters.
In this formula, V is the speed output value, V' is the previous speed output value, VI is the input value for the speed reference, and Ts is the scan time set value.

#### ■ Speed Output Value during Acceleration

The speed output value during acceleration is calculated as follows:

If VI > V' (V' ≥ 0), then V = V' + ADV.
If VI < V' (V' ≤ 0), then V = V' − ADV.

$$\text{ADV (acceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)}}{\text{AT (s)} \times 10{,}000}$$

#### ■ Speed Output Value during Deceleration

The speed output value during deceleration is calculated as follows:

If VI < V' (V' > 0), then V = V' + BDV.
If VI > V' (V' < 0), then V = V' − BDV.

$$\text{BDV (deceleration rate)} = \frac{-\text{LV} \times \text{Ts (0.1 ms)}}{\text{BT (s)} \times 10{,}000}$$

#### ■ Speed Output Value during a Quick Stop

The speed output value during a quick stop is calculated as follows:

If QS = OFF (VI < V', V' > 0), then V = V' + QDV.
If QS = OFF (VI > V', V' < 0), then V = V' − QDV.

$$\text{QDV (Quick Stop Rate)} = \frac{-\text{LV} \times \text{Ts (0.1 ms)}}{\text{QT (s)} \times 10{,}000}$$

#### ■ Current Acceleration/Deceleration Rate

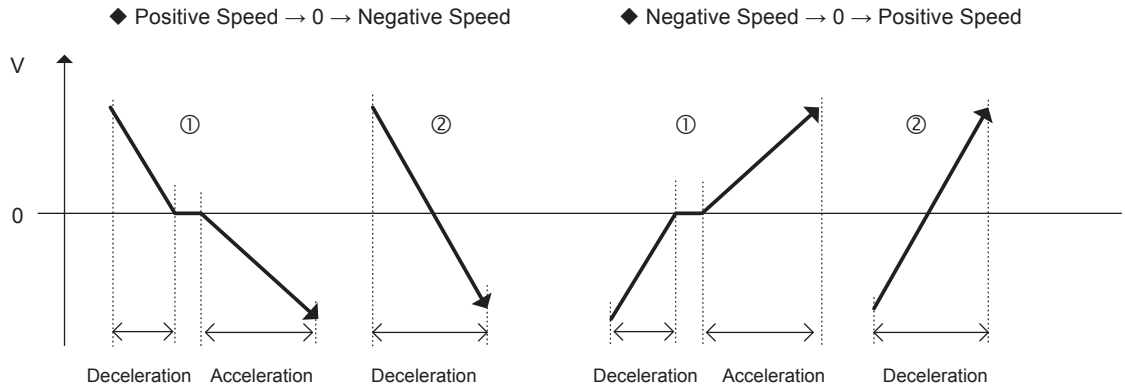
The DVDT (current acceleration/deceleration rate) is calculated as follows after V (speed output) has been calculated:
DVDT = V - V'

ARY (accelerating) turns ON when V' ≥ 0 and ADV > 0, or when V' ≤ 0 and ADV < 0.
BRY (decelerating) turns ON at the following times:
- When V' < 0 and BDV > 0, or when V' > 0 and BDV < 0
- When V' < 0 and QDV > 0, or when V' > 0 and QDV < 0

LSP (zero speed) turns ON when V equals 0. EQR (equal) turns ON when VI equals V.
ARY (accelerating) turns ON when V ≠ V' and DVDT and V have the same sign and BRY (decelerating) turns ON when V ≠ V' and DVDT and V do not have the same sign.
If RN (line running) is opened (OFF), the outputs for V and DVDT are set to 0.

### [ c ] Precaution When Input Speed Changes Across a Speed of 0

If a reference is input that causes the speed to cross a speed of 0, the output changes as shown by ② in the following figure.




① If operation stops at a speed of 0, operation will proceed according to the set deceleration and acceleration times.

② If the speed reference crosses the point where speed equals 0, operation is controlled by the deceleration time so that the speed does not fluctuate.

# 5.8.12 Linear Accelerator/Decelerator 2 (SLAU)

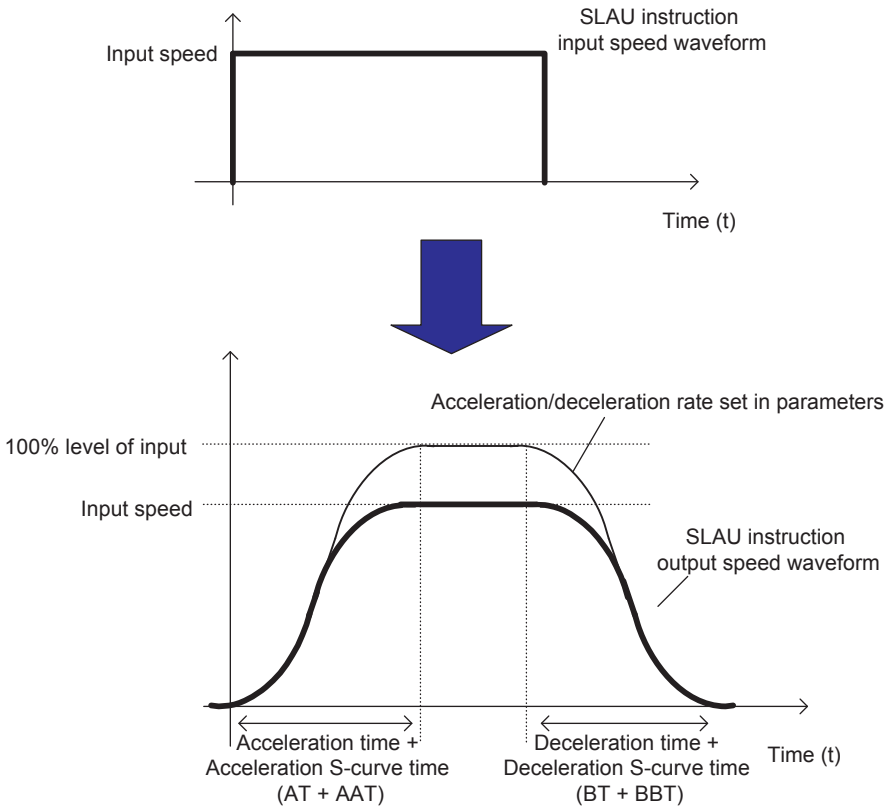
## ( 1 ) Operation

The SLAU instruction outputs the speed that results from applying a variable acceleration or deceleration rate to the input speed. Operation for the acceleration or deceleration rate is performed in an S curve according to predefined parameters in a parameter table.
The input value to the SLAU instruction can be an integer or a real number.
The structure of the parameter table depends on the data type.

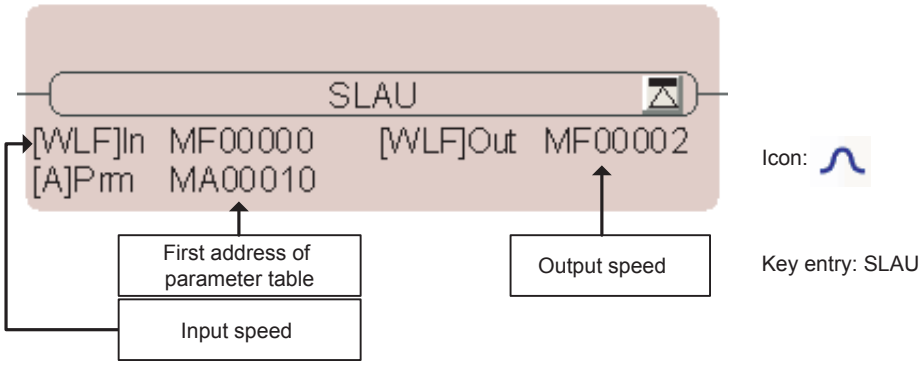
- Double-length integers can be used only for CPU software version 2.30 or higher. For earlier versions, the lower 16 bits of the double-length integer are used in the calculations as an integer.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.



## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input speed (In) | × | ○ | ○*1 | ○ | × | ○ | × |
| First address of parameter table (Prm) | × | × | × | × | ○*2 | ○ | ○ |
| Output speed (Out) | × | ○*2 | ○*1, 2 | ○*2 | × | ○ | × |

∗ 1. This data type can be used only for version 2.30 or higher. For earlier versions, the lower 16 bits of the double-length integer are used in the calculations as an integer.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration for SLAU Instruction with Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | LV | 100% level of input | Scale for 100% input | IN |
| 2 | W | AT | Acceleration time | Time to accelerate from 0% to 100% (0.1 s) | IN |
| 3 | W | BT | Deceleration time | Time to decelerate from 100% to 0% (0.1 s) | IN |
| 4 | W | QT | Quick stop time | Time to make a quick stop from 100% to 0% (0.1 s) | IN |
| 5 | W | AAT | Acceleration S-curve time | Acceleration S-curve region time (0.01 to 32.00 s) | IN |
| 6 | W | BBT | Deceleration S-curve time | Deceleration S-curve region time (0.01 to 32.00 s) | IN |
| 7 | W | V | Current speed | SLAU output (output to Out) | OUT |
| 8 | W | DVDT1 | Current acceleration/ deceleration rate 1 | Scaling with the normal acceleration rate set to 5,000 | OUT |
| 9 | W | – | (Reserved.) | Spare register | – |
| 10 | W | ABMD | Speed increase when holding | Amount of speed change until the speed stabilizes after the hold command is executed | OUT |
| 11 | W | REM1 | Remainder | Remainder of the acceleration/deceleration rate | OUT |
| 12 | W | – | (Reserved.) | Spare register | – |
| 13 | W | VIM | Previous speed reference | For storage of the previous speed reference input value | OUT |
| 14 | L | DVDT2 | Current acceleration/ deceleration rate 2 | 1,000 times the actual acceleration/deceleration | OUT |
| 16 | L | DVDT3 | Current acceleration/ deceleration rate 3 | Current acceleration/deceleration rate ( = DVDT2/ 1,000) | OUT |
| 18 | L | REM2 | Remainder | Remainder of the S-curve region acceleration/deceleration rate | OUT |
| 20 | W | REM3 | Remainder | Remainder of the current speed | OUT |
| 21 | W | DVDTK | DVDT1 coefficient | Scaling factor for DVDT (Current acceleration rate 1) (-32,768 to 32,767) | OUT |

∗ The relay input and output bits are assigned as follows. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | RN | Line running | This input is closed to run the line. | IN |
| 1 | QS | Quick stop | This input is opened to execute a quick stop. | IN |
| 2 | DVDTF | Skip execution of DVDT1 operation | This input is closed to skip execution of the DVDT operation. | IN |
| 3 | DVDTS | DVDT1 operation selection | Selects the method for calculating DVDT | IN |
| 4 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | ARY | Accelerating | This output is closed during acceleration. | OUT |
| 9 | BRY | Decelerating | This output is closed during deceleration. | OUT |
| A | LSP | Zero speed | This output is closed during zero speed. | OUT |
| B | EQU | Equal | This output is closed when the input speed equals the output speed. | OUT |
| C | – | (Reserved.) | Spare output relay | OUT |
| D | CCF | Work relay | System internal work relay | OUT |
| E | BBF | Work relay | System internal work relay | OUT |
| F | AAF | Work relay | System internal work relay | OUT |

- If QS (quick stop) is opened, QT (quick stop time) is used as the acceleration/deceleration time.

### [ b ] Parameter Table Configuration for SLAU Instruction with Double-length Integers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs*1 | IN/OUT |
| 1 | W | – | (Reserved.) | – | – |
| 2 | L | LV | 100% level of input | Scale for 100% of input value | IN |
| 4 | L | AT | Acceleration time | Time to accelerate from 0% to 100% (0.1 s) | IN |
| 6 | L | BT | Deceleration time | Time to decelerate from 100% to 0% (0.1 s) | IN |
| 8 | L | QT | Quick stop time | Time to make a quick stop from 100% to 0% (0.1 s) | IN |
| 10 | L | AAT | Acceleration S-curve time | Acceleration S-curve region time (0.01 s) | IN |
| 12 | L | BBT | Deceleration S-curve time | Deceleration S-curve region time (0.01 s) | IN |
| 14 | L | V | Current speed | SLAU output (also the output to the A register) | OUT |
| 16 | L | DVDT | Current acceleration/ deceleration rate | The current acceleration or deceleration rate is output. (The output is truncated below the decimal point.)*2 | OUT |
| 18 | L | ABMD | Speed increase when holding | Amount of speed change until the speed stabilizes after the hold command is executed | OUT |
| 20 | D | V_D | Current speed | SLAU output for system use (double-precision real number) | IN/OUT |
| 24 | D | DVDT_D | Current acceleration/ deceleration rate | Current acceleration or deceleration rate for system use (double-precision real number) | IN/OUT |

∗ 1. D is a double-length real number expressed in four words. The MPE720 cannot display this value as a real number.
∗ 2. The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | RN | Line running | This input is closed to run the line. | IN |
| 1 | QS | Quick stop | This input is opened to execute a quick stop. | IN |
| 2 | DVDTF | Acceleration/deceleration rate flag | When the input is closed, DVDT (current acceleration/ deceleration rate) is multiplied by 1,000 and then output. | IN |
| 3 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | ARY | Accelerating | This output is closed during acceleration. | OUT |
| 9 | BRY | Decelerating | This output is closed during deceleration. | OUT |
| A | LSP | Zero speed | This output is closed during zero speed. | OUT |
| B | EQU | Equal | This output is closed when the input value equals the output value. | OUT |
| C to F | – | Work relays | System internal work relays | IN/OUT |

- If QS (quick stop) is opened, QT (quick stop time) is used as the acceleration/deceleration time.

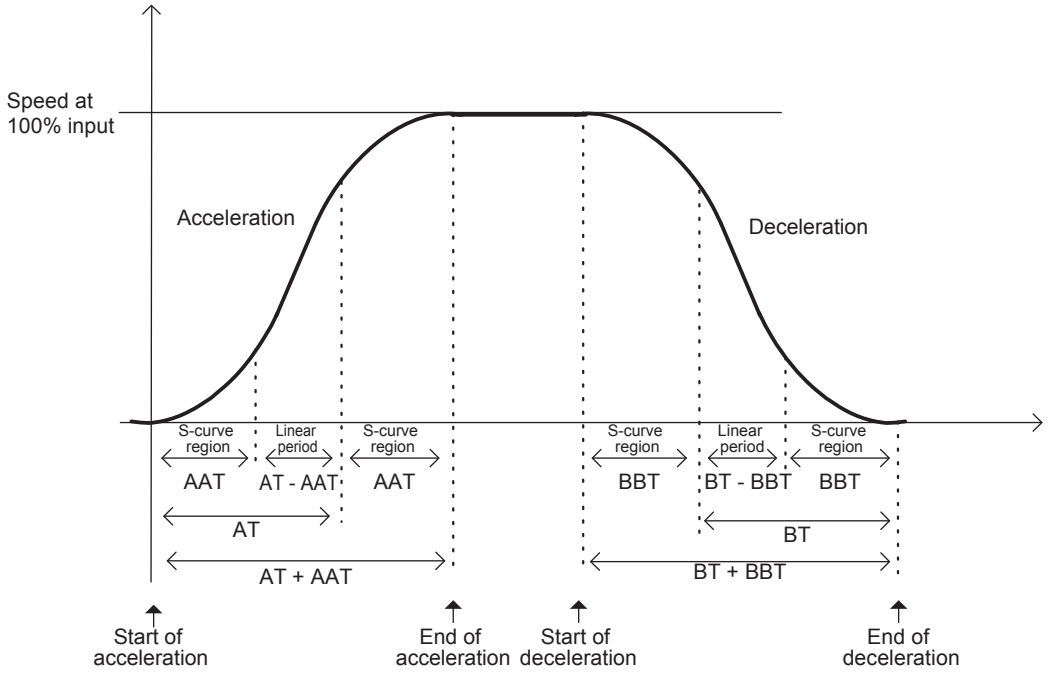
## [ c ] Parameter Table Configuration for SLAU Instruction with Real Numbers

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | – | (Reserved.) | Spare register | – |
| 2 | F | LV | 100% level of input | Scale for 100% input | IN |
| 4 | F | AT | Acceleration time | Time to accelerate from 0% to 100% (s) | IN |
| 6 | F | BT | Deceleration time | Time to decelerate from 100% to 0% (s) | IN |
| 8 | F | QT | Quick stop time | Time to make a quick stop from 100% to 0% (s) | IN |
| 10 | F | AAT | Acceleration S-curve time | Acceleration S-curve region time (s) | IN |
| 12 | F | BBT | Deceleration S-curve time | Deceleration S-curve region time (s) | IN |
| 14 | F | V | Current speed | SLAU output (output to Out) | OUT |
| 16 | F | DVDT1 | Current acceleration/ deceleration rate 1 | The actual acceleration or deceleration rate is output. | OUT |
| 18 | F | ABMD | Speed increase when holding | Amount of speed change until the speed stabilizes after the hold command is executed | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | RN | Line running | This input is closed to run the line. | IN |
| 1 | QS | Quick stop | This input is opened to execute a quick stop. | IN |
| 2 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | ARY | Accelerating | This output is closed during acceleration. | OUT |
| 9 | BRY | Decelerating | This output is closed during deceleration. | OUT |
| A | LSP | Zero speed | This output is closed during zero speed. | OUT |
| B | EQU | Equal | This output is closed when the input speed equals the output speed. | OUT |
| C to F | – | (Reserved.) | Spare output relays | OUT |

- If QS (quick stop) is opened, QT (quick stop time) is used as the acceleration/deceleration time.




Refer to *( 4 ) Additional Information* for details on the processing that is performed internally by the LAU instruction.

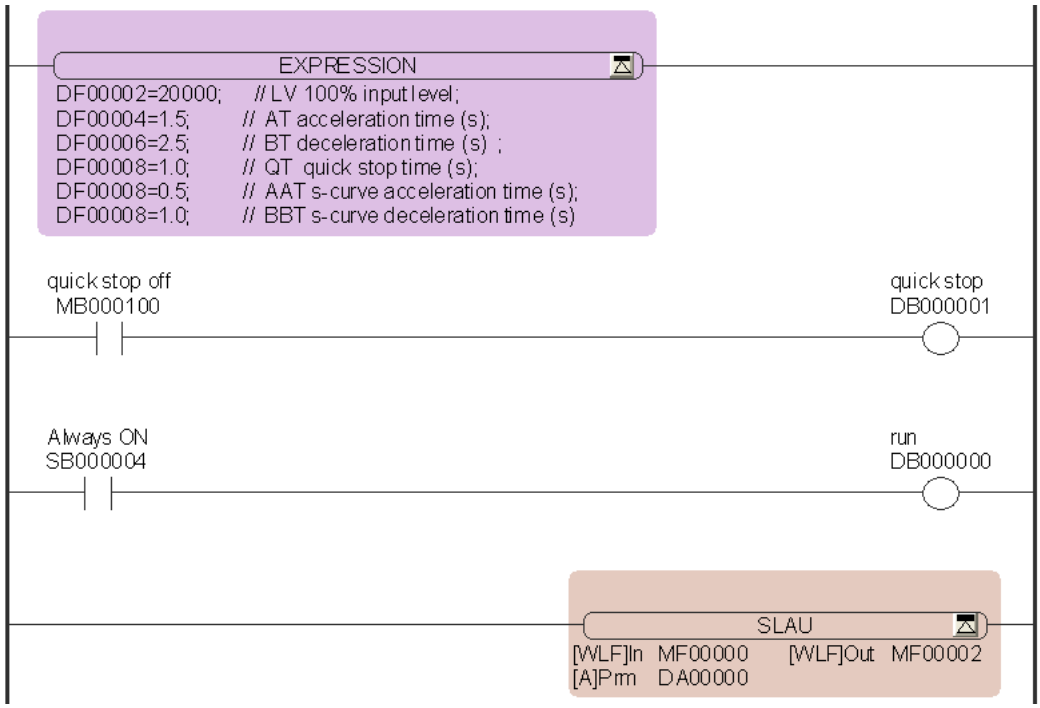
INFO

When QS (quick stop) opens (OFF), the output decelerates at the quick stop time and the output speed is set to 0.
It is not necessary to set the input speed to 0 in the same way as for the LAU instruction.
For a quick stop, the speed is decelerated linearly without applying the S-curve.
Set the parameters so that AT or BT (linear acceleration or deceleration time) is greater than or equal to AAT or BBT (S-curve acceleration or deceleration time).

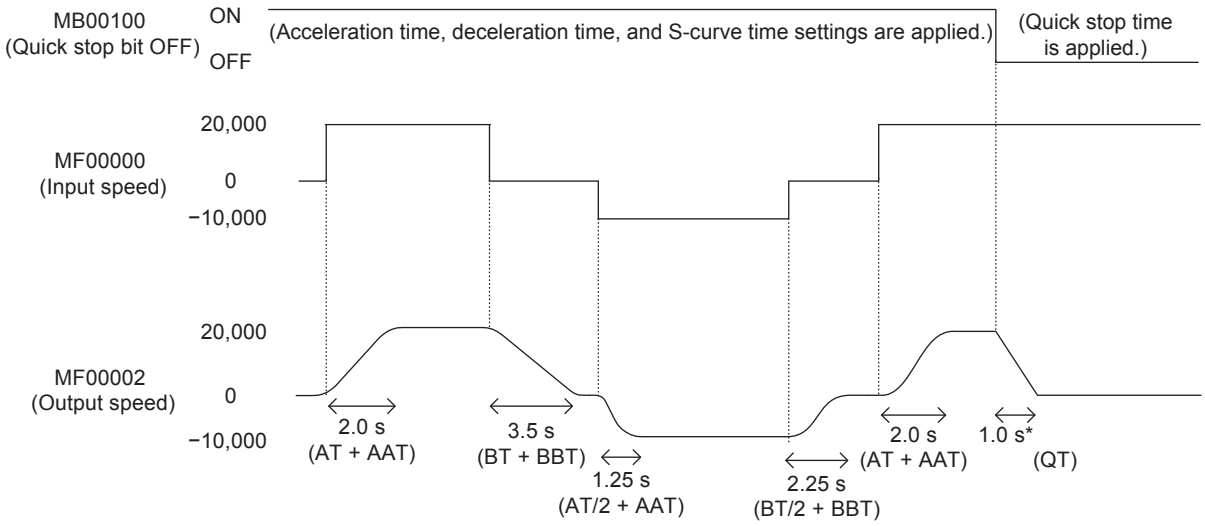
## ( 3 ) Programming Example

In the following programming example, the SLAU instruction for real numbers is executed with the specified acceleration and deceleration rates where MF00000 is the input speed and MF00002 is the output speed.
The following parameters are set with an EXPRESSION instruction to create the acceleration or deceleration rate.

- Speed when input level of acceleration or deceleration rate is 100% = 20,000
- Acceleration time = 1.5 s
- Deceleration time = 2.5 s
- Quick stop time = 0.5 s
- Acceleration S-curve time = 0.5 s
- Deceleration S-curve time = 1.0 s



The following figure shows how each register operates.




* If the quick stop bit is turned OFF, the speed is decelerated to a stop using the quick stop time, regardless of the S-curve time and input speed.

## ( 4 ) Additional Information

The following operations are performed internally by the SLAU instruction.

### [ a ] Operation of the SLAU Instruction for Integers

The SLAU instruction for integers calculates the speed output value during acceleration, deceleration, quick stops, S-curve acceleration, S-curve deceleration, and the current acceleration or deceleration rates using the formulas shown below based on predefined parameters.
In this formula, V is the speed output value, V' is the previous speed output value, VI is the input value for the speed reference, and Ts is the scan time set value.

#### ■ Speed Output Value during Acceleration

The speed output value during acceleration is calculated as follows:

If VI > V' (V' ≥ 0) outside an S-curve region (ADVS > ADV), then V = V' + ADV.
If VI < V' (V' ≤ 0) outside an S-curve region (ADVS > ADV), then V = V' − ADV.

$$\text{ADV (acceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM1}}{\text{AT (0.1 s)} \times 1{,}000}$$

#### ■ Speed Output Value during Deceleration

The speed output value during deceleration is calculated as follows:

If VI > V' (V'< 0) outside an S-curve region (BDVS > BDV), then V = V' + BDV.
If VI < V' (V'> 0) outside an S-curve region (BDVS > BDV), then V = V' − BDV.

$$\text{BDV (deceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM1}}{\text{BT (0.1 s)} \times 1{,}000}$$

## ■ Speed Output Value during a Quick Stop

The speed output value during a quick stop is calculated as follows:

If QS = OFF (VI > V', V' < 0), then V = V' + QDV.
If QS = OFF (VI < V', V' > 0), then V = V' − QDV.

$$\text{QDV (Quick Stop Rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)} + \text{REM1}}{\text{QT (0.1 s)} \times 1{,}000}$$

- For a quick stop, the speed is decelerated linearly without applying the S-curve.

## ■ Speed Output Value during S-Curve Acceleration

The speed output value during S-curve acceleration is calculated as follows:

If VI > V' (V' ≥ 0) inside an S-curve region (ADVS < ADV), then V = V' + ADVS.
If VI < V' (V' ≤ 0) inside an S-curve region (ADVS < ADV), then V = V' − ADVS.

ADVS (S-curve region acceleration rate) = ADVS' ± AADVS

$$\text{AADVS} = \frac{\text{ADV} \times \text{Ts (0.1 ms)} + \text{REM2}}{\text{AAT (0.01 s)} \times 100}$$

## ■ Speed Output Value during S-Curve Deceleration

The speed output value during S-curve deceleration is calculated as follows:

If VI > V' (V'< 0) inside an S-curve region (BDVS < BDV), then V = V' + BDVS.
If VI < V' (V'> 0) inside an S-curve region (BDVS < BDV), then V = V' − BDVS.

BDVS (S-curve region deceleration rate) = BDVS' ± BBDVS

$$\text{BBDVS} = \frac{\text{BDV} \times \text{Ts (0.1 ms)} + \text{REM2}}{\text{BBT (0.01 s)} \times 100}$$

## ■ Current Acceleration/Deceleration Rate

If DVDTF (skip execution of DVDT1 operation) is ON, DVDT1 (current acceleration/deceleration rate 1) will be calculated according to the setting of DVDTS (DVDT1 operation selection) using one of the following formulas. If DVDTF is OFF, DVDT1 is set to 0.

$$\text{If DVDTS is ON, DVDT1} = \frac{(\text{V} - \text{V}') \times 5{,}000}{\text{ADV}}$$

$$\text{If DVDTS is OFF, DVDT1} = (\text{V} - \text{V}') \times \text{DVDTK}$$

The value for DVDT2 (current acceleration/deceleration rate 2) is calculated as follows:

During acceleration: Inside the S-curve region: DVDT2 = ±ADVS
Outside the S-curve region: DVDT2 = ±ADV
During deceleration: Inside the S-curve region: DVDT2 = ±BDVS
Outside the S-curve region: DVDT2 = ±BDV
During a quick stop: DVDT = ±QDV

The result of ABMD (speed increase upon holding) is output after the following operation is performed.

$$\text{ABMD} = \frac{\text{DVDT2}' \times \text{DVDT2}'}{2 \times \text{AADVS (BBDVS)}}$$

DVDT2': Previous value of DVDT2 (current acceleration/deceleration rate 2)

INFO

ARY (accelerating) turns ON at the following times:
- When V' ≥ 0 and ADV > 0, or when V' ≤ 0 and ADV < 0
- If V' ≥ 0 and ADVS > 0 inside an S-curve region, or if V' ≤ 0 and ADVS < 0 inside an S-curve region

BRY (decelerating) turns ON at the following times:
- When V' < 0 and BDV > 0, or when V' > 0 and BDV < 0
- When V' < 0 and QDV > 0, or when V' > 0 and QDV < 0
- When V' < 0 and BDVS > 0 inside an S-curve region, or if V' > 0 and BDVS < 0 inside an S-curve region

LSP (zero speed) turns ON when V equals 0. EQU (equal) turns ON when VI equals V.
If RN (line running) is opened (OFF), the outputs for V, DVDT1, DVDT2, DVDT3, REM1, REM2, and REM3 are set to 0.

### [ b ] Operation of the SLAU Instruction for Double-length Integers or Real Numbers

The SLAU instruction for double-length integers or real numbers calculates the speed output value during acceleration, deceleration, quick stops, S-curve acceleration, S-curve deceleration, and the current acceleration or deceleration rates using the formulas shown below.
In this formula, V is the speed output value, V' is the previous speed output value, VI is the input value for the speed reference, Ts is the scan time set value, ADVS' is the previous ADVS value, and BDVS' is the previous BDVS value.

#### ■ Speed Output Value during Acceleration

The speed output value during acceleration is calculated as follows:

If VI > V' (V' ≥ 0) outside an S-curve region (ADVS > ADV), then V = V' + ADV.
If VI < V' (V' ≤ 0) outside an S-curve region (ADVS > ADV), then V = V' − ADV.

$$\text{ADV (acceleration rate)} = \frac{\text{LV} \times \text{Ts (0.1 ms)}}{\text{AT (s)} \times 10{,}000}$$

#### ■ Speed Output Value during Deceleration

The speed output value during deceleration is calculated as follows:

If VI > V' (V'< 0) outside an S-curve region (BDVS > BDV), then V = V' + BDV.
If VI < V' (V'> 0) outside an S-curve region (BDVS > BDV), then V = V' − BDV.

$$\text{BDV (deceleration rate)} = \frac{-\text{LV} \times \text{Ts (0.1 ms)}}{\text{BT(s)} \times 10{,}000}$$

#### ■ Speed Output Value during a Quick Stop

The speed output value during a quick stop is calculated as follows:

If QS = OFF (VI > V', V' < 0), then V = V' + QDV.
If QS = OFF (VI < V', V' > 0), then V = V' − QDV.

$$\text{QDV (Quick Stop Rate)} = \frac{-\text{LV} \times \text{Ts (0.1 ms)}}{\text{QT (s)} \times 10{,}000}$$

- For a quick stop, the speed is decelerated linearly without applying the S-curve.

## ■ Speed Output Value during S-Curve Acceleration

The speed output value during S-curve acceleration is calculated as follows:

If VI > V' (V' ≥ 0) inside an S-curve region (ADVS < ADV), then V = V' + ADVS.
If VI < V' (V' ≤ 0) inside an S-curve region (ADVS < ADV), then V = V' − ADVS.

ADVS (S-curve region acceleration rate) = ADVS' ± AADVS

$$\text{AADVS} = \frac{\text{ADV} \times \text{Ts (0.1 ms)}}{\text{AAT (s)} \times 10{,}000}$$

## ■ Speed Output Value during S-Curve Deceleration

The speed output value during S-curve deceleration is calculated as follows:

If VI > V' (V'< 0) inside an S-curve region (BDVS < BDV), then V = V' + BDVS.
If VI < V' (V'> 0) inside an S-curve region (BDVS < BDV), then V = V' − BDVS.

BDVS (S-curve region deceleration rate) = BDVS' ± BBDVS

$$\text{BBDVS} = \frac{\text{BDV} \times \text{Ts (0.1 ms)}}{\text{BBT (s)} \times 10{,}000}$$

## ■ Current Acceleration/Deceleration Rate

The value of DVDT (current acceleration/deceleration rate 1) is output after the following operation is performed:

During acceleration: Inside the S-curve region: DVDT = ADVS
Outside the S-curve region: DVDT = ADV
During deceleration: Inside the S-curve region: DVDT = BDVS
Outside the S-curve region: DVDT = BDV
During a quick stop: DVDT = QDV

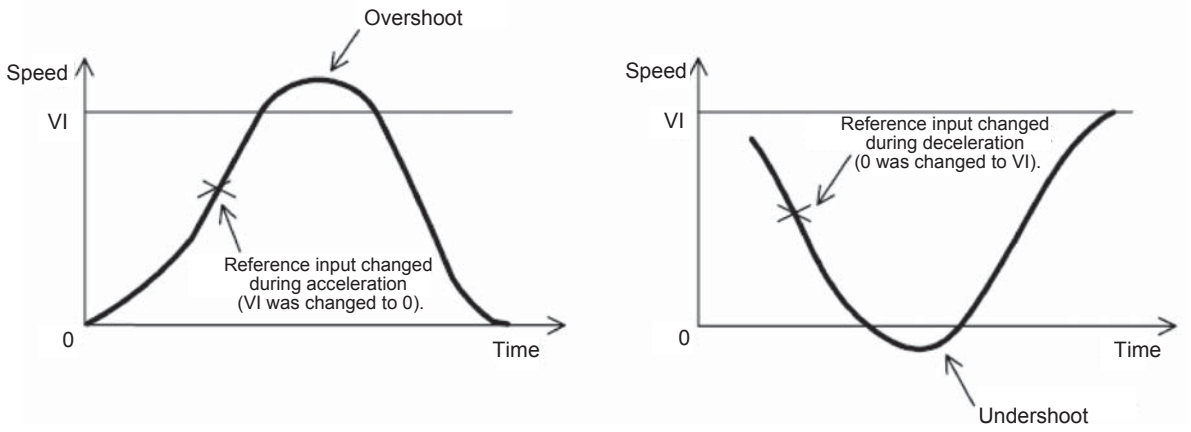
The result of ABMD (speed increase upon holding) is output after the following operation is performed.

$$\text{ABMD} = \frac{\text{DVDT} \times \text{DVDT}}{2 \times \text{AADVS (BBDVS)}}$$

LSP (zero speed) turns ON when V equals 0. EQU (equal) turns ON when VI equals V.
If RN (line running) is opened (OFF), the outputs for V, DVDT, and AVMD are set to 0.

### [ c ] Precautions in Using the SLAU Instruction for Integers

Do not change the input value before the input speed (VI) is reached (i.e., during acceleration or deceleration). Otherwise, overshooting or undershooting may occur as shown in the following figures.

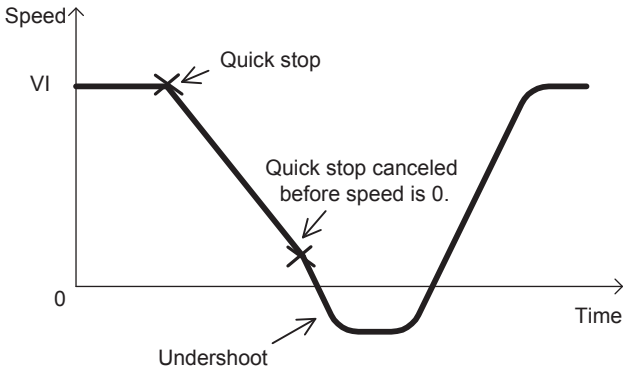



If VI (input value) must be changed while accelerating or decelerating, take one of the following measures in your application program.

- Use the SLAU instruction for real numbers.
- Use the SLAU instruction for integers together with the LIMIT instruction. Specifically, use the output value of the SLAU instruction for integers as the input value to the LIMIT instruction to prevent overshooting or undershooting.

### [ d ] Precaution When Canceling a Quick Stop While Decelerating during a Quick Stop

When decelerating for a quick stop, do not cancel the quick stop before the output speed reaches 0. Otherwise, undershooting may occur while approaching the input speed.




If you must reset the quick stop before the output speed reaches 0 and undershooting is a problem, take one of the following measures in your application program.

- Do not cancel the quick stop before the output speed reaches 0.
- Use the LIMIT instruction on the output speed to prevent undershooting when the quick stop is canceled.

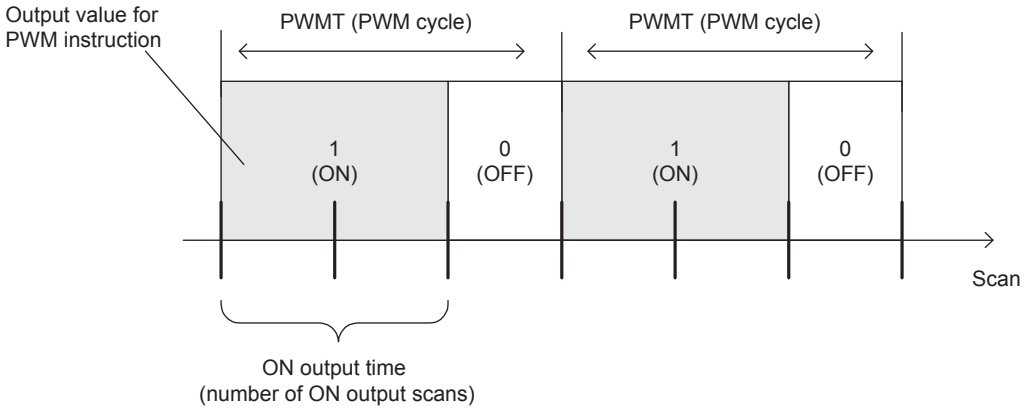
# 5.8.13 Pulse Width Modulation (PWM)

## ( 1 ) Operation

The PWM instruction converts the input value (from -100.00% to 100.00%) using pulse-width modulation and outputs the result to the output value and parameter table. The input value and output value must be integers. Double-length integers and real numbers cannot be used.

IMPORTANT

If using an integer, set an integral multiple of 1 ms for the scan time.



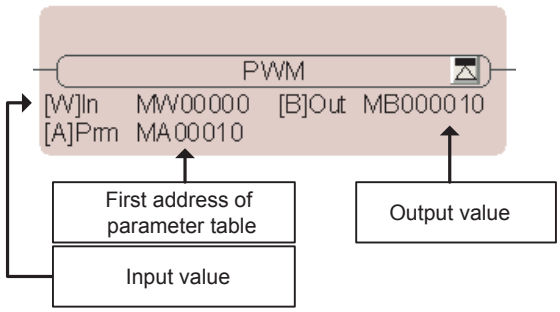

The ON output time and number of ON output scans of the PWM instruction can be calculated with the following formula.
X is the input value, PWMT is the PWM cycle (ms), and Ts is the scan time set value (ms).

$$\text{ON output time} = \frac{\text{PWMT}\ (X + 10{,}000)}{20{,}000}$$

$$\text{Number of ON output scans} = \frac{\text{PWMT}\ (X + 10{,}000)}{Ts \times 20{,}000}$$

- The relation between the input value and the PWM output ON ratio is as follows:
  - Input value 100.00% → 100% ON (ON output time = PWMT)
  - Input value 0.00% → 50% ON (ON output time = PWMT/2)
  - Input value -100.00% → 0% ON (ON output time = 0)
- After turning ON the power supply, close PWMRST (PWM reset) to clear all internal calculations before using the PWM instruction. When the PWM reset bit is closed, all internal calculations are reset and then the PMW operation starts execution from that point.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Input value (In) | × | O | × | × | × | O | O |
| First address of parameter table (Prm) | × | × | × | × | O* | O | O |
| Output value (Out) | O* | × | × | × | × | O | × |

* C and # registers cannot be used.

### [ a ] Ranges of Input and Output Values

The input value must be between -10,000 and 10,000 in units of 0.01%.
If the input exceeds this range, processing is performed for the upper limit (10,000) and the lower limit (-10,000).
The output value is set to 1 when the PWM output is ON, or to 0 when the PWM output is OFF.

### [ b ] Parameter Table Configuration

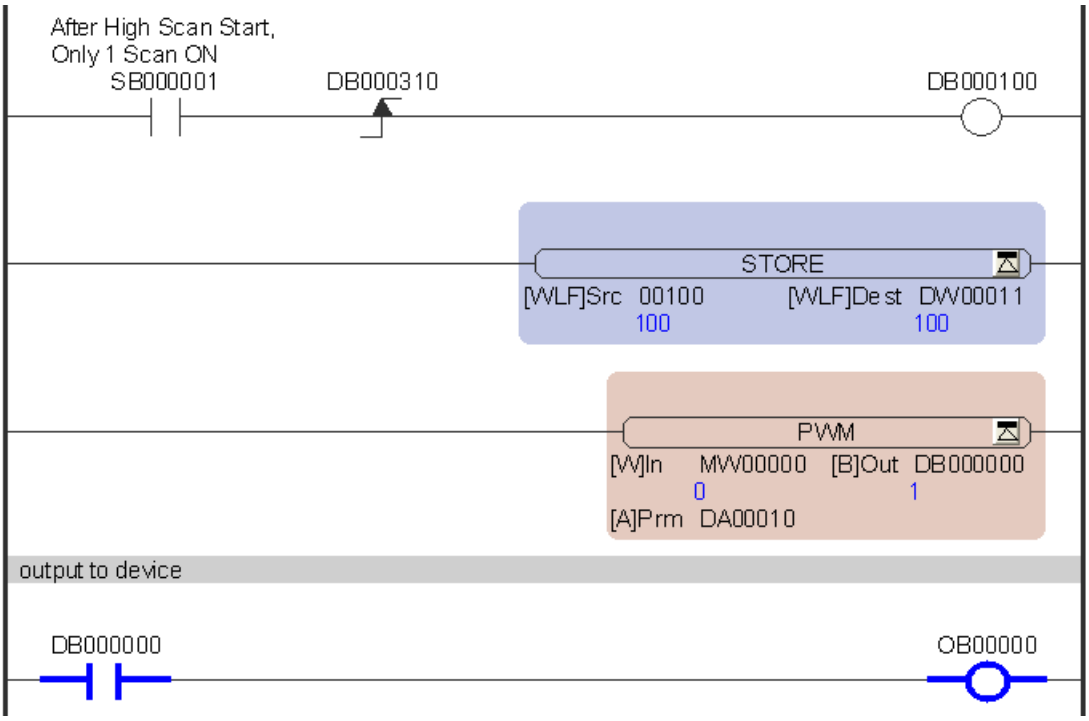
| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | W | RLY | Relay I/O | Relay inputs and relay outputs* | IN/OUT |
| 1 | W | RWMT | PWM cycle | PWM cycle (1 ms)<br>Range: 1 to 32,767 ms | IN |
| 2 | W | ONCNT | ON output setting timer | ON output setting timer (1 ms) | OUT |
| 3 | W | CVON | ON output counting timer | ON output counting timer (1 ms) | OUT |
| 4 | W | CVONREM | ON output counting timer remainder | ON output counting timer remainder (0.1 ms) | OUT |
| 5 | W | OFFCNT | OFF output setting timer | OFF output setting timer (1 ms) | OUT |
| 6 | W | CVOFF | OFF output counting timer | OFF output counting timer (1 ms) | OUT |
| 7 | W | CVOFFREM | OFF output counting timer remainder | OFF output counting timer remainder (0.1 ms) | OUT |

* The relay input and output bits are assigned as given below. (Close = Bit change to 1 (ON), Open = Bit change to 0 (OFF))

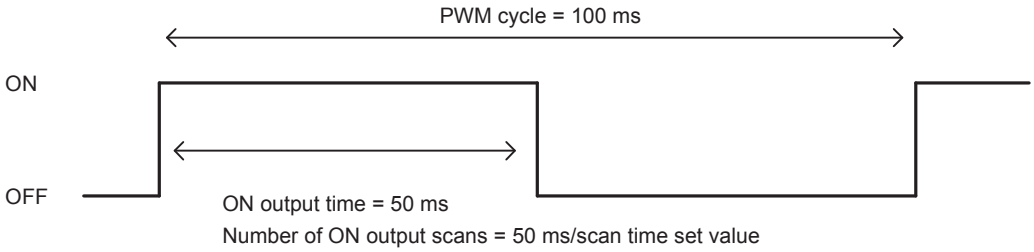
| Bit | Symbol | Name | Specification | I/O |
|---|---|---|---|---|
| 0 | PWMRST | PWM reset bit | This input is closed to reset the PWM operation. | IN |
| 2 to 7 | – | (Reserved.) | Spare input relays | IN |
| 8 | PWMOUT | PWM output | PWM output<br>(The output value is set to 1 when the output is ON, or to 0 when the output is OFF.) | OUT |
| 9 to F | – | (Reserved.) | Spare output relays | OUT |

## ( 3 ) Programming Example

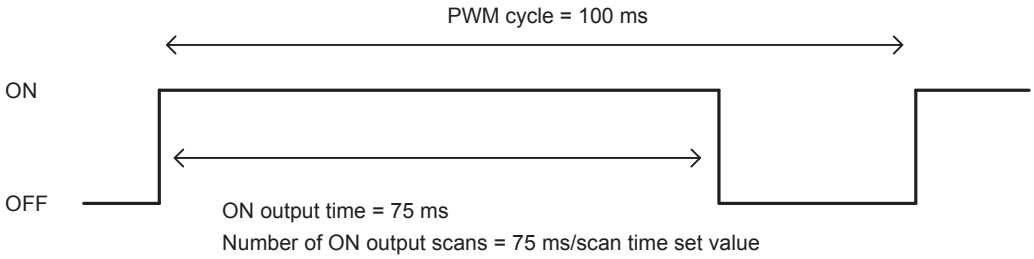
In the following programming example, the PWM output for the input value in MW00000 is stored in OB000000 where the PWM cycle is 100 ms.




This figure shows the output of OB000000 when MW00000 is 0 (0%: ON output time is 1/2 of the PWM cycle).




This figure shows the output of OB000000 when MW00000 is 7,500 (75%: ON output time is 3/4 of the PWM cycle).

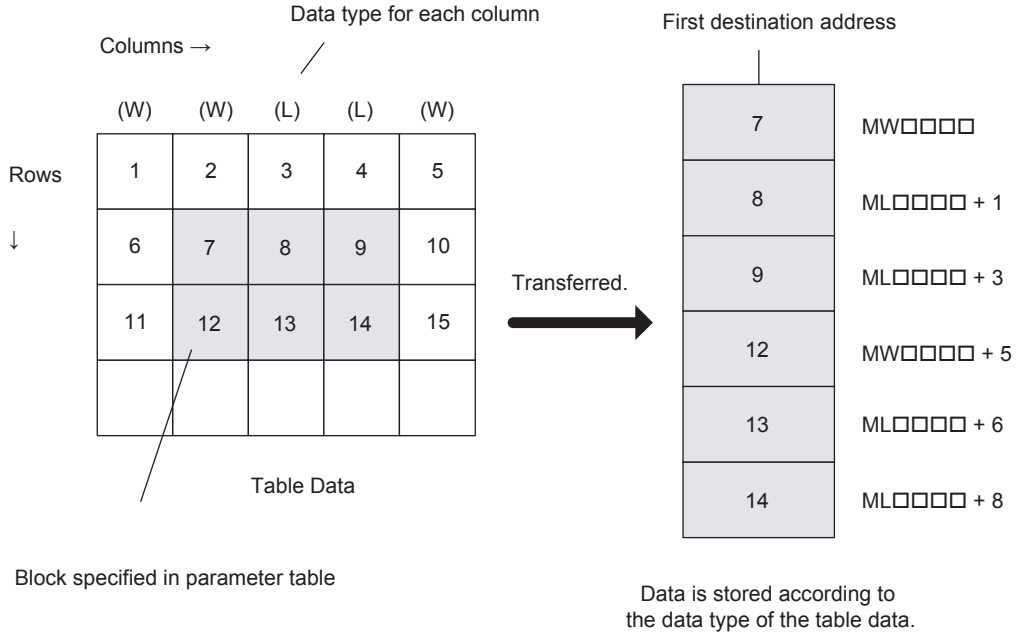


# 5.9 Table Manipulation Instructions

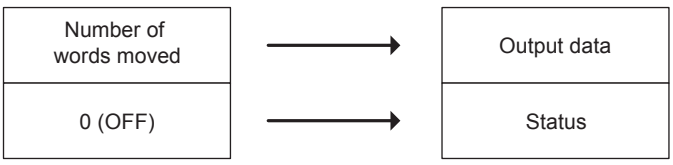
## 5.9.1 Read Table Block (TBLBR)

### ( 1 ) Operation

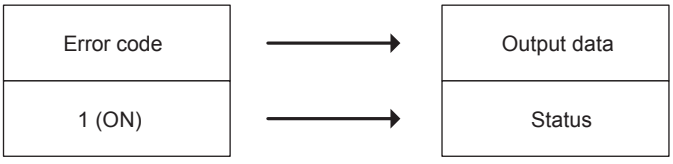
The TBLBR instruction moves the block of the table data that is specified by the table name, row number, and column number to a continuous area that starts at the first destination address. The data is stored in the destination area according to the data type of the elements that were read.
If an error occurs when accessing the table, such as data that is outside of the valid range or not enough data length at the destination, an error is output and no data is read. The contents in the destination area will remain unchanged.
If the instruction ends normally, the number of words that were moved is output, and the Status bit is turned OFF. If an error occurs, an error code is output and the Status bit is turned ON.




#### [ a ] If the Move Succeeds

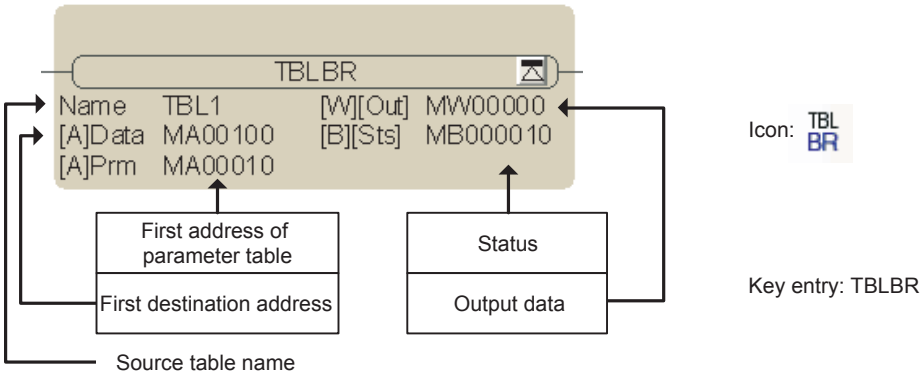



#### [ b ] If the Move Fails




- If the move fails, the destination area will retain the contents from before the instruction was executed.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First destination address (Data) | × | × | × | × | ○*2 | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW1 | First row number of table elements | First row number of table elements to move (1 to 65,535) | IN |
| 2 | L | COL1 | First column number of table elements | First column number of table elements to move (1 to 32,767) | IN |
| 4 | W | RLEN | Number of row elements | Number of row elements (1 to 32,767) | IN |
| 5 | W | CLEN | Number of column elements | Number of column elements (1 to 32,767) | IN |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

- The error codes apply to all table manipulation instructions.

## ( 3 ) Programming Example

In the following programming example, the specified block in record table data TBL1 is moved to an area that starts at MW00100 when switch 1 (DB000000) turns ON.

The parameter table is set as shown in the following table.

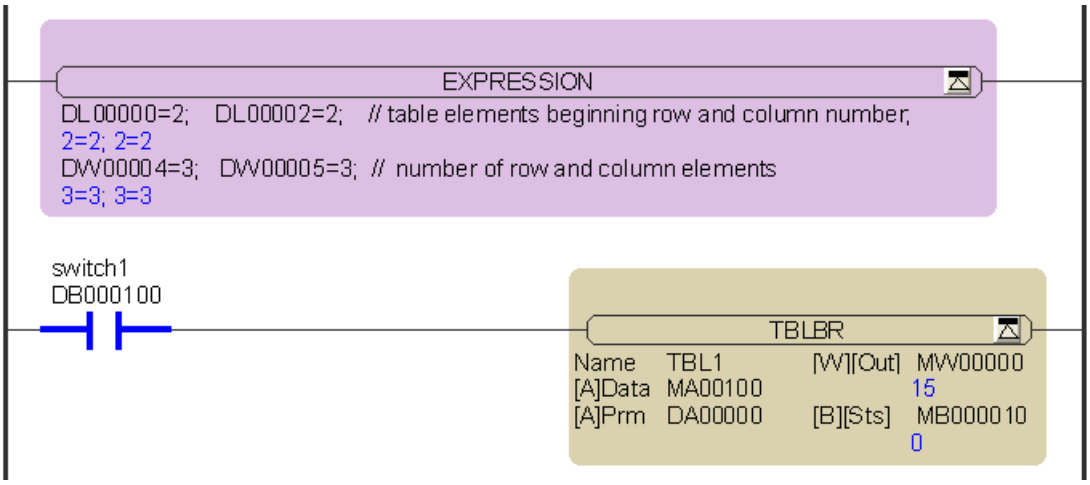
| Register | Data | Remarks |
|---|---|---|
| DL00000 | 2 | First row number of table elements |
| DL00002 | 2 | First column number of table elements |
| DW00004 | 3 | Number of row elements |
| DW00005 | 3 | Number of column elements |

The contents of table data TBL1 are given below.

| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( L ) | 4 ( L ) | 5 ( F ) |
|---|---|---|---|---|---|
| 1 | 1000 | 1001 | 10000 | 10001 | 1.1 |
| 2 | 2000 | 2002 | 20000 | 20002 | 1.2 |
| 3 | 3000 | 3003 | 30000 | 30003 | 1.3 |
| 4 | 4000 | 4004 | 40000 | 40004 | 1.4 |
| 5 | 5000 | 5005 | 50000 | 50005 | 1.5 |

Block to move

- The column data types are given in parentheses.




After the instruction is executed, the data is moved to an area that starts from MW00100 as shown below.
The number of words that was moved is set to 15 in MW00000 (output data), and MB000010 (status) is set to 0 (move successful).

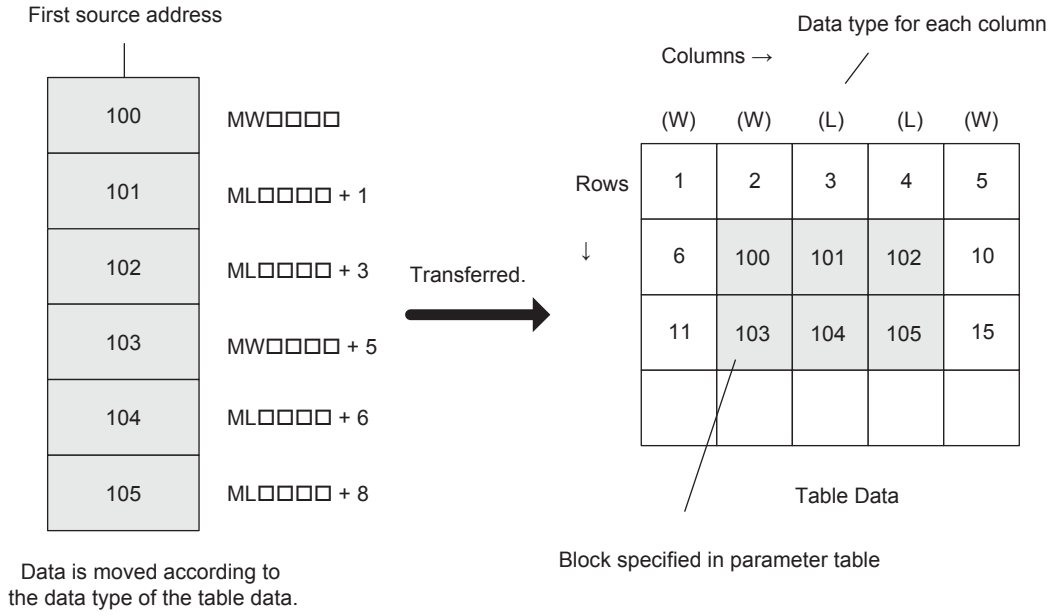
| Register | Data | Register | Data | Register | Data |
|---|---|---|---|---|---|
| MW00100 | 2002 | ML00101 | 20000 | ML00103 | 20002 |
| MW00105 | 3003 | ML00106 | 30000 | ML00108 | 30003 |
| MW00110 | 4004 | ML00111 | 40000 | ML00113 | 40004 |

- The registers are assigned as shown in the above table.

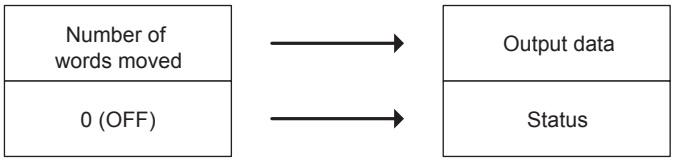
## 5.9.2 Write Table Block (TBLBW)

### ( 1 ) Operation

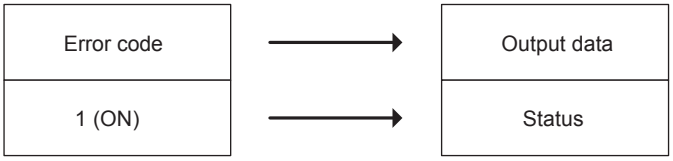
The TBLBW instruction moves the data from a continuous area that starts at the first source address to a block of the table data that is specified by the table name, row number, and column number. The data is stored under the assumption that the data type of the source area and each element in the table data are the same.
If an error occurs when accessing the table, such as data that is outside of the valid range or not enough data length at the source, an error is output and no data is written. The contents in the destination area will remain unchanged.
If the instruction ends normally, the number of words that were moved is output, and the Status bit is turned OFF. If an error occurs, an error code is output and the Status bit is turned ON.




#### [ a ] If the Move Succeeds

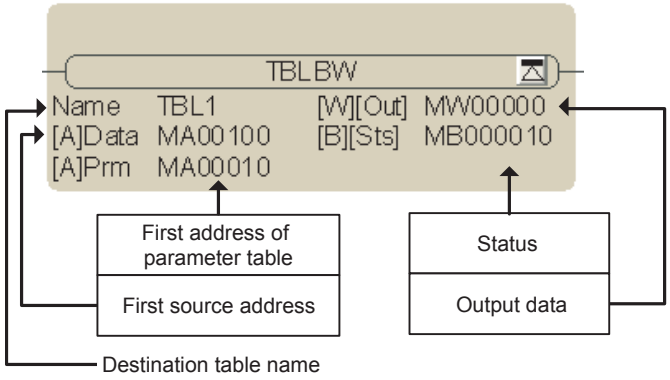



#### [ b ] If the Move Fails




- If the move fails, the destination area will retain the contents from before the instruction was executed.

## ( 2 ) Format




Icon: TBL BR

Key entry: TBLBW

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First source address (Data) | × | × | × | × | ○*2 | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW1 | First row number of table elements | First row number of table elements to move (1 to 65,535) | IN |
| 2 | L | COL1 | First column number of table elements | First column number of table elements to move (1 to 32,767) | IN |
| 4 | W | RLEN | Number of row elements | Number of row elements (1 to 32,767) | IN |
| 5 | W | CLEN | Number of column elements | Number of column elements (1 to 32,767) | IN |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

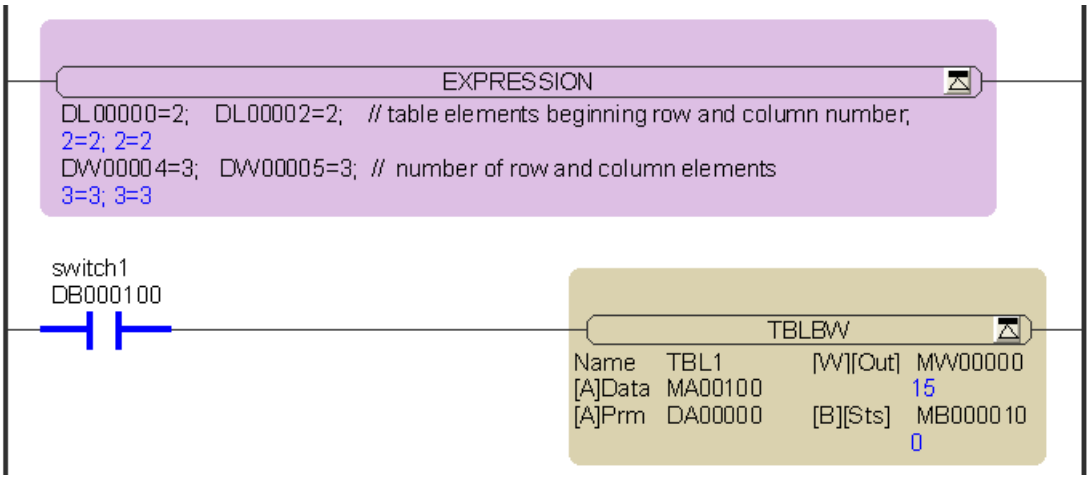
- The error codes apply to all table manipulation instructions.

## ( 3 ) Programming Example

In the following programming example, an area of data that starts at MW00100 is moved to a specified block in record table data TBL1 when switch 1 (DB00000) turns ON.

The parameter table is set as shown in the following table.

| Register | Data | Remarks |
|---|---|---|
| DL00000 | 2 | First row number of table elements |
| DL00002 | 2 | First column number of table elements |
| DW00004 | 3 | Number of row elements |
| DW00005 | 3 | Number of column elements |




The data that is written is given below.

| Register | Data | Register | Data | Register | Data |
|---|---|---|---|---|---|
| MW00100 | 1 | ML00101 | 2 | ML00103 | 3 |
| MW00105 | 4 | ML00106 | 5 | ML00108 | 6 |
| MW00110 | 7 | ML00111 | 8 | ML00113 | 9 |

The following table shows the contents of table data TBL1 after the instruction is executed.
The number of words that were moved is set to 15 in MW00000, and MB000010 (status) is set to 0 (move successful).

| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( L ) | 4 ( L ) | 5 ( F ) |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | 1 | 2 | 3 | |
| 3 | | 4 | 5 | 6 | |
| 4 | | 7 | 8 | 9 | |
| 5 | | | | | |

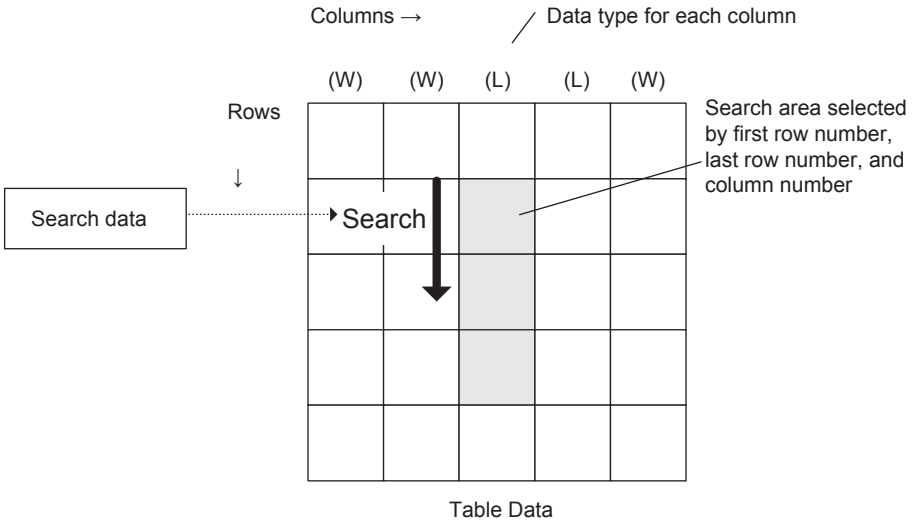
Area that is written

- The column data types are given in parentheses.

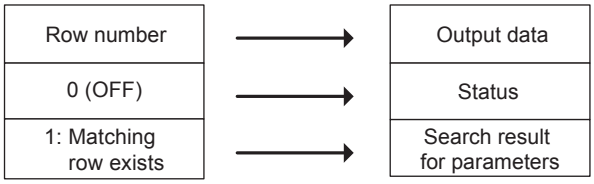
## 5.9.3 Search for Table Row (TBLSRL)

### ( 1 ) Operation

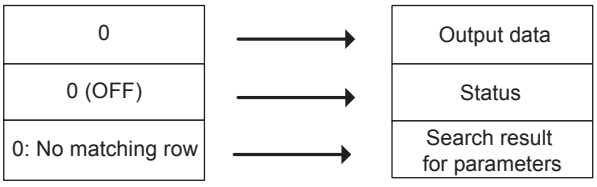
The TBLSRL instruction searches for the search data in column elements of the table data that is specified by the table name, row numbers, and column number. The search result is output as the row number of the data that matches the search data. The type of the data to be searched is automatically determined by the data type of the specified column elements.
If the instruction ends normally and the search data is found, the search result in the input parameter table is set to 1, the output data is set to the row number, and the status is turned OFF. If the search data is not found, the search result and output data are set to 0. If an error occurs, an error code is set in the output data and the status is turned ON.



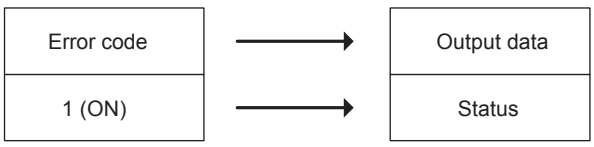

#### [ a ] Search Data Found



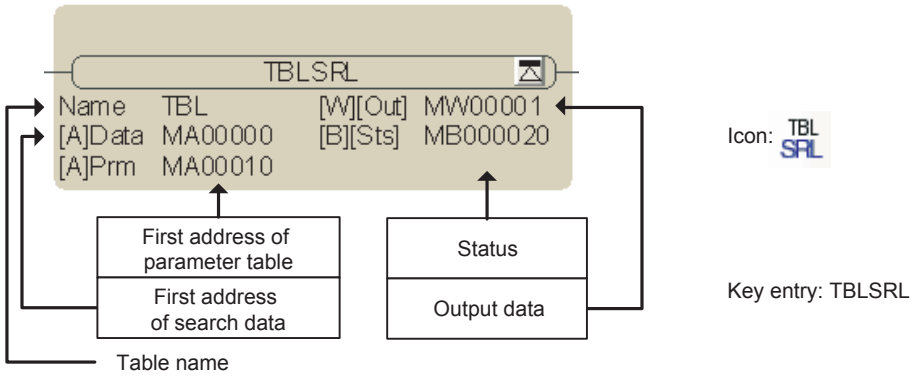

#### [ b ] Search Data Not Found




#### [ c ] Search Error



## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of search data (Data) | × | × | × | × | ○ | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW1 | First row number of table elements | First row number of table elements to search (1 to 65,535) | IN |
| 2 | L | ROW2 | Last row number of table elements | Last row number of table elements to search (1 to 65,535) | IN |
| 4 | L | COLUMN | Column number of table elements | Column number of table elements to search (1 to 32,767) | IN |
| 6 | W | FIND | Search result | Search result<br>0: No matching row<br>1: Matching row exists | OUT |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

## ( 3 ) Programming Example

In the following programming example, a search is made for search data 32 in MW00000 in part of a column of array table data TBL1.

The parameter table is set as shown in the following table.

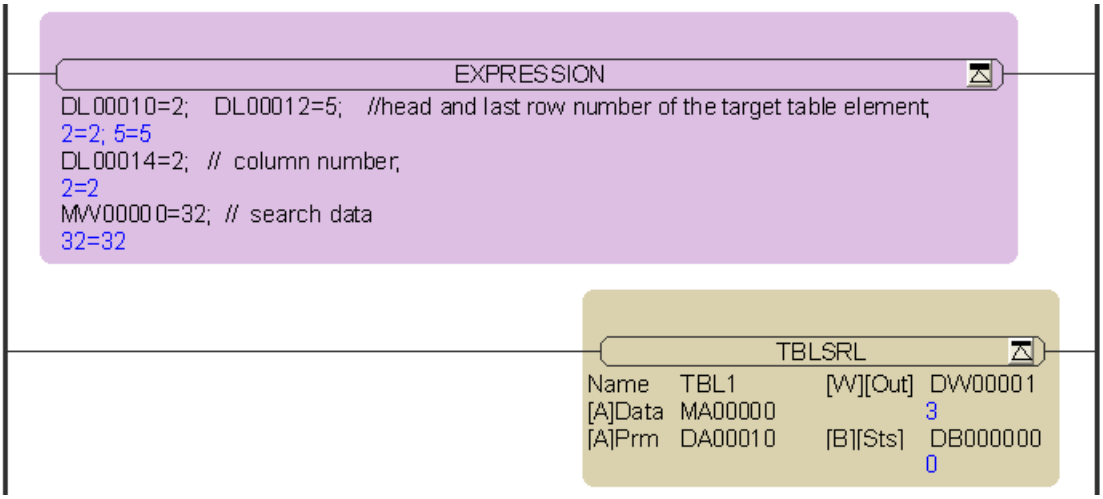
| Register | Data | Remarks |
|---|---|---|
| DL00010 | 2 | First row number of table elements |
| DL00012 | 5 | Last row number of table elements |
| DL00014 | 2 | Column number of table elements |

The contents of table data TBL1 are given below. (Table elements are integer data.)

| Column / Row | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 11 | 12 | 13 | 14 | 15 |
| 2 | 21 | 22 | 23 | 24 | 25 |
| 3 | 31 | 32 | 33 | 34 | 35 |
| 4 | 41 | 42 | 43 | 44 | 45 |
| 5 | 51 | 52 | 53 | 54 | 55 |

Area to search

A match for 32 (MW00000) was found in row number 3 in the search area, so DW00001 is set to 3.

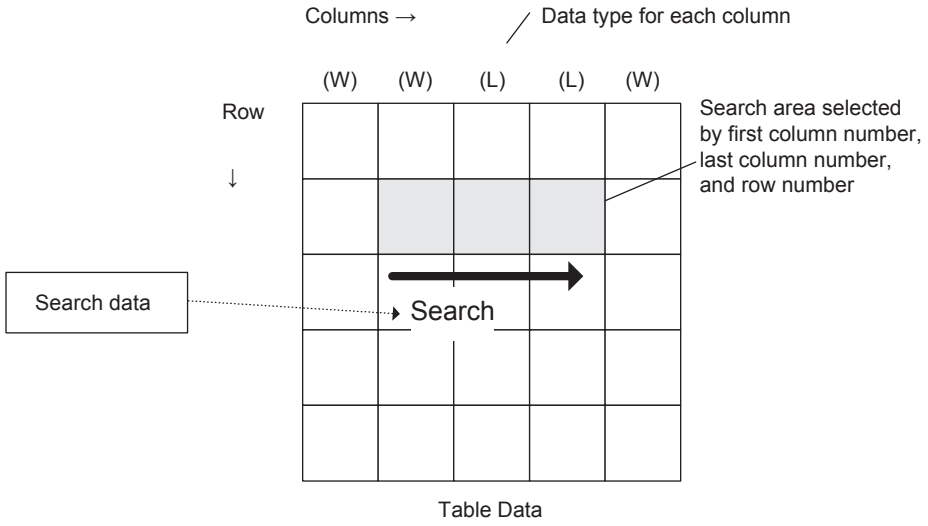


# 5.9.4 Search for Table Column (TBLSRC)

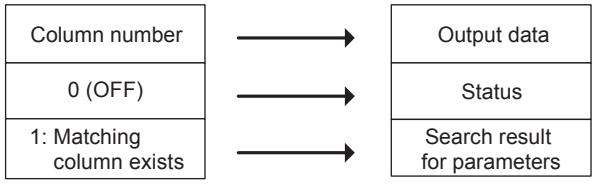
## ( 1 ) Operation

The TBLSRC instruction searches for the search data in row elements of the table data that is specified by the table name, column numbers, and row number. The search result is output as the column number of the data that matches the search data. The type of the data to be searched is automatically determined by the data types of the specified row elements.

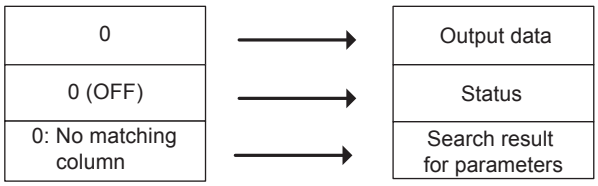
If the instruction ends normally and the search data is found, the search result in the input parameter table is set to 1, the output data is set to the column number, and the status is turned OFF. If the search data is not found, the search result and output data are set to 0. If an error occurs, an error code is set in the output data and the status is turned ON.



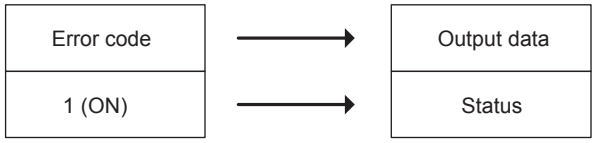

### [ a ] Search Data Found



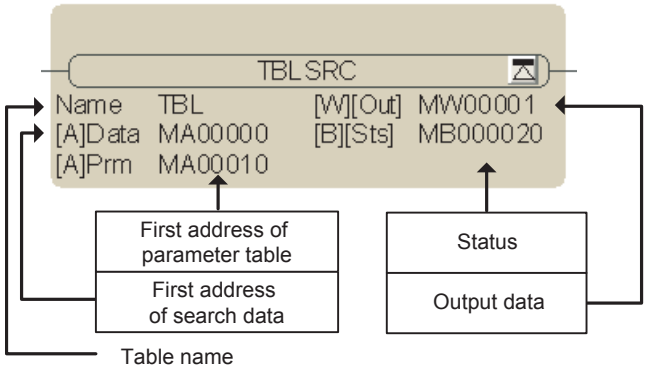

### [ b ] Search Data Not Found




### [ c ] Search Error



## ( 2 ) Format




Icon: TBL SRC

Key entry: TBLSRC

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of search data (Data) | × | × | × | × | ○ | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW1 | Row number of table elements | Row number of table elements to search (1 to 65,535) | IN |
| 2 | L | COLUMN1 | First column number of table elements | First column number of table elements to search (1 to 32,767) | IN |
| 4 | L | COLUMN2 | Last column number of table elements | Last column number of table elements to search (1 to 32,767) | IN |
| 6 | W | FIND | Search result | Search result<br>0: No matching column<br>1: Matching column exists | OUT |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Unexpected element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

## ( 3 ) Programming Example

In the following programming example, a search is made for search data 34 in MW00000 in part of a row of array table data TBL1.

The parameter table is set as shown in the following table.

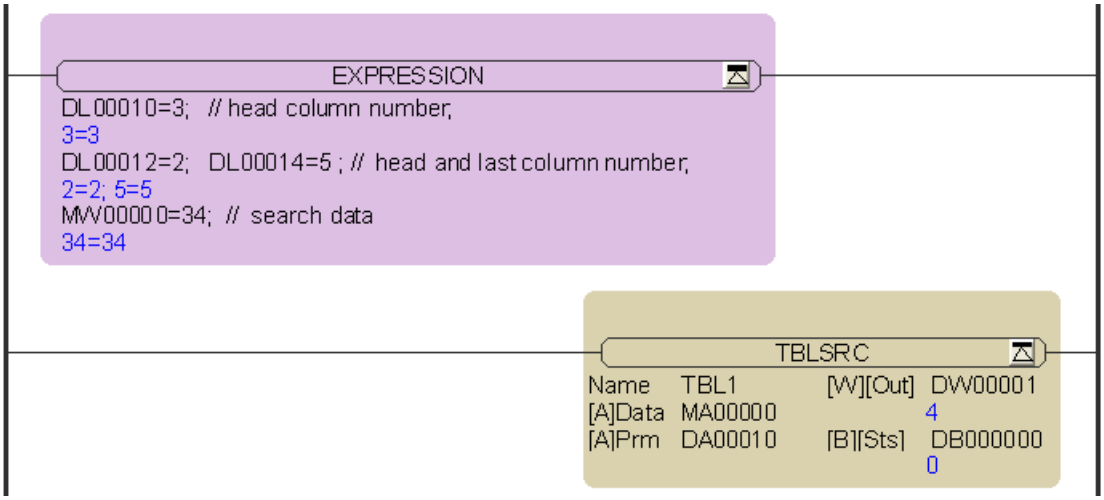
| Register | Data | Remarks |
|---|---|---|
| DL00010 | 3 | Row number of table elements |
| DL00012 | 2 | First column number of table elements |
| DL00014 | 5 | Last column number of table elements |

The contents of table data TBL1 are given below. (Table elements are integer data.)

| Column / Row | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 11 | 12 | 13 | 14 | 15 |
| 2 | 21 | 22 | 23 | 24 | 25 |
| 3 | 31 | 32 | 33 | 34 | 35 |
| 4 | 41 | 42 | 43 | 44 | 45 |
| 5 | 51 | 52 | 53 | 54 | 55 |

Area to search

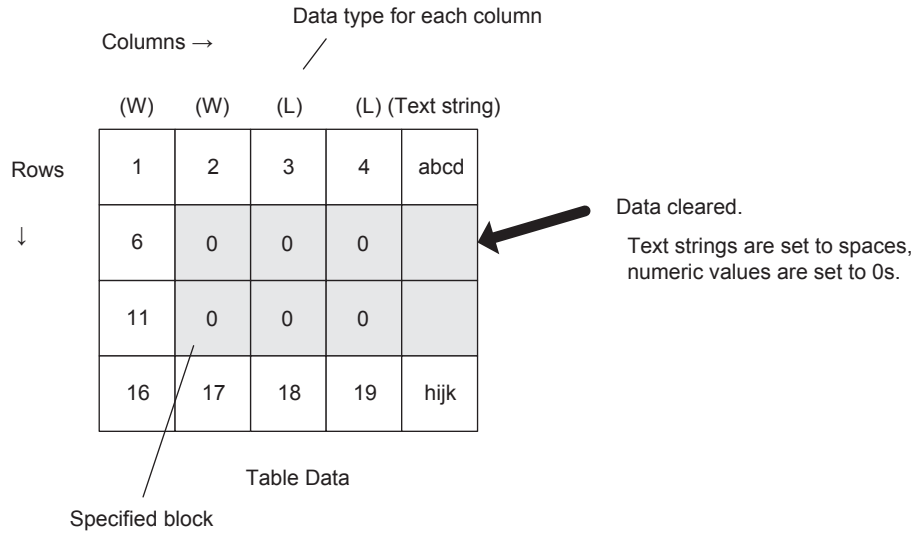
A match for 34 (MW00000) was found in column number 4 in the search area, so DW00001 is set to 4.



# 5.9.5 Clear Table Block (TBLCL)

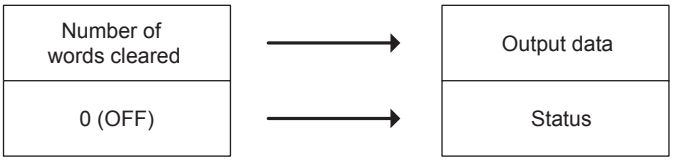
## ( 1 ) Operation

The TBLCL instruction clears the block of data in the table data that is specified by the table name, row numbers, and column numbers. The elements are filled with spaces if the data type is for text strings, and 0s if the data type is for numeric values.
If both the first row number and the first column number of the table element are 0, the entire table will be cleared.
If an error occurs when accessing the table, such as data that is outside of the valid range or not enough data length at the destination, an error is output and no data is written.
If the instruction ends normally, the number of words that were cleared is output and the status is turned OFF. If an error occurs, an error code is set in the output data and the status is turned ON.

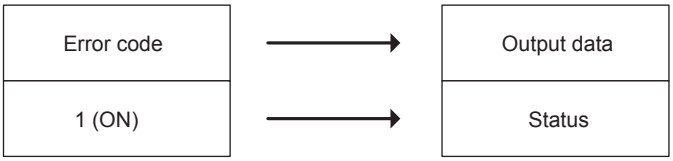



- If the first row number and column number of the table element are both 0, the entire table is cleared.

### [ a ] If the Clear Succeeds

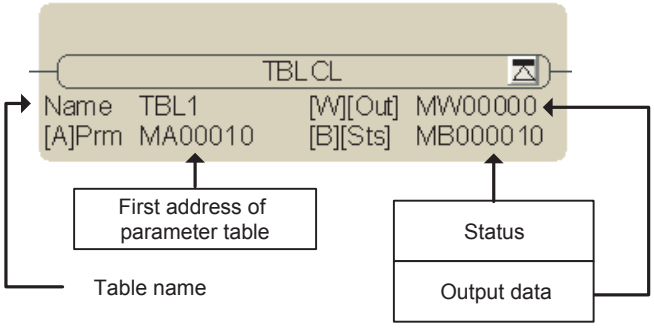



### [ b ] If the Clear Fails




- If the clear fails, the table data will retain the contents from before the instruction was executed.

## ( 2 ) Format




Icon: TBL CL

Key entry: TBLCL

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

* 1. Optional.
* 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW | First row number of table elements | First row number of table elements to clear (1 to 65,535) | IN |
| 2 | L | COL | First column number of table elements | First column number of table elements to clear (1 to 32,767) | IN |
| 4 | W | RLEN | Number of row elements | Number of row elements (1 to 32,767) | IN |
| 5 | W | CLEN | Number of column elements | Number of column elements (1 to 32,767) | IN |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

## ( 3 ) Programming Example

In the following programming example, the specified block is cleared from record table data TBL1 when switch 1 (DB000100) turns ON.
The parameter table is set as shown in the following table.

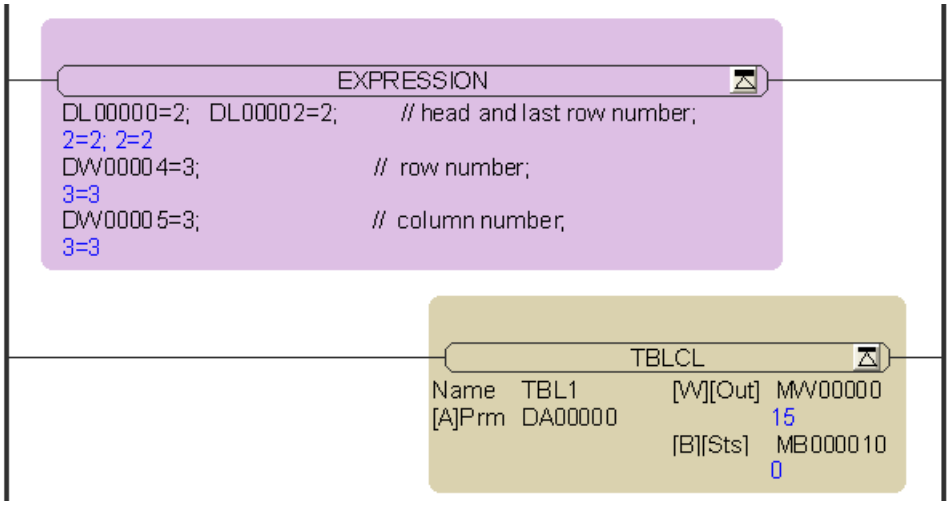
| Register | Data | Remarks |
|---|---|---|
| DL00000 | 2 | First row number of table elements |
| DL00002 | 2 | First column number of table elements |
| DW00004 | 3 | Number of row elements |
| DW00005 | 3 | Number of column elements |

The contents of table data TBL1 are given below.

| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( L ) | 4 (Text string) | 5 ( F ) |
|---|---|---|---|---|---|
| 1 | 1000 | 1001 | 10000 | ABCD | 1.1 |
| 2 | 2000 | 2002 | 20000 | BCDE | 1.2 |
| 3 | 3000 | 3003 | 30000 | CDEF | 1.3 |
| 4 | 4000 | 4004 | 40000 | DEFG | 1.4 |
| 5 | 5000 | 5005 | 50000 | EFGH | 1.5 |

Area to clear

- The column data types are given in parentheses.




The data is cleared after the instruction is executed as shown below.

| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( L ) | 4 (Text string) | 5 ( F ) |
|---|---|---|---|---|---|
| 1 | 1000 | 1001 | 10000 | ABCD | 1.1 |
| 2 | 2000 | 0 | 0 | | 1.2 |
| 3 | 3000 | 0 | 0 | | 1.3 |
| 4 | 4000 | 0 | 0 | | 1.4 |
| 5 | 5000 | 5005 | 50000 | EFGH | 1.5 |

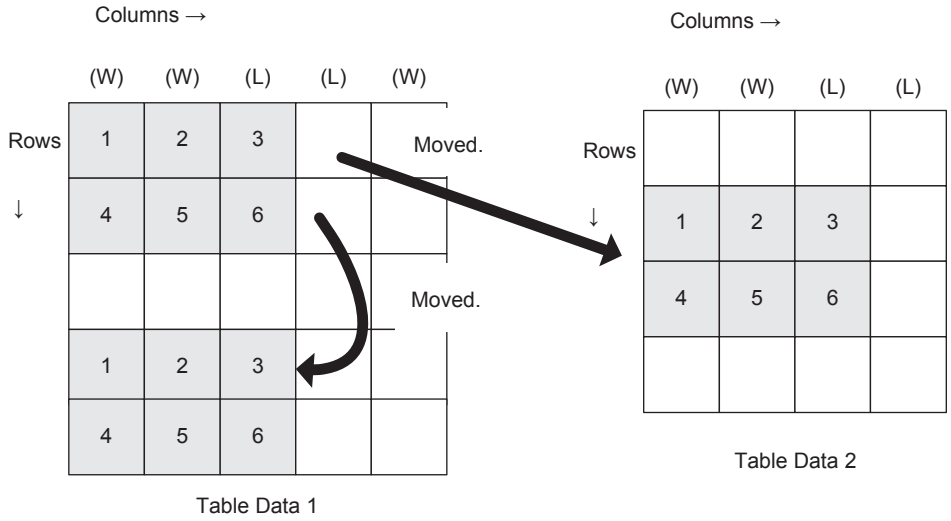
Area that was cleared

- The column data types are given in parentheses.

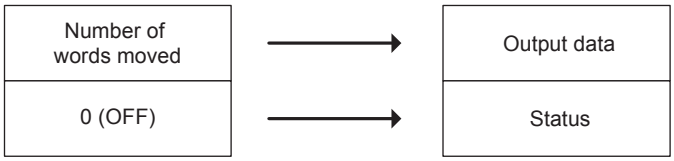
# 5.9.6 Move Table Block (TBLMV)

## ( 1 ) Operation

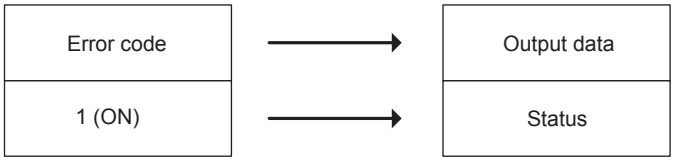
The TBLMV instruction moves a block of data in the table data that is specified by the table name, row number, and column number to a different table block. The block can be moved between different tables or within the same table. If the data type of the column elements in the source and destination do not match, an error is output and no data is moved.
If the instruction ends normally, the number of words that were moved is output, and the Status bit is turned OFF. If an error occurs, an error code is output and the Status bit is turned ON.




### [ a ] If the Move Succeeds

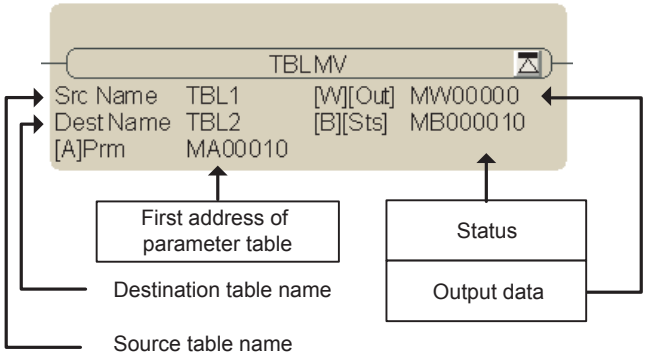



### [ b ] If the Move Fails




- If the move fails, the table data will retain the contents from before the instruction was executed.

## ( 2 ) Format




Icon: TBL MV

Key entry: TBLMV

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of parameter table (Prm) | × | × | × | × | ○ | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW1 | First row number of table elements | First row number of table elements at source to move (1 to 65,535) | IN |
| 2 | L | COLUMN1 | First column number of table elements | First column number of table elements at source to move (1 to 32,767) | IN |
| 4 | W | RLEN | Number of row elements | Number of row elements (1 to 32,767) | IN |
| 5 | W | CLEN | Number of column elements | Number of column elements (1 to 32,767) | IN |
| 6 | L | ROW2 | First row number of table elements | First row number of table elements at destination (1 to 65,535) | IN |
| 8 | L | COLUMN2 | First column number of table elements | First column number of table elements at destination (1 to 32,767) | IN |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

## ( 3 ) Programming Example

In the following programming example, the specified block in record table data TBL1 is moved to the specified block in table data TBL2 when switch 1 (DB000100) turns ON.

The contents of table data TBL1 are given below.

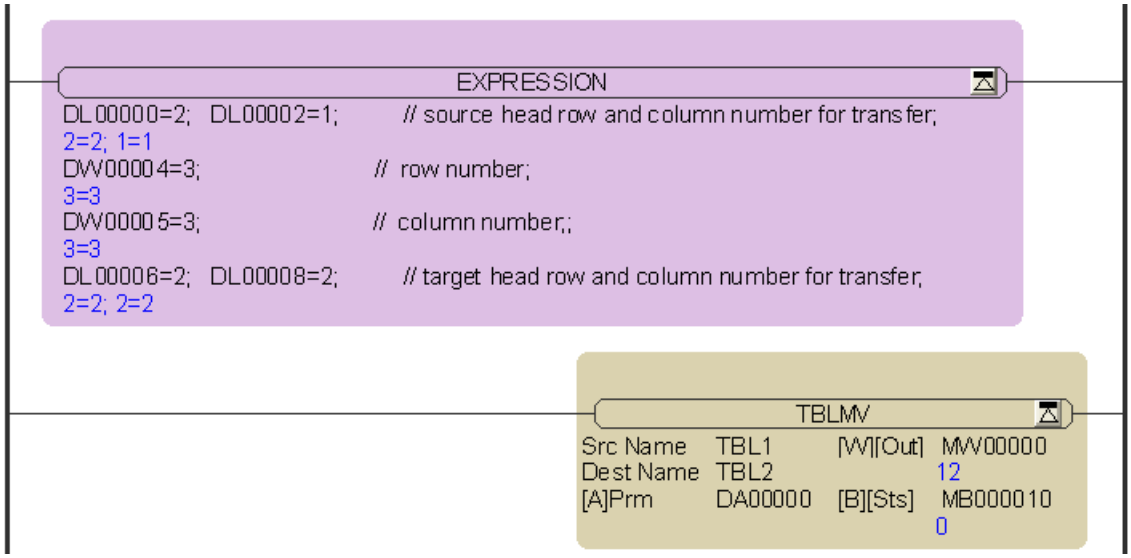
| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( L ) |
|---|---|---|---|
| 1 | 1000 | 1001 | 10000 |
| 2 | 2000 | 2002 | 20000 |
| 3 | 3000 | 3003 | 30000 |
| 4 | 4000 | 4004 | 40000 |
| 5 | 5000 | 5005 | 50000 |

Block to move

- The column data types are given in parentheses.

The parameter table is set as shown in the following table.

| Register | Data | Remarks |
|---|---|---|
| DL00000 | 2 | First row number at source |
| DL00002 | 1 | First column number at source |
| DW00004 | 3 | Number of row elements |
| DW00005 | 3 | Number of column elements |
| DL00006 | 2 | First row number at destination |
| DL00008 | 2 | First column number at destination |




This table shows the contents of table data TBL2 after the instruction is executed.

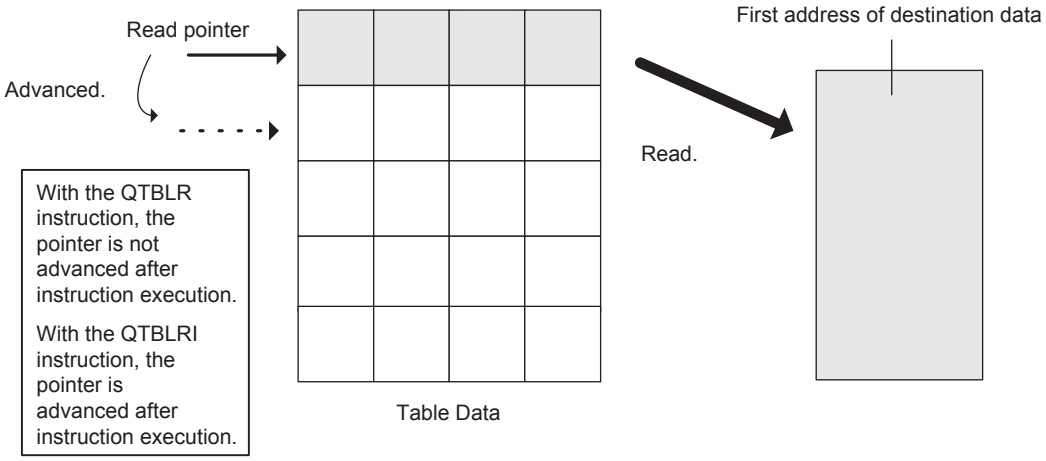
| Column / Row | 1 ( W ) | 2 ( W ) | 3 ( W ) | 4 ( L ) | 5 ( L ) |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | 2000 | 2002 | 20000 | |
| 3 | | 3000 | 3003 | 30000 | |
| 4 | | 4000 | 4004 | 40000 | |
| 5 | | | | | |

Block that was moved

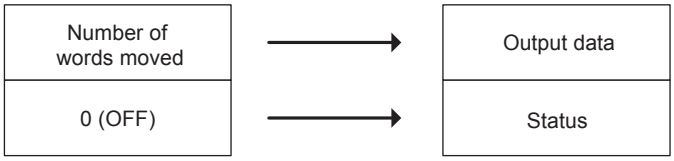
# 5.9.7 Read Queue Table (QTBLR and QTBLRI)

## ( 1 ) Operation

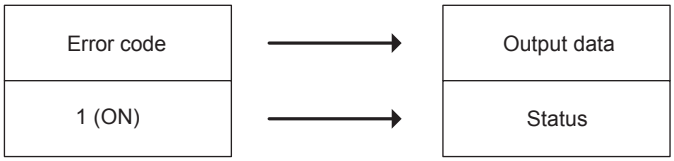
Column elements of the table data that are specified by the table name, row number, and column number are continuously read and stored in a continuous area that starts at a specified register. The data types of the elements that are read are automatically determined by the table that is specified. The data types of the destination registers are ignored and the data is stored according to the data types in the table without any conversion.
The QTBLR instruction does not change the queue table read pointer. The QTBLRI instruction advances the queue table read pointer by one row.
If an error occurs when accessing the table, such as a table name error, an out of range row number, or an empty queue buffer, an error is output, no data is read, and the pointer is not advanced.
The contents of the destination registers will be retained.
If the instruction ends normally, the number of words that were moved is output, and the Status bit is turned OFF. If an error occurs, an error code is output and the Status bit is turned ON.




### [ a ] If the Read Succeeds

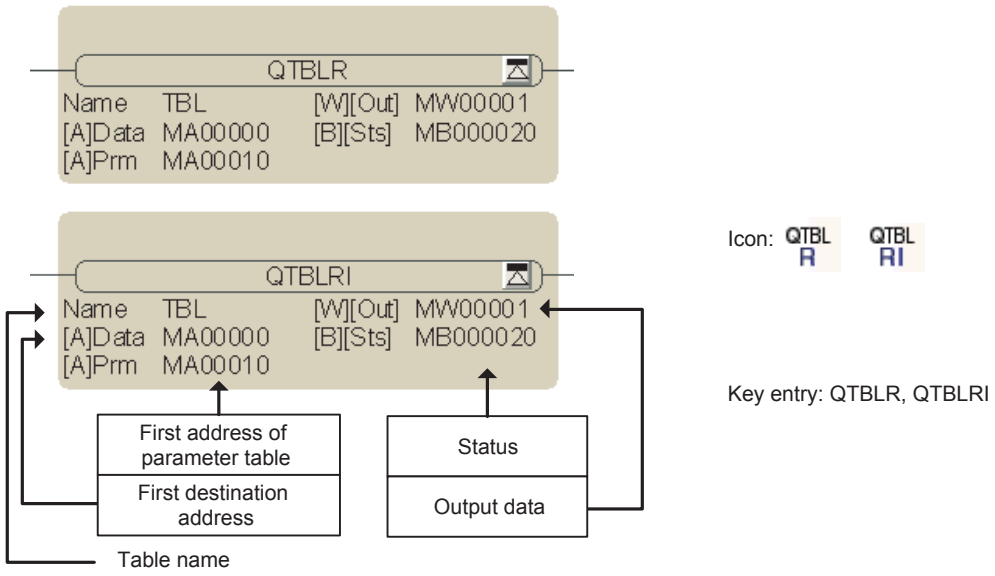



### [ b ] If the Read Fails




- If the read fails, the data at the destination will retain the contents from before the instruction was executed.

## ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First destination address (Data) | × | × | × | × | ○ | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○*2 | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW | Relative row number of table elements | Relative row number of table elements at source to move (1 to 65,535) | IN |
| 2 | L | COLUMN | First column number of table elements | First column number of table elements at source to move (1 to 32,767) | IN |
| 4 | W | CLEN | Number of column elements | Number of column elements to move (1 to 32,767) | IN |
| 5 | W | Reserved. | | | |
| 6 | L | RPTR | Read pointer | Read pointer of the queue after execution | OUT |
| 8 | L | WPTR | Write pointer | Write pointer of the queue after execution | OUT |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

### [ c ] Setting the Relative Row Numbers of Table Elements

| Relative Row Number | Row That Is Read | Remarks |
|---|---|---|
| 0 | Read pointer row | The pointer is advanced only for the QTBLRI instruction. |
| 1 | Read pointer row | Pointer is not advanced. |
| 2 | Read pointer row – 1 | Pointer is not advanced. |
| 3 | Read pointer row – 2 | Pointer is not advanced. |
| : | : | |
| n | Read pointer row – (n – 1) | Pointer is not advanced. |

## ( 3 ) Programming Example

In the following programming example, the specified column elements in array table data TBL1 are read and stored in MW00010 to MW00012 when switch 2 (DB000002) turns ON.
Before switch 2 is turned ON, the table data is set as shown below by turning ON switch 1 three times while changing the contents of MW00000 to MW00002. (Refer to information on the Write Queue Table instruction.)

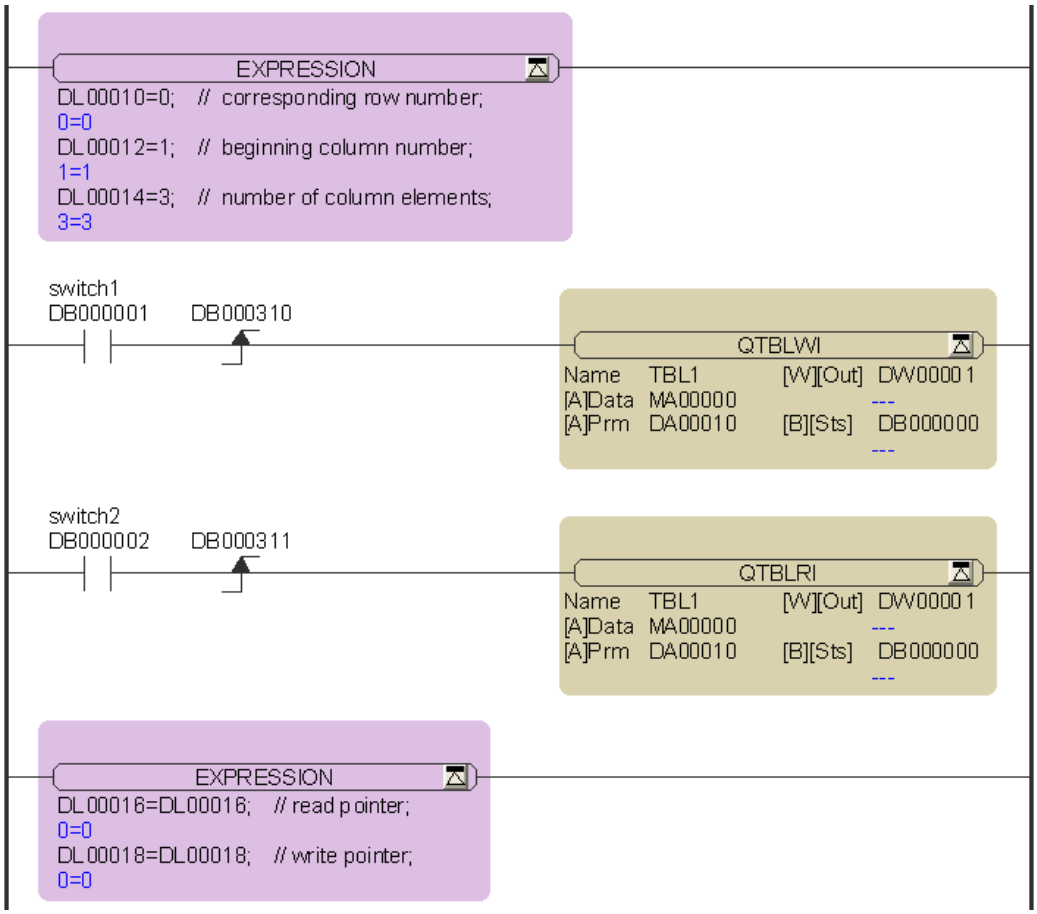
The contents of table data TBL1 are given below.

| Row \ Column | 1 (W) | 2 (W) | 3 (W) |
|---|---|---|---|
| 1 | 11 | 12 | 13 |
| 2 | 21 | 22 | 23 |
| 3 | 31 | 32 | 33 |

- The column data types are given in parentheses.

The parameter table is set as shown in the following table.

| Register | Data | Remarks |
|---|---|---|
| DL00010 | 0 | Relative row number |
| DL00012 | 1 | First column number |
| DW00014 | 3 | Number of row elements |




Here, switch 2 (DB000002) is turned ON three times. The data that is read changes each time from the first time to the third time, as shown below.

| Register | 1st Data | 2nd Data | 3rd Data |
|---|---|---|---|
| MW00010 | 11 | 21 | 31 |
| MW00011 | 12 | 22 | 32 |
| MW00012 | 13 | 23 | 33 |

The read pointer is advanced each time the instruction is executed starting at the first row on the first pass, the second row on the second pass, and so on, therefore resulting in the table shown above.

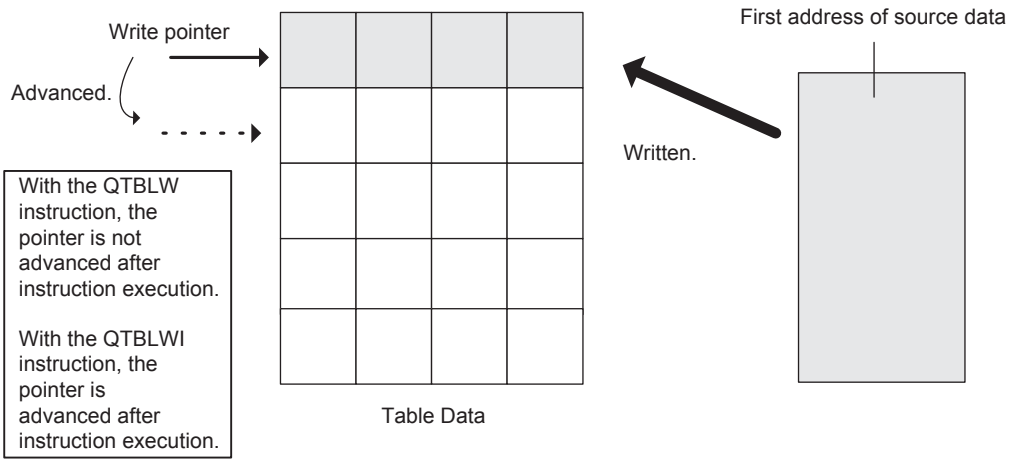
INFO

When the power supply is turned ON, values of the read pointer and write pointer are undefined. Always execute the QTBLCL instruction before using the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction.
An operation error may occur if the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction is executed without first executing the QTBLCL instruction.

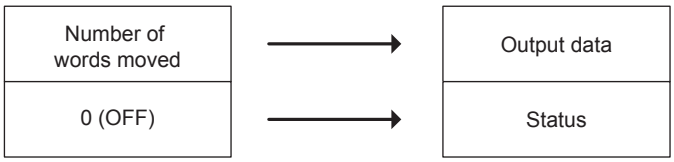
## 5.9.8 Write Queue Table (QTBLW and QTBLWI)

### ( 1 ) Operation

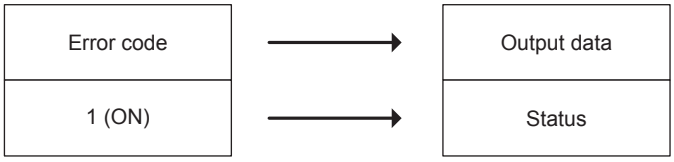
Data in a continuous area that starts at a specified register is continuously written to columns in a specified table. The instruction is processed under the assumption that the data type of the source and destination are the same.
The QTBLW instruction does not change the queue table write pointer. The QTBLWI instruction advances the queue table write pointer by one row.
If an error occurs when accessing the table, such as a table name error, an out of range row number, or a full queue buffer, an error is output, no data is written, and the pointer is not advanced.
The contents of the destination registers will be retained.
If the instruction ends normally, the number of words that were moved is output, and the Status bit is turned OFF. If an error occurs, an error code is output and the Status bit is turned ON.




#### [ a ] If the Write Succeeds

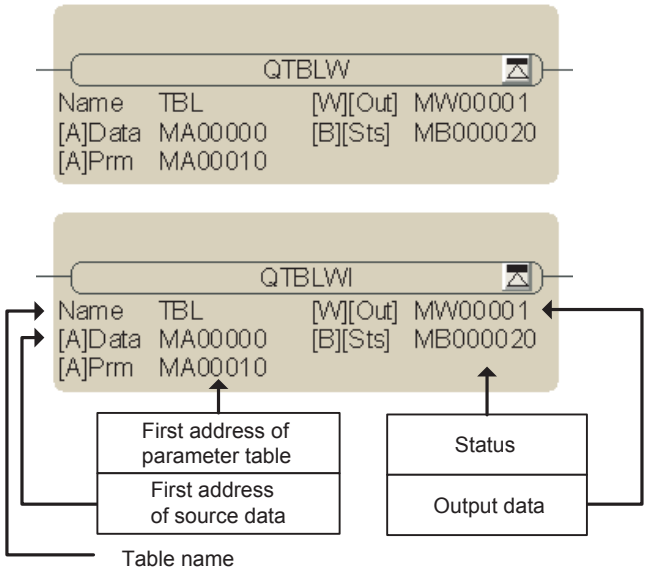



#### [ b ] If the Write Fails




- If the write fails, the table data will retain the contents from before the instruction was executed.

## ( 2 ) Format




Icon: QTBL W QTBL WI

Key entry: QTBLW, QTBLWI

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| First address of source data (Data) | × | × | × | × | ○ | × | × |
| First address of parameter table (Prm) | × | × | × | × | ○*2 | × | × |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts)*1 | ○*2 | × | × | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### [ a ] Parameter Table Configuration

| Address | Data Type | Symbol | Name | Specification | I/O |
|---|---|---|---|---|---|
| 0 | L | ROW | Relative row number of table elements | Relative row number of table elements at destination (1 to 65,535) | IN |
| 2 | L | COLUMN | First column number of table elements | First column number of table elements at destination (1 to 32,767) | IN |
| 4 | W | CLEN | Number of column elements | Number of column elements to move (1 to 32,767) | IN |
| 5 | W | Reserved. | | | |
| 6 | L | RPTR | Read pointer | Read pointer of the queue after execution | OUT |
| 8 | L | WPTR | Write pointer | Write pointer of the queue after execution | OUT |

### [ b ] Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

### [ c ] Setting the Relative Row Number of Table Elements

| Relative Row Number | Row That Is Read | Remarks |
|---|---|---|
| 0 | Write pointer row | The pointer is advanced only for the QTBLWI instruction. |
| 1 | Write pointer row | Pointer is not advanced. |
| 2 | Write pointer row – 1 | Pointer is not advanced. |
| 3 | Write pointer row – 2 | Pointer is not advanced. |
| : | : | |
| n | Write pointer row – (n – 1) | Pointer is not advanced. |

## ( 3 ) Programming Example

In the following programming example, the data from MW00000 to MW00002 is written to the specified column elements in array table data TBL1 when switch 1 (DB000001) turns ON.

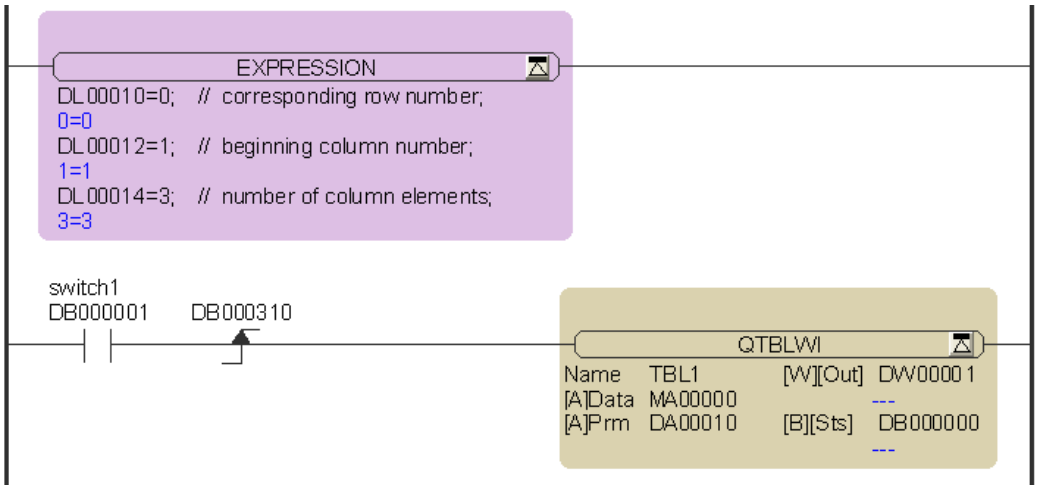
Initialize table data TBL1 before executing this type of programming.

| Row \ Column | 1 (W) | 2 (W) | 3 (W) |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |

- The column data types are given in parentheses.

The parameter table is set as shown in the following table.

| Register | Data | Remarks |
|---|---|---|
| DL00010 | 0 | Relative row number |
| DL00012 | 1 | First column number |
| DW00014 | 3 | Number of row elements |

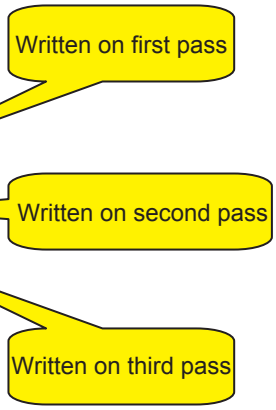



After changing the contents of MW00000 to MW00002 as shown in the following table, turn ON the switch 1 (DB000001) three times.

| Register | 1st Data | 2nd Data | 3rd Data |
|---|---|---|---|
| MW00000 | 11 | 21 | 31 |
| MW00001 | 12 | 22 | 32 |
| MW00002 | 13 | 23 | 33 |

The write pointer is advanced each time the instruction is executed starting at the first row on the first pass, the second row on the second pass, and so on. After three executions, TBL1 will be set with data as shown below.

| Column / Row | 1 (W) | 2 (W) | 3 (W) |
|---|---|---|---|
| 1 | 11 | 12 | 13 |
| 2 | 21 | 22 | 23 |
| 3 | 31 | 32 | 33 |



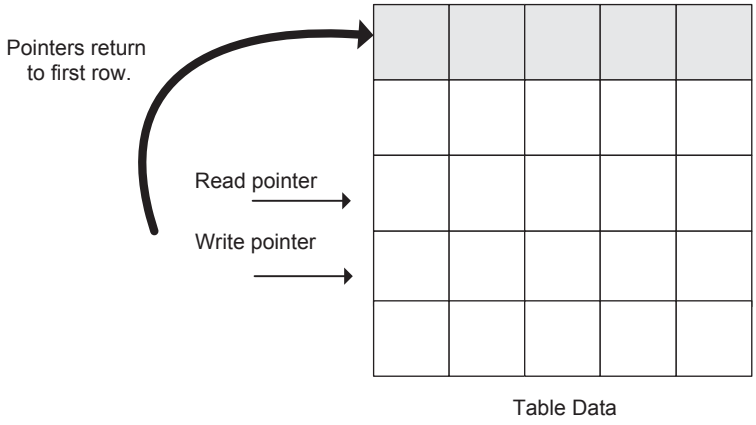

INFO

When the power is turned ON, values of the read pointer and write pointer are undefined. Always execute the QTBLCL instruction before using the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction.
An operation error may occur if the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction is executed without first executing the QTBLCL instruction.

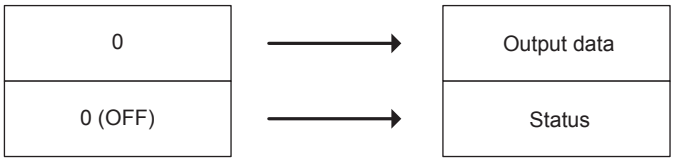
## 5.9.9 Clear Queue Table Pointers (QTBLCL)

### ( 1 ) Operation

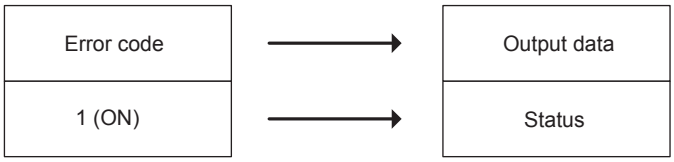
The QTBLCL instruction returns the queue read and queue write pointers to their initial values (first row) for the table data that is specified by the table name.
If the instruction ends normally, the output data is set to 0 and the status is turned OFF. If an error occurs, an error code is set in the output data and the status is turned ON.




#### [ a ] If the Queue Clear Succeeds

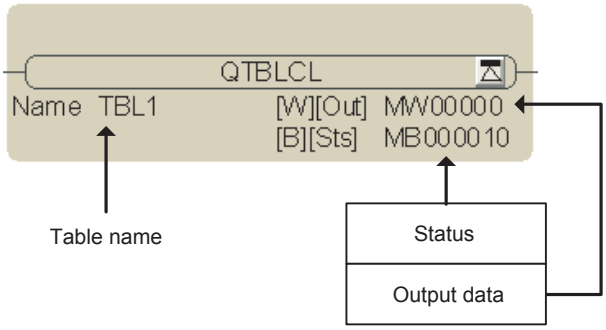



#### [ b ] If the Queue Clear Fails




- If the clear fails, the queues will retain the contents from before the instruction was executed.

## ( 2 ) Format




Icon: QTBL CL

Key entry: QTBLCL

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Output data (Out)*1 | × | ○*2 | × | × | × | ○ | × |
| Status (Sts))*1 | ○*2 | × | × | × | × | × | × |

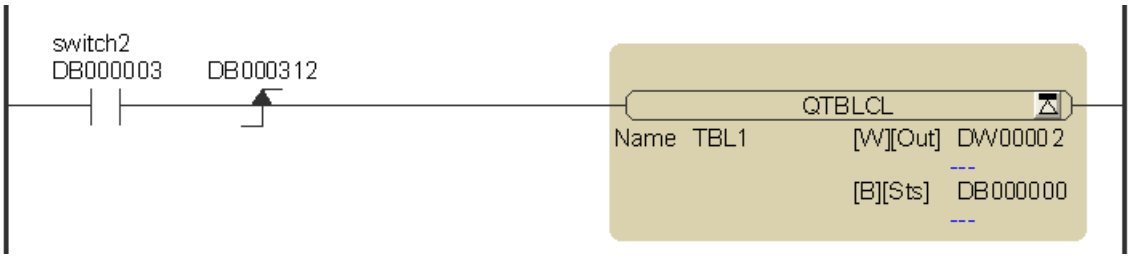
∗ 1. Optional.
∗ 2. C and # registers cannot be used.

### ■ Error Codes

| Error Code | Error Name | Meaning |
|---|---|---|
| 0001 hex | Table undefined | The target table is undefined. |
| 0002 hex | Outside range of row numbers | The row number of the table element is outside the target table. |
| 0003 hex | Outside range of column numbers | The column number of the table element is outside the target table. |
| 0004 hex | Incorrect number of elements | The number of target elements is invalid. |
| 0005 hex | Insufficient storage area | The storage area is insufficient. |
| 0006 hex | Insufficient element type | The data type specified for the element is wrong. |
| 0007 hex | Queue buffer error | An attempt was made to read from an empty queue buffer, or to write to a full queue buffer by advancing the pointer. |
| 0008 hex | Queue table error | The specified table is not a queue table. |
| 0009 hex | System error | An unexpected error was detected in the system during instruction execution. |

## ( 3 ) Programming Example

In the following programming example, the queue pointers for the specified queue table are initialized when switch 2 (DB000003) turns ON.




INFO

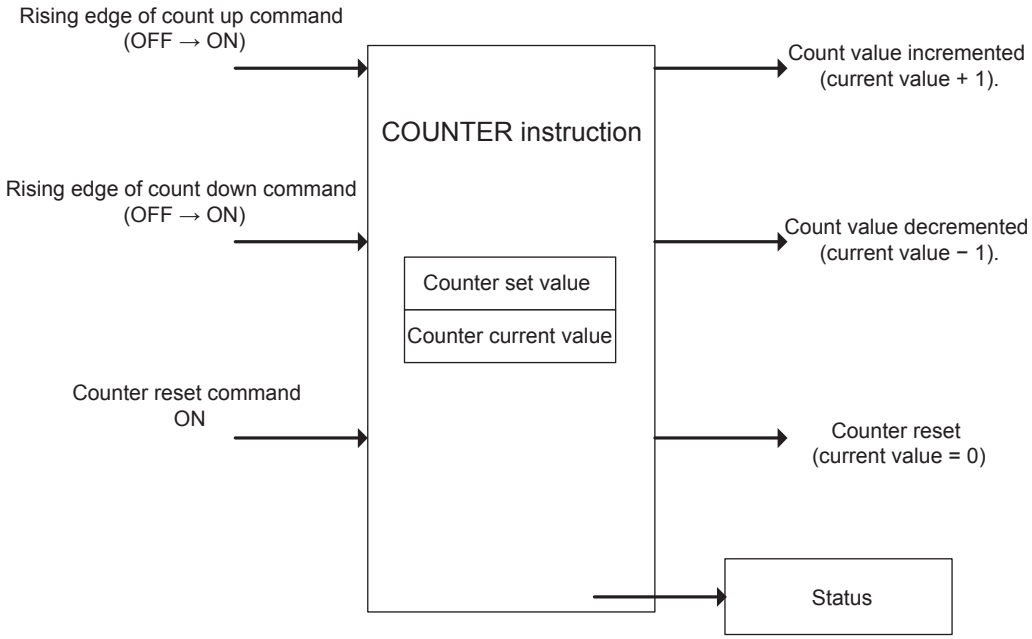
When the power is turned ON, values of the read pointer and write pointer are undefined. Always execute the QTBLCL instruction before using the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction.
An operation error may occur if the QTBLR, QTBLRI, QTBLW, or QTBLWI instruction is executed without first executing the QTBLCL instruction.

# 5.10 System Function Instructions

## 5.10.1 Counter (COUNTER)

### ( 1 ) Operation

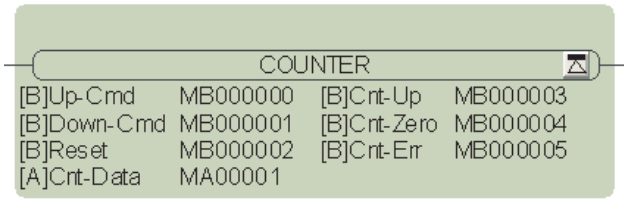
When the count up or count down command changes from OFF to ON, the current value is incremented or decremented.
When the counter reset command turns ON, the current value of the counter is set to 0. The current value of the counter is compared against the set value and the result is output.
If a counter error occurs (i.e., if the current value is greater than the set value), the current value will neither be incremented nor decremented.




Three status are output as shown below.

- Count matched (current value = set value).
- Count is zero (current value = 0).
- Counter error
  (current value > set value or current value < 0).

## ( 2 ) Format




Icon: COUN TER

Key entry: COUNTER

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Count up command (Up-Cmd) | ○ | × | × | × | × | × | × |
| Count down command (Down-Cmd) | ○ | × | × | × | × | × | × |
| Counter reset command (Reset) | ○ | × | × | × | × | × | × |
| First address of counter processing data area (Cnt-Data) | × | × | × | × | ○*1 | × | × |
| Count up (Cnt-Up) | ○*2 | × | × | × | × | × | × |
| Zero count (Cnt-Zero) | ○*2 | × | × | × | × | × | × |
| Count error (Cnt-Err) | ○*2 | × | × | × | × | × | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

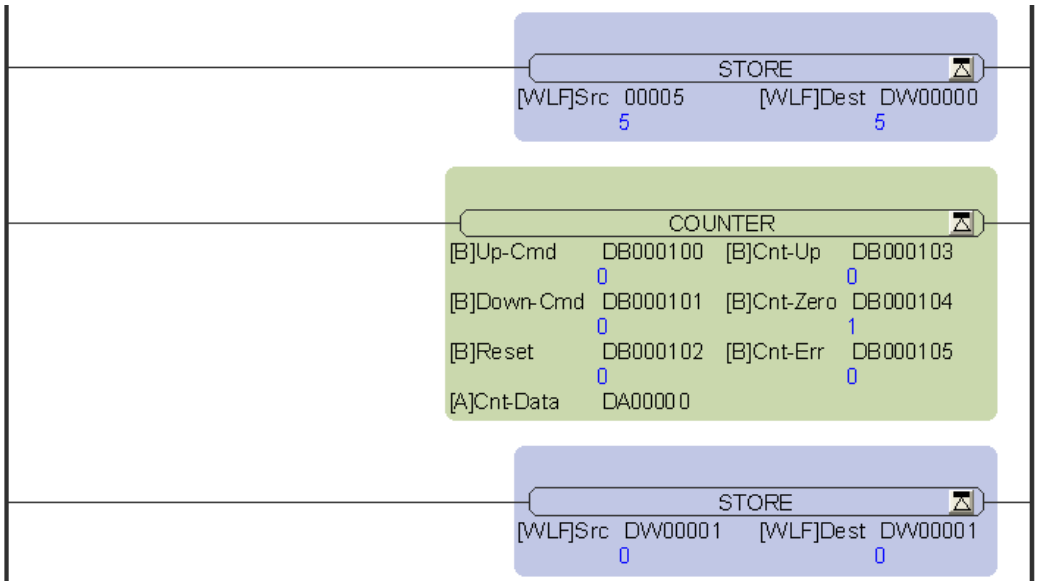
The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Count up command (Up-Cmd) | The count value is incremented when this command changes from OFF to ON.* | IN |
| Count down command (Down-Cmd) | The count value is decremented when this command changes from OFF to ON.* | IN |
| Counter reset command (Reset) | The current value is reset to 0 when this command turns ON. | IN |
| First address of counter processing data area (Cnt-Data) | +0 word: Set value | IN |
| | +1 word: Current value | OUT |
| | +2 word: Work flags | OUT |
| Count up (Cnt-Up) | Turns ON when the current value equals the set value. | OUT |
| Zero count (Cnt-Zero) | Turns ON when the current value equals 0. | OUT |
| Count error (Cnt-Err) | Turns ON when the current value is greater than the set value.<br>Also turns ON when the current value is less than 0. | OUT |

∗ If the count up command and count down command change from OFF to ON at the same time, the current value stays the same.

## ( 3 ) Programming Example

In the following programming example, the first line sets the counter set value to 5, and the third line monitors the counter current value in DW00001.
When DB000100 changes from OFF to ON, DW00001 is incremented, and when DB000101 changes from OFF to ON, DW00001 is decremented.



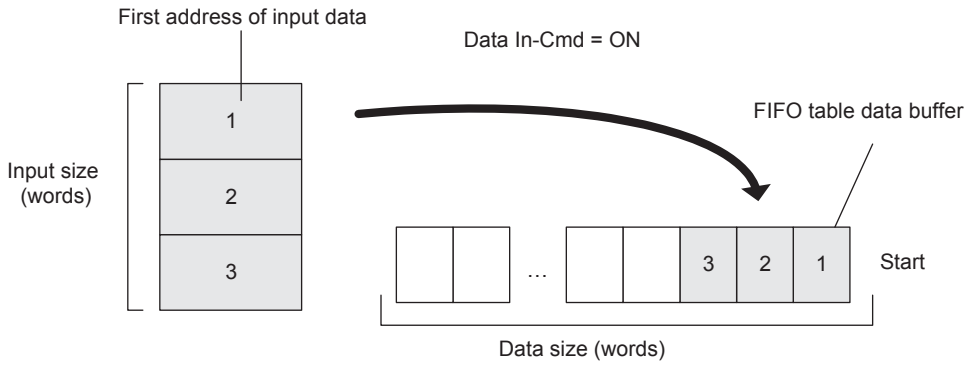
# 5.10.2 First-in First-out (FINFOUT)

## ( 1 ) Operation

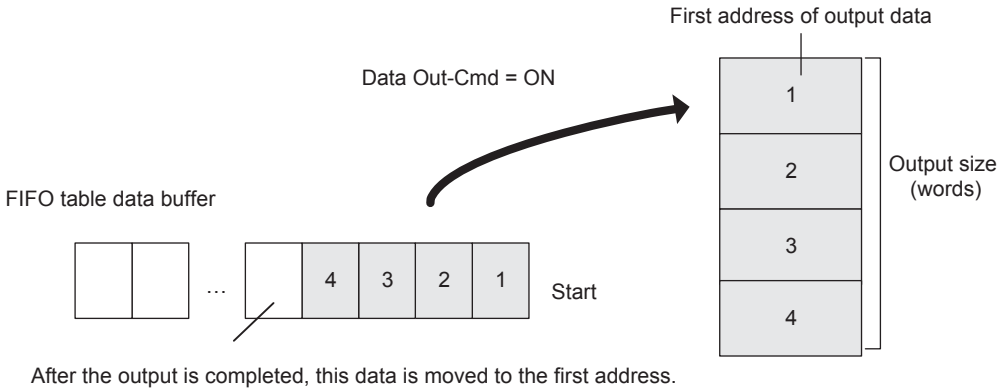
The FINFOUT instruction calls a first-in first-out block data transfer function. The FIFO data table consists of a 4-word header and a data buffer. Always set the three words with the data size, input size, and output size before you execute this instruction.

- When the Data Input Command (In-Cmd) turns ON, the specified number of data items from the specified input data area are stored sequentially in the data area of the FIFO table.
- When the Data Output Command (Out-Cmd) turns ON, the specified number of data items are moved from the first address in the data area of the FIFO table to the specified output data area.
- When the Reset Command (Reset) turns ON, the number of stored words is set to 0 and Tbl-Emp (FIFO table empty) turns ON.
- If the data empty size is less than the input size or if the data size is less than the output size, Tbl-Err (FIFO table error) turns ON.

### [ a ] If the Data Input Command (In-Cmd) Is ON




### [ b ] If the Data Option Command (Out-Cmd) Is ON

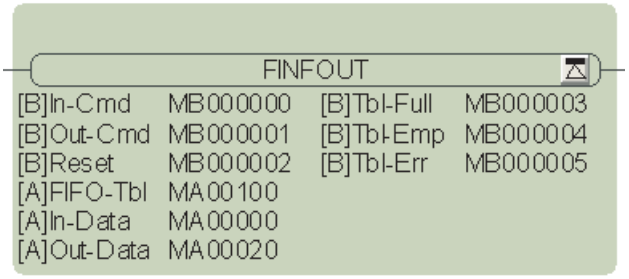



### [ c ] If the Reset Command (Reset) Is ON

The number of words stored in the FIFO table is set to 0.

- The contents of the table buffer are retained and not cleared to 0.

## ( 2 ) Format




Icon: FIN FOUT

Key entry: FINFOUT

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Data input command (In-Cmd) | ○ | × | × | × | × | × | × |
| Data output command (Out-Cmd) | ○ | × | × | × | × | × | × |
| Reset command (Reset) | ○ | × | × | × | × | × | × |
| First address of FIFO table (FIFO-Tbl) | × | × | × | × | ○*1 | × | × |
| First address of input data (In-Data) | × | × | × | × | ○*1 | × | × |
| First address of output data (Out-Data) | × | × | × | × | ○*1 | × | × |
| FIFO table full (Tbl-Full) | ○*2 | × | × | × | × | × | × |
| FIFO table empty (Tbl-Emp) | ○*2 | × | × | × | × | × | × |
| FIFO table error (Tbl-Err) | ○*2 | × | × | × | × | × | × |

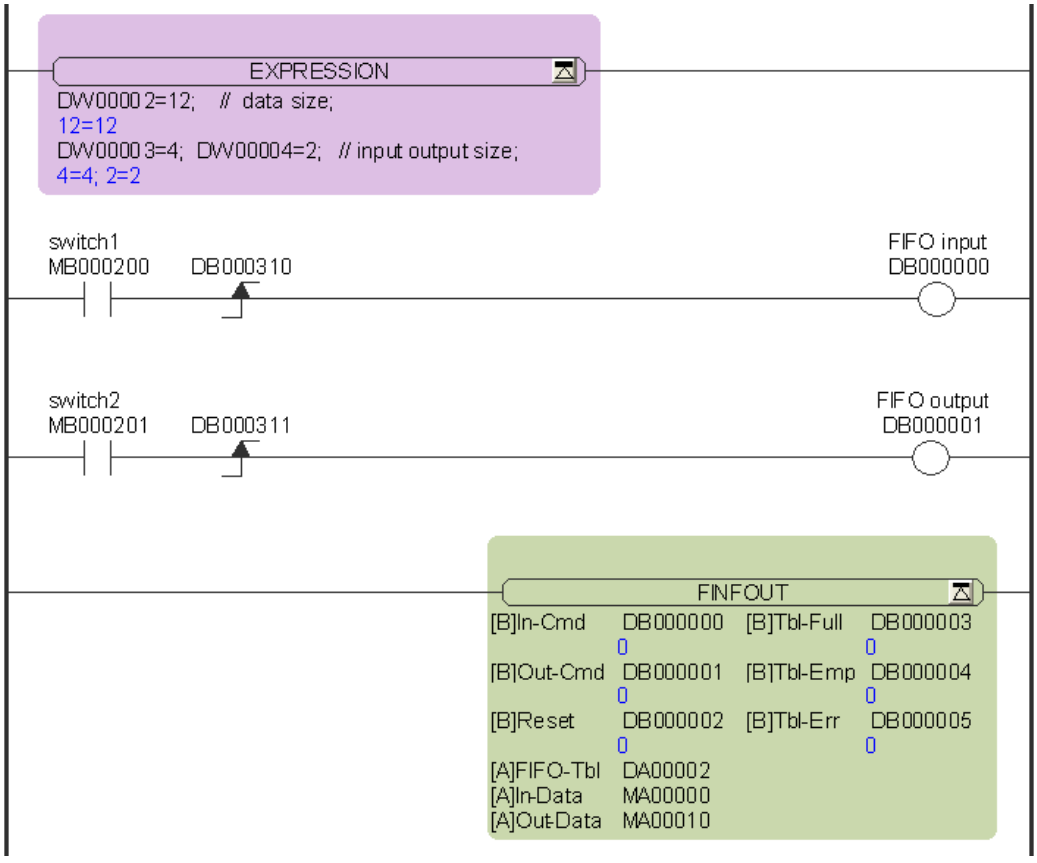
∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Data input command (In-Cmd) | Data is stored in the FIFO table when this command turns ON. | IN |
| Data output command (Out-Cmd) | Data is transferred out of the FIFO table when this command turns ON. | IN |
| Reset command (Reset) | The number of words to store is set to 0 when this command turns ON. | IN |
| First address of FIFO table (FIFO-Tbl) | +0 word: Data size | IN |
| | +1 word: Input size | IN |
| | +2 word: Output size | IN |
| | +3 word: Data storage size | OUT |
| | +4 word and on: Data | OUT |
| First address of input data (In-Data) | First address of input data | IN |
| First address of output data (Out-Data) | First address of output data | IN |
| FIFO table full (Tbl-Full) | Turns ON when the FIFO table is full. | OUT |
| FIFO table empty (Tbl-Emp) | Turns ON when the FIFO table is empty. | OUT |
| FIFO table error (Tbl-Err) | Turns ON when the FIFO table has an error. | OUT |

## ( 3 ) Programming Example

In the following programming example, a FIFO table is created with a data size of 12 words, input size of 4 words, and an output size of 2 words, and then the FINFOUT instruction is executed.



The data from MW00000 to MW00003 is stored in the FIFO table buffer when switch 1 turns ON.
The data storage size in DW00005 is set to 4.

| Register | Data |
|---|---|
| MW00000 | 123 |
| MW00001 | 234 |
| MW00002 | 345 |
| MW00003 | 456 |

| FIFO Table Data Buffer | Data |
|---|---|
| DW00006 | 123 |
| DW00007 | 234 |
| DW00008 | 345 |
| DW00009 | 456 |
| DW00010 | 0 |
| : | : |
| DW00017 | 0 |

Stored area

Next, when switch 2 turns ON, two words of data from the first address in the FIFO table buffer are output to the area from MW0010 to MW0011. The data storage size in DW00005 is set to 2.

| FIFO Table Data Buffer | Data |
|---|---|
| DW00006 | 123<br>→ 345 |
| DW00007 | 234<br>→ 456 |

| Register | Data |
|---|---|
| MW00010 | 123 |
| MW00011 | 234 |
| MW00012 | 0 |
| | |

Stored area

The output data is moved out and the remaining data is moved forward.

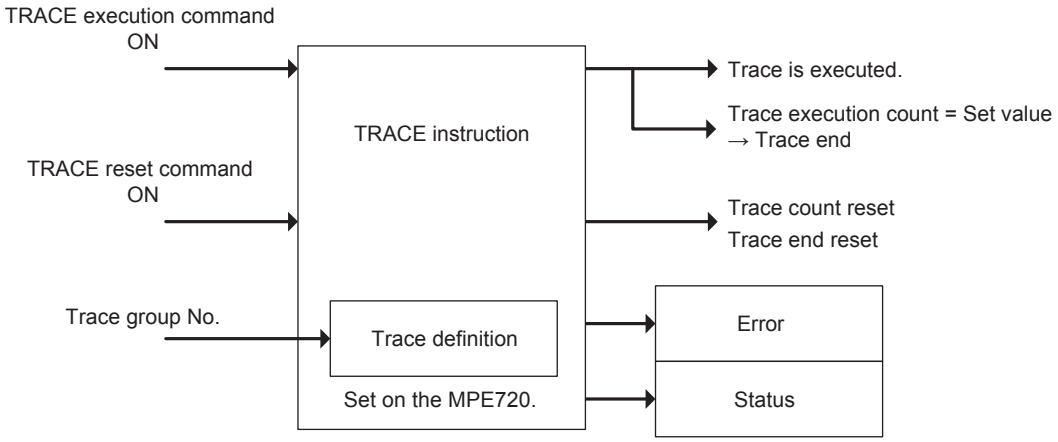
# 5.10.3 Trace (TRACE)

## ( 1 ) Operation

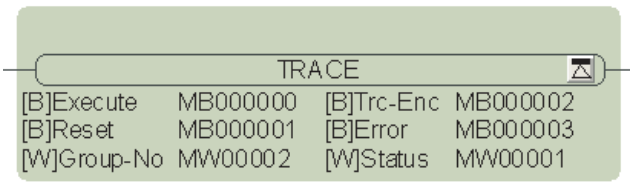
The TRACE instruction performs trace execution control of the trace data that is specified by the trace group number (1 to 4).

- The trace definition is set in the Data Trace Definitions in the MPE720. Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for details.

- The trace is executed if Execute (trace execution command) is ON.
- The trace counter is reset when Reset (trace reset command) turns ON. This also resets Trc-End (trace end).
- Trc-End (trace end) turns ON when the specified number of traces have been executed.




## ( 2 ) Format




Icon: TRACE

Key entry: TRACE

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Trace execution command (Execute) | ○ | × | × | × | × | × | × |
| Trace reset command (Reset) | ○ | × | × | × | × | × | × |
| Trace group No. (Group-No) | × | ○ | × | × | × | ○ | ○ |
| Trace end (Trc-End) | ○* | × | × | × | × | × | × |
| Error | ○* | × | × | × | × | × | × |
| Status | × | ○* | × | × | × | ○ | × |

* C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Trace execution command (Execute) | Trace execution begins when this command turns ON. | IN |
| Trace reset command (Reset) | Trace execution is reset when this command turns ON. | IN |
| Trace group No. (Group-No) | Trace group No. specification (1 to 4) | IN |
| Trace end (Trc-End) | Turns ON when the trace ends. | OUT |
| Error | Turns ON when an error occurs. | OUT |
| Status | Trace execution status | OUT |

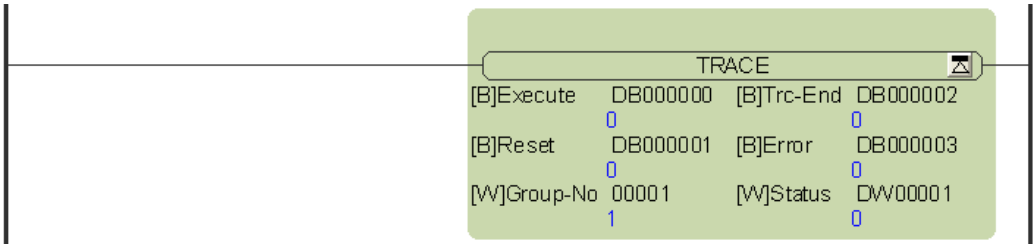
The status configuration is shown below.

| Bit | Name | Remarks |
|---|---|---|
| 0 | Trace data full | Turns ON after once going through the data trace memory of the specified group. |
| 1 to 7 | Reserved for system. | – |
| 8 | No trace definition | The function will not be executed. |
| 9 | Group No. error | The function will not be executed. |
| 10 to 12 | Reserved for system. | – |
| 13 | Execution timing error | The function will not be executed. |
| 14 | Reserved for system. | – |
| 15 | Reserved for system. | – |

## ( 3 ) Programming Example

In the following programming example, the definition for trace group number 1 is used to execute a trace. The trace starts when DB000000 turns ON.

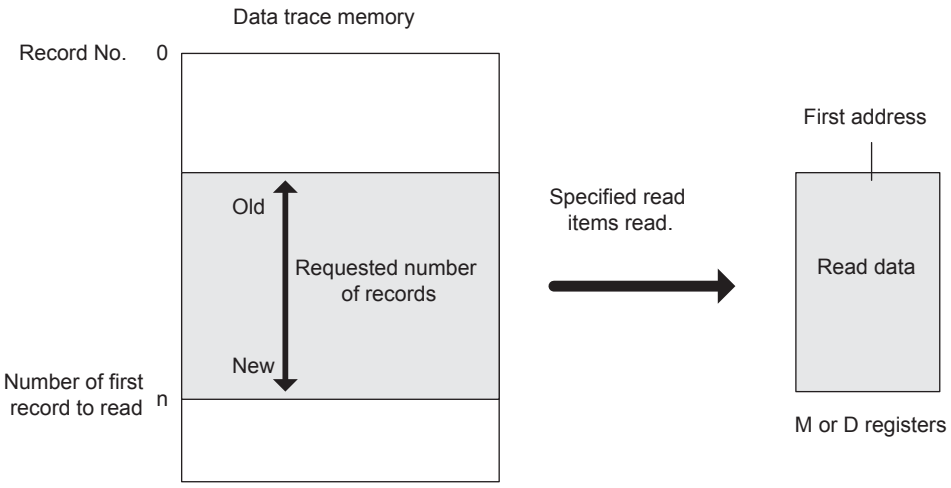
- Set the data trace definition for trace group number 1 on the MPE720 in advance. Make sure to set the sampling condition to *Program*.



# 5.10.4 Read Data Trace (DTRC-RD)

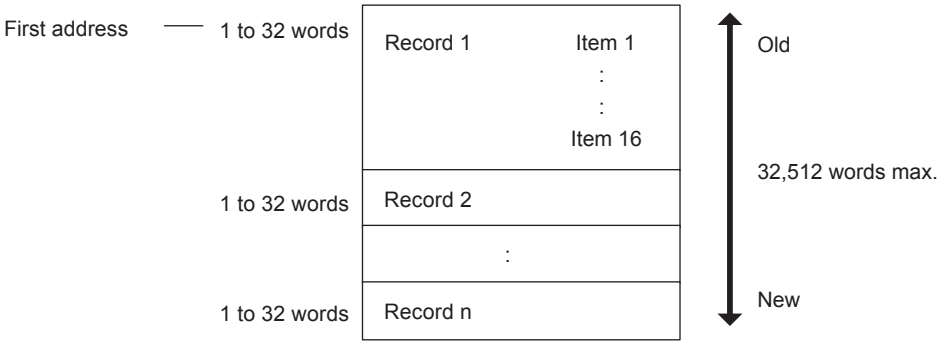
## ( 1 ) Operation

The DTRC-RD instruction reads trace data in the Machine Controller and stores it in registers. The data in the trace memory can be read by specifying the first record number and the number of records. You can designate and read only the required items in a record.

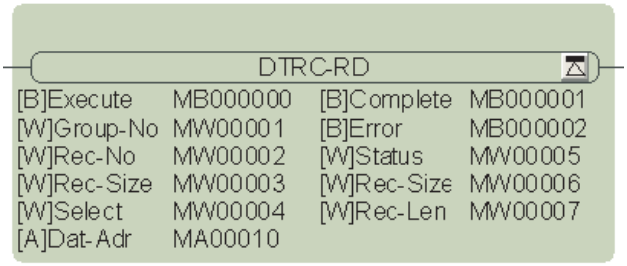



### ■ Structure of Read Data

The length of a record can be from 1 to 32 words, depending on the selected data items. The maximum number of records can be from 1,015 to 32,511 depending on the record length.



## ( 2 ) Format




Icon: DTRC -RD

Key entry: DTRCRD

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Trace read execution command (Execute) | ○ | × | × | × | × | × | × |
| Trace group No. (Group-No) | × | ○ | × | × | × | × | × |
| Record No. (Rec-No) | × | ○ | × | × | × | × | × |
| Number of records (Rec-Size) | × | ○ | × | × | × | × | × |
| Item selection (Select) | × | ○ | × | × | × | × | × |
| First address (Dat-Adr) | × | × | × | × | ○*1 | × | × |
| Trace completed (Complete) | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |
| Status | × | ○*2 | × | × | × | × | × |
| Number of records read (Rec-Size) | × | ○*2 | × | × | × | × | × |
| Length of 1 read record (Rec-Len) | × | ○*2 | × | × | × | × | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

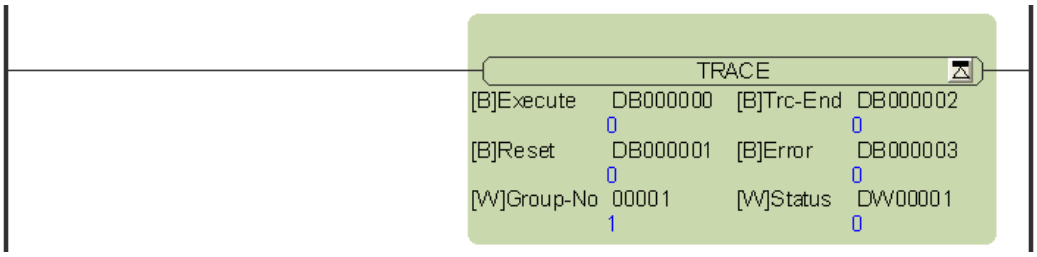
| Parameter Name | Description | I/O |
|---|---|---|
| Trace read execution command (Execute) | Data trace read execution command | IN |
| Trace group No. (Group-No) | Data trace group No. (1 to 4). | IN |
| Record No. (Rec-No) | Number of first record to read<br>(0 to maximum records – 1) | IN |
| Number of records (Rec-Size) | Requested number of records to read<br>(0 to maximum records – 1) | IN |
| Item selection (Select) | Items to read (0001 to FFFF hex)<br>Bits 0 to F correspond to data specifiers 1 to 16 in the trace definition. | IN |
| First address (Dat-Adr) | Number of first register to read (MA, DA) | IN |
| Trace completed (Complete) | Turns ON when the trace read ends. | OUT |
| Error | Turns ON when an error occurs. | OUT |
| Status | Data trace read execution status | OUT |
| Number of records read (Rec-Size) | Number of records that were read | OUT |
| Length of 1 read record (Rec-Len) | Length of 1 read record (words) | OUT |

The status configuration is shown below.

| Bit | Name | Remarks |
|---|---|---|
| 0 to 7 | Reserved for system. | – |
| 8 | No trace definition | The function will not be executed. |
| 9 | Group No. error | The function will not be executed. |
| 10 | Specified record No. error | The function will not be executed. |
| 11 | Specified number of records error | The function will not be executed. |
| 12 | Data storage error | The function will not be executed. |
| 13 | Reserved for system. | – |
| 14 | Reserved for system. | – |
| 15 | Address input error | The function will not be executed. |

## ( 3 ) Programming Example

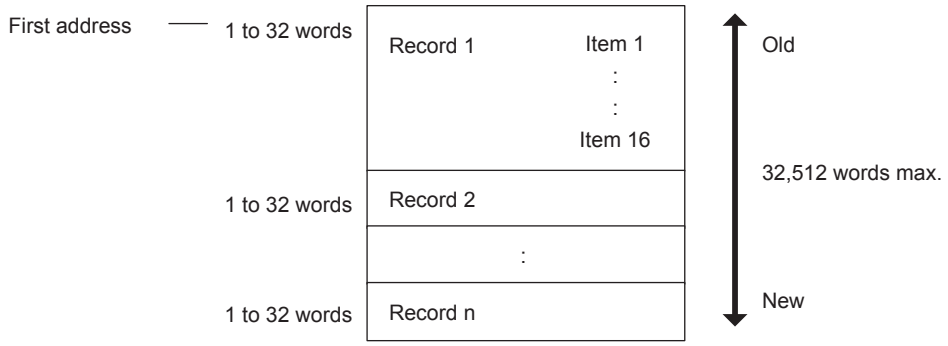
In the following programming example, a data trace is executed for group definition number 1.
The trace is executed when DB000000 turns ON.



## ( 4 ) Additional Information

### [ a ] Structure of Read Data

The read data is structured as shown in the following figure.




### [ b ] Record Lengths

A record consists of the selected data items.
The record length (number of words in a single record) is determined by the selected registers and the number of data items.

- Number of words for 1 record = Bn × 1 word + Wn × 1 word + Ln × 2 words + Fn × 2 words
  Bn: Number of selected bit registers
  Wn: Number of selected integer registers
  Ln: Number of selected double-length integer registers
  Fn: Number of selected real number registers
  The maximum total is 16 registers.
- Maximum record length = 32 words (with 16 double-length integers or real number registers)
- Minimum record length = 1 word (with 1 record for each bit or integer register)

### [ c ] Number of Records

The number of records that can be specified depends on the record length as shown below.

- Number of records with the maximum record length: 0 to 1,015
- Number of records with the minimum record length: 0 to 32,511
  (Upper limit: 32,521 divided by the record length - 1)

### [ d ] Latest Record Number

The most recent record number for each trace group is stored in the system registers as shown below.

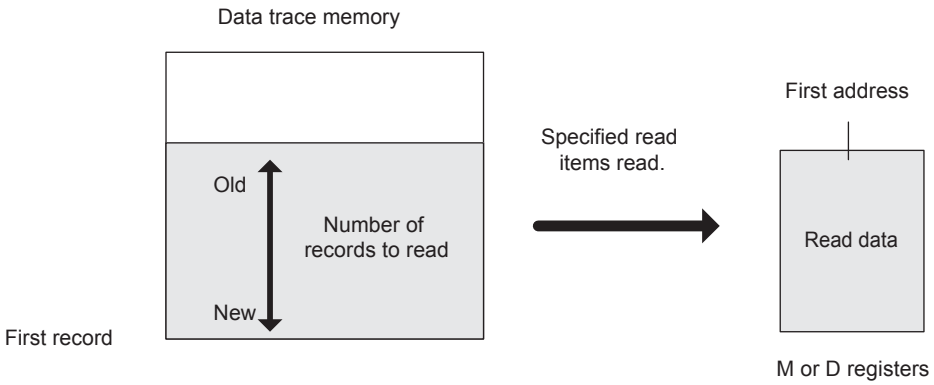
| System Register Address | Description |
|---|---|
| SW00100 | Latest record number in group 1. |
| SW00101 | Latest record number in group 2. |
| SW00102 | Latest record number in group 3. |
| SW00103 | Latest record number in group 4. |
| SW00104 | – |
| SW00105 | – |
| SW00106 | – |
| SW00107 | – |

# 5.10.5 Read Inverter Trace (ITRC-RD)

## ( 1 ) Operation

The ITRC-RD instruction reads trace data in the Inverter and stores it in registers. You can specify the required records and read them from the trace buffer. You can designate and read only the required items in each record.

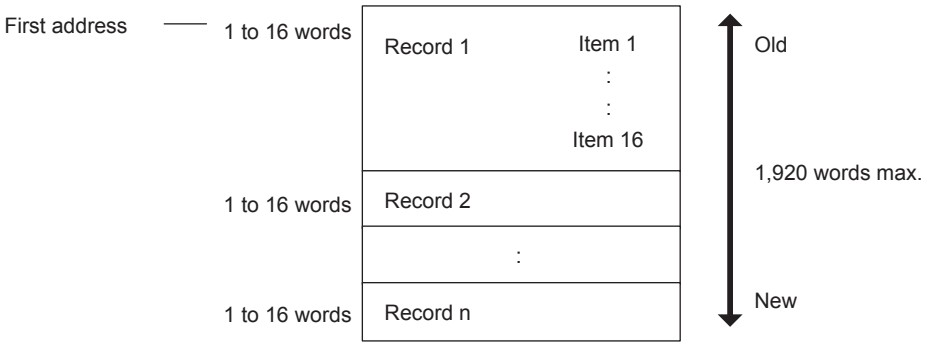
Applicable Inverters:
This instruction is applicable to Inverters that are connected to the 215IF.



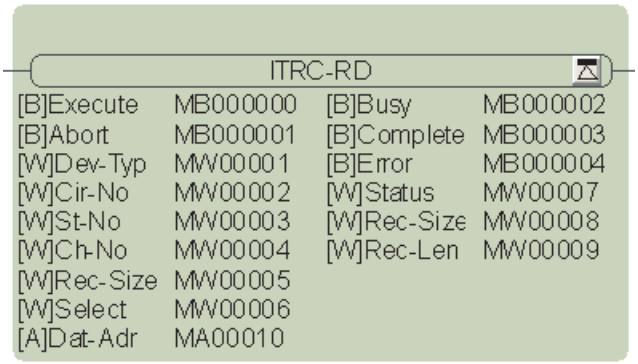

### ■ Structure of Read Data

The length of a record can be from 1 to 16 words, depending on the selected data items.
The maximum number of records is 120.

- Records are always read from the first record.



## ( 2 ) Format




Icon: ITRC -RD

Key entry: ITRC-RD

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Trace read execution command (Execute) | ○ | × | × | × | × | × | × |
| Trace read abort command (Abort) | ○ | × | × | × | × | × | × |
| Communications device type (Dev-Typ) | × | ○ | × | × | × | ○ | ○ |
| Circuit number (Cir-No) | × | ○ | × | × | × | ○ | ○ |
| Slave station number (St-No) | × | ○ | × | × | × | ○ | ○ |
| Communications buffer channel number (Ch-No) | × | ○ | × | × | × | ○ | ○ |
| Number of records (Rec-Size) | × | ○ | × | × | × | ○ | ○ |
| Item selection (Select) | × | ○ | × | × | × | ○ | ○ |
| First address (Dat-Adr) | × | × | × | × | ○*1 | × | × |
| Busy | ○*2 | × | × | × | × | × | × |
| Complete | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |
| Status | × | ○*2 | × | × | × | ○ | × |
| Number of records read (Rec-Size) | × | ○*2 | × | × | × | ○ | × |
| Read record length (Rec-Len) | × | ○*2 | × | × | × | ○ | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

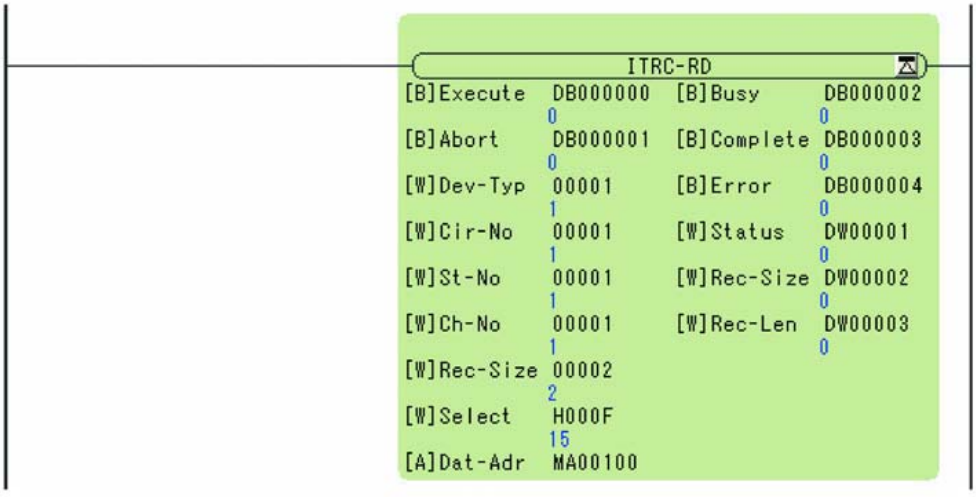
| Parameter Name | Description | I/O |
|---|---|---|
| Trace read execution command (Execute) | Reading begins when this command turns ON. | IN |
| Trace read abort command (Abort) | Reading is aborted when this command turns ON. | IN |
| Communications device type (Dev-Typ) | 215IF = 1 | IN |
| Circuit number (Cir-No) | 215IF = 1 or 2 | IN |
| Slave station number (St-No) | 215IF = 1 to 64 | IN |
| Communications buffer channel number (Ch-No) | 215IF = 1 to 3 | IN |
| Number of records (Rec-Size) | Number of records to read (1 to 64) | IN |
| Item selection (Select) | Items to read (0001 to FFFF hex)<br>Bits 0 to F corresponds to trace data items 1 to 26. | IN |
| First address (Dat-Adr) | First register address to read at source (MA, DA) | IN |
| Busy | Turns ON while reading Inverter trace data is in progress. | OUT |
| Complete | Turns ON when reading Inverter trace data is completed. | OUT |
| Error | Turns ON when an error occurs. | OUT |
| Status | Inverter trace data read execution status | OUT |
| Number of records read (Rec-Size) | Number of records that were read | OUT |
| Read record length (Rec-Len) | Length of records that were read | OUT |

The status configuration is shown below.

| Bit | Name | Remarks |
|---|---|---|
| 0 to 8 | Reserved for system. | – |
| 9 | Communications parameter error | The function will not be executed. |
| 10 | Reserved for system. | – |
| 11 | Specified number of records error | The function will not be executed. |
| 12 | Data storage error | The function will not be executed. |
| 13 | Communications error | The function will not be executed. |
| 14 | Reserved for system. | – |
| 15 | Address input error | The function will not be executed. |

## ( 3 ) Programming Example

In the following programming example, trace data is read from an Inverter.
Two records of trace data are read from the Inverter that is connected to station 1 of the 215IF on circuit 1. The data is stored in the area that starts with MW00100.

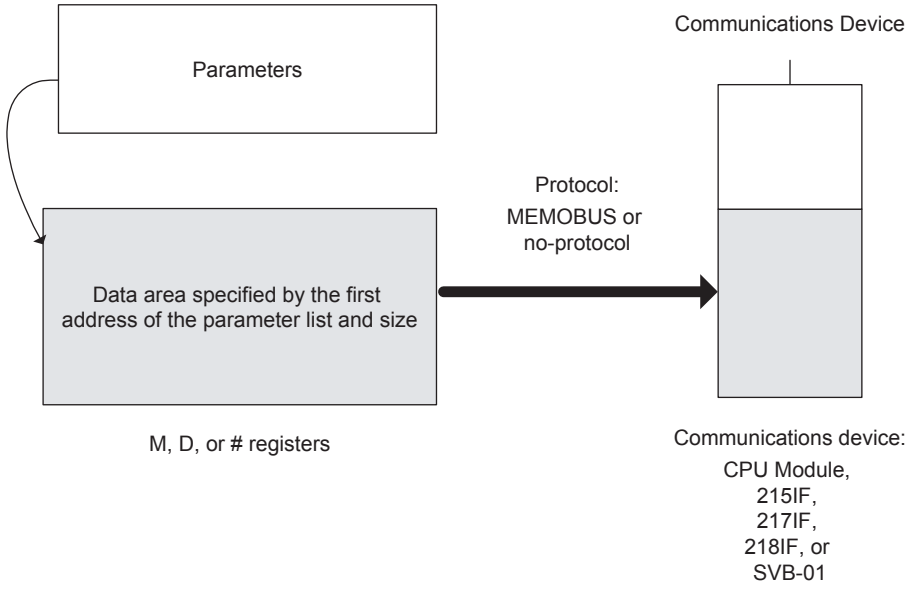


# 5.10.6 Send Message (MSG-SND)

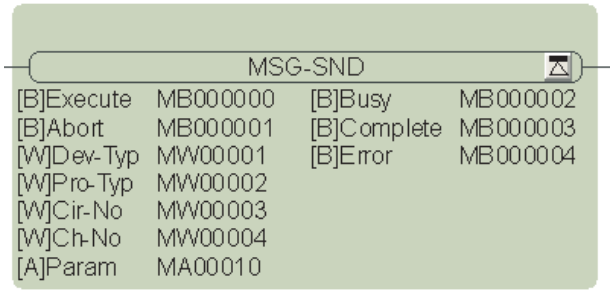
## ( 1 ) Operation

The MSG-SND instruction sends a message to a remote station of the specified communications device type on the specified circuit.
This instruction supports the following communications devices and protocols.

Communications devices: CPU Module, 215IF, 217IF, 218IF, and SVB-01
Protocol: MEMOBUS communications or no-protocol



## ( 2 ) Format




Icon: MSG -SND

Key entry: MSG-SND

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Send execution command (Execute) | ○ | × | × | × | × | × | × |
| Send abort command (Abort) | ○ | × | × | × | × | × | × |
| Communications device type (Dev-Typ) | × | ○ | × | × | × | ○ | ○ |
| Communications protocol (Pro-Typ) | × | ○ | × | × | × | ○ | ○ |
| Circuit number (Cir-No) | × | ○ | × | × | × | ○ | ○ |
| Communications buffer channel number (Ch-No) | × | ○ | × | × | × | ○ | ○ |
| First address of parameter list (Param) | × | × | × | × | ○*1 | × | × |
| Busy | ○*2 | × | × | × | × | × | × |
| Complete | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Send execution command (Execute) | The message is sent when this command turns ON. | IN |
| Send abort command (Abort) | Sending the message is aborted when this command turns ON. | IN |
| Communications device type (Dev-Typ) | CPU Module = 8, 215IF = 1, 217IF = 5, 218IF = 6, 218IF-02 = 16, and SVB-01 = 10 | IN |
| Communications protocol (Pro-Typ) | MEMOBUS = 1, No-protocol = 2 | |
| Circuit number (Cir-No) | CPU Module = 1 or 2, 215IF = 1 to 8, 217IF = 1 to 24, 218IF(-02) = 1 to 8, and SVB-01 = 1 to 16 | IN |
| Communications buffer channel number (Ch-No) | CPU Module = 1 or 2, 215IF = 1 to 13, 217IF = 1, 218IF(-02) = 1 to 10, and SVB-01 = 1 to 8 | IN |
| First address of parameter list (Param) | First address of parameter list (MA, DA, or #A) | IN |
| Busy | Turns ON while sending the message is in progress. | OUT |
| Complete | Turns ON when sending the message is completed. | OUT |
| Error | Turns ON when an error occurs. | OUT |

### [ a ] Parameter Details

This section describes the parameters in detail. The parameter number corresponds to the word offset from the first address of the parameter list.

For example, if the first address of the parameter list is MA00100, set the value in MW00110 to set PARAM10.

| Parameter No. | IN/OUT | Description | |
|---|---|---|---|
| | | MEMOBUS | No-protocol |
| PARAM00 | OUT | Processing result | Processing result |
| PARAM01 | OUT | Status | Status |
| PARAM02 | IN | Remote station number | Remote station number |
| PARAM03 | SYS | Reserved for system. | Reserved for system. |
| PARAM04 | IN | Function code | – |
| PARAM05 | IN | Data address | Data address |
| PARAM06 | IN | Data size | Data size |
| PARAM07 | IN | Remote CPU number | Remote CPU number |
| PARAM08 | IN | Coil offset | – |
| PARAM09 | IN | Input relay offset | – |
| PARAM10 | IN | Input register offset | – |
| PARAM11 | IN | Hold register offset | Register offset |
| PARAM12 | SYS | Reserved for system. | Reserved for system. |
| PARAM13 | SYS | Reserved for system. | Reserved for system. |
| PARAM14 | SYS | Reserved for system. | Reserved for system. |
| PARAM15 | SYS | Reserved for system. | Reserved for system. |
| PARAM16 | SYS | Reserved for system. | Reserved for system. |

#### ■ Processing Result (PARAM00)

This parameter outputs the result of processing to the upper byte. The lower byte is used for system analysis.

- 00□□ hex: Processing (Busy)
- 10□□ hex: Processing completed (Complete)
- 8□□□ hex: Error

  The following errors can occur.

  - 81□□ hex: Function code error

    An attempt was made to send an unused function code. Or an unused function code was received.
  - 82□□ hex: Address setting error

    The data address, coil offset, input relay offset, input register offset, or hold register offset is outside of the valid range.
  - 83□□ hex: Data size error

    The size of the send or receive data was set outside of the valid range.
  - 84□□ hex: Circuit number setting error

    The set circuit number is outside of the valid range.
  - 85□□ hex: Channel number setting error

    The set channel number is outside of the valid range.
  - 86□□ hex: Station address error

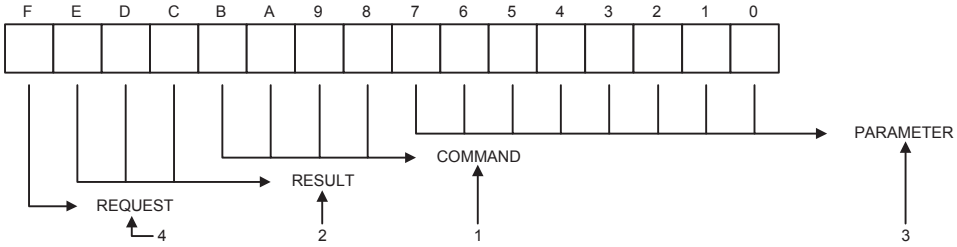
    The set station number is outside of the valid range.
  - 88□□ hex: Communications section error

    The communications section returned an error response.
  - 89□□ hex: Device selection error

    A device that cannot be used was selected.

## ■ Status (PARAM01)

The status of the communications section is output to this parameter.
The bit assignments are shown in the following figure.




### 1. COMMAND

The abbreviations and meanings of the commands are given in the following table.

| Code | Abbreviation | Meaning |
|---|---|---|
| 1 | U_SEND | Sends a general-purpose message. |
| 2 | U_REC | Receives a general-purpose message. |
| 3 | ABORT | Aborts operation. |
| 8 | M_SEND | Sends a MEMOBUS command and ends when a response is received. |
| 9 | M_REC | Receives a MEMOBUS command and returns a response. |
| C | MR_SEND | Sends a MEMOBUS response. |

### 2. RESULT

The abbreviations and meanings of the results are listed in the following table.

| Code | Abbreviation | Meaning |
|---|---|---|
| 0 | – | Execution is in progress. |
| 1 | SEND_OK | The send was completed normally. |
| 2 | REC_OK | The reception was completed normally. |
| 3 | ABORT_OK | Aborting was completed. |
| 4 | FMT_NG | A parameter format error occurred. |
| 5 | SEQ_NG or INIT_NG | A command sequence error occurred or a token was not received. There is no connection to a communications system. |
| 6 | RESET_NG or O_RING_NG | A reset status exists.<br>Out of ring: A token was not received within the token monitoring time. |
| 7 | REC_NG | A data reception error occurred. (An error was detected in a low-level program.) |

### 3. PARAMETER

If the RESULT is 4 (FMT_NG), one of the following error codes is given. Otherwise, the remote station address is given.

| Code | Error |
|---|---|
| 00 | No error |
| 01 | Station address out of range |
| 02 | MEMOBUS response monitoring time error |
| 03 | Number of retries setting error |
| 04 | Cyclic area setting error |
| 05 | Message signal CPU number error |
| 06 | Message signal register address error |
| 07 | Message signal number of words error |

### 4. REQUEST

1 = Request
2 = Reception completed notification

## ■ Remote Station Number (PARAM02)/Serial

1 to 254: The message is sent to the station with the specified device address.

## ■ Function Code (PARAM04)

Set the MEMOBUS function code to send.
The function codes are listed in the following table.

| Function Code | | Setting |
|---|---|---|
| 00 hex | Not used. | × |
| 01 hex | Read Coil Status | ○ |
| 02 hex | Read Input Relay Status | ○ |
| 03 hex | Read Hold Register Contents | ○ |
| 04 hex | Read Input Register Contents | ○ |
| 05 hex | Change Single Coil Status | ○ |
| 06 hex | Write Single Hold Register | ○ |
| 07 hex | Not used. | × |
| 08 hex | Loopback Test | ○ |
| 09 hex | Expanded Read Hold Register Contents | ○ |
| 0A hex | Expanded Read Input Register Contents | ○ |
| 0B hex | Expanded Write Hold Register | ○ |
| 0C hex | Not used. | × |
| 0D hex | Expanded Read Nonconsecutive Hold Registers | ○ |
| 0E hex | Expanded Write Nonconsecutive Hold Registers | ○ |
| 0F hex | Change Multiple Coil Status | ○ |
| 10 hex | Write Multiple Hold Registers | ○ |
| 11 hex to 20 hex | Not used. | × |
| 21 hex to 3F hex | Reserved for system. | × |
| 40 hex to 4F hex | Reserved for system. | × |
| 50 hex and higher | Not used. | × |

- ○: Can be set, ×: Cannot be set.
- When the target device is operating as the master, only MW and MB data can be read and written. When the target device is operating as a slave, MB, MW, IB, and IW data can be read and written for coils, hold registers, input relays, and input registers, respectively.

### ■ Data Addresses

The range of addresses that can be set for each function code are given in the following table.

| Function Code | | Setting |
|---|---|---|
| 00 hex | Not used. | Not valid. |
| 01 hex | Read Coil Status | 0 to 65,535 (0 to FFFF hex)* |
| 02 hex | Read Input Relay Status | 0 to 65,535 (0 to FFFF hex)* |
| 03 hex | Read Hold Register Contents | 0 to 32,767 (0 to 7FFF hex) |
| 04 hex | Read Input Register Contents | 0 to 32,767 (0 to 7FFF hex) |
| 05 hex | Change Single Coil Status | 0 to 65,535 (0 to FFFF hex)* |
| 06 hex | Write Single Hold Register | 0 to 32,767 (0 to 7FFF hex) |
| 07 hex | Not used. | Not valid. |
| 08 hex | Loopback Test | Not valid. |
| 09 hex | Expanded Read Hold Register Contents | 0 to 32,767 (0 to 7FFF hex) |
| 0A hex | Expanded Read Input Register Contents | 0 to 32,767 (0 to 7FFF hex) |
| 0B hex | Expanded Write Hold Register | 0 to 32,767 (0 to 7FFF hex) |
| 0C hex | Not used. | Not valid. |
| 0D hex | Expanded Read Nonconsecutive Hold Registers | 0 to 32,767 (0 to 7FFF hex) |
| 0E hex | Expanded Write Nonconsecutive Hold Registers | 0 to 32,767 (0 to 7FFF hex) |
| 0F hex | Change Multiple Coil Status | 0 to 65,535 (0 to FFFF hex)* |
| 10 hex | Write Multiple Hold Registers | 0 to 32,767 (0 to 7FFF hex) |

* Requests to read or write relays or coils: Set the first bit address of the data.
Requests to read or write consecutive registers: Set the first word address of the data.
Requests to read or write nonconsecutive registers: Set the first word address of the address table.

### [ b ] Data Size (PARAM06)

Set the data size (number of bits or number of words) for the read or write request. The setting range depends on the function code.
The setting ranges for serial data sizes are listed in the following table.

| Function Code | | Data Size Setting Range | |
|---|---|---|---|
| | | 215IF or 218IF | CPU Module, 217IF, or SVB-01 |
| 00 hex | Not used. | Not valid. | |
| 01 hex | Read Coil Status | 1 to 2,000 (1 to 07D0 hex) bits | |
| 02 hex | Read Input Relay Status | 1 to 2,000 (1 to 07D0 hex) bits | |
| 03 hex | Read Hold Register Contents | 1 to 125 (1 to 007D hex) words | |
| 04 hex | Read Input Register Contents | 1 to 125 (1 to 007D hex) words | |
| 05 hex | Change Single Coil Status | Not valid. | |
| 06 hex | Write Single Hold Register | Not valid. | |
| 07 hex | Not used. | Not valid. | |
| 08 hex | Loopback Test | Not valid. | |
| 09 hex | Expanded Read Hold Register Contents | 1 to 508 (1 to 01FC hex) words | 1 to 252 (1 to 00FC hex) words |
| 0A hex | Expanded Read Input Register Contents | 1 to 508 (1 to 01FC hex) words | 1 to 252 (1 to 00FC hex) words |
| 0C hex | Not used. | Not valid. | |
| 0D hex | Expanded Read Nonconsecutive Hold Registers | 1 to 508 (1 to 01FC hex) words | 1 to 252 (1 to 00FC hex) words |
| 0E hex | Expanded Write Nonconsecutive Hold Registers | 1 to 254 (1 to 01FE hex) words | 1 to 126 (1 to 007E hex) words |
| 0F hex | Change Multiple Coil Status | 1 to 800 (1 to 0320 hex) bits | |
| 10 hex | Write Multiple Hold Registers | 1 to 100 (1 to 0064 hex) words | |

### ■ Remote CPU Number (PARAM07)

Specify the remote CPU number. If the remote device is an MP2000-series Machine Controller, set the number to 1. If the remote device is any other Yaskawa Controller that consists of more than one CPU Module, set the CPU number of the send destination.
In all other cases, set the number to 0.

### ■ Coil Offset (PARAM08)

Set the offset to the word address of the coil. This setting is valid for function codes 01, 05, and 0F hex.

### ■ Input Relay Offset (PARAM09)

Set the offset to the word address of the input relay. This setting is valid for function code 02 hex.

### ■ Input Register Offset (PARAM10)

Set the offset to the word address of the input register. This setting is valid for function codes 04 and 0A hex.

### ■ Hold Register Offset (PARAM11)

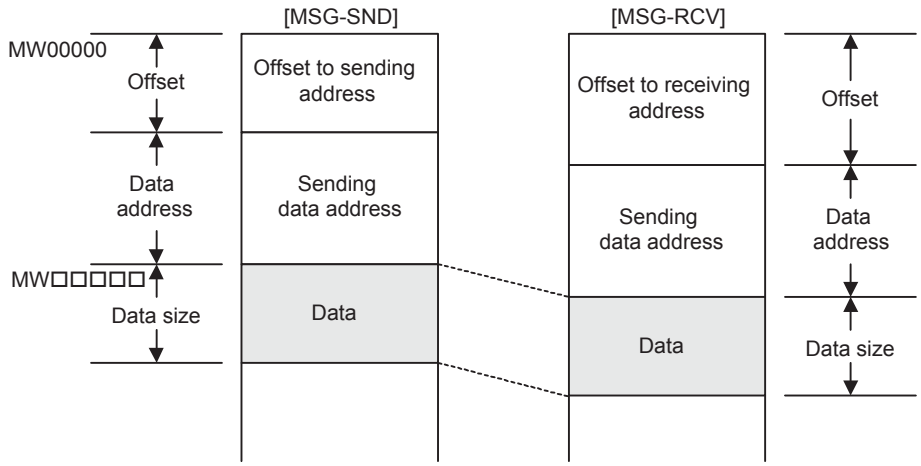
Set the offset to the word address of the hold register. This setting is valid for function codes 03, 06, 09, 0B, 0D, 0E, and 10 hex.

### ■ Reserved for System (PARAM12)

The channel number that is currently in use is held in this parameter. Set this parameter to 0000 hex from a user program in the first scan after the power supply is turned ON. Do not change the value of the parameter after that. It is used by the system.

### ■ Relationship between Data Addresses, Sizes, and Offsets

The following figure shows the relationship between the data addresses, sizes, and offsets.




### ■ No-protocol Communications

It is not necessary to set PARAM04, PARAM09, and PARAM10. Only MW registers can be sent. PARAM11 is the offset to the word address of the MW registers.

### [ c ] Inputs

#### ■ Execute (Send Execution Command)

The message is sent when this command turns ON.

#### ■ Abort (Send Abort Command)

This command aborts sending the message. It takes priority over Execute (Send Execution Command).

#### ■ Dev-Typ (Communications Device Type)

Specify the communications device type.
CPU Module = 8, 215IF = 1, 217IF = 5, 218IF = 6, 218IF-02 = 16, and SVB-01 = 11

#### ■ Pro-Typ (Communications Protocol)

Specify the communications protocol. For no-protocol communications, a response is not received from the remote device.
MEMOBUS: Set this input to 1.
No-protocol: Set this input to 2.

#### ■ Cir-No (Circuit Number)

Specify the circuit number.
CPU Module = 1 or 2, 215IF = 1 to 8, 217IF = 1 to 24, 218IF = 1 to 8, and SVB-01 = 1 to 16

#### ■ Ch-No (Channel Number)

Specify the channel number of the communications section. Do not set the same channel number more than once for the same circuit.
CPU Module = 1, 215IF = 1 to 13, 217IF = 1, 218IF = 1 to 10, and SVB-01 = 1 to 8

#### ■ PARAM (First Address of Parameter List)

Set the first address of the parameter list. For details on this parameter, refer to *[ a ] Parameter Details* earlier in this section.

### [ d ] Output

#### ■ Busy (Processing)

This item indicates that processing is in progress. Keep Execute ON while Busy is ON.

#### ■ Complete (Processing Completed)

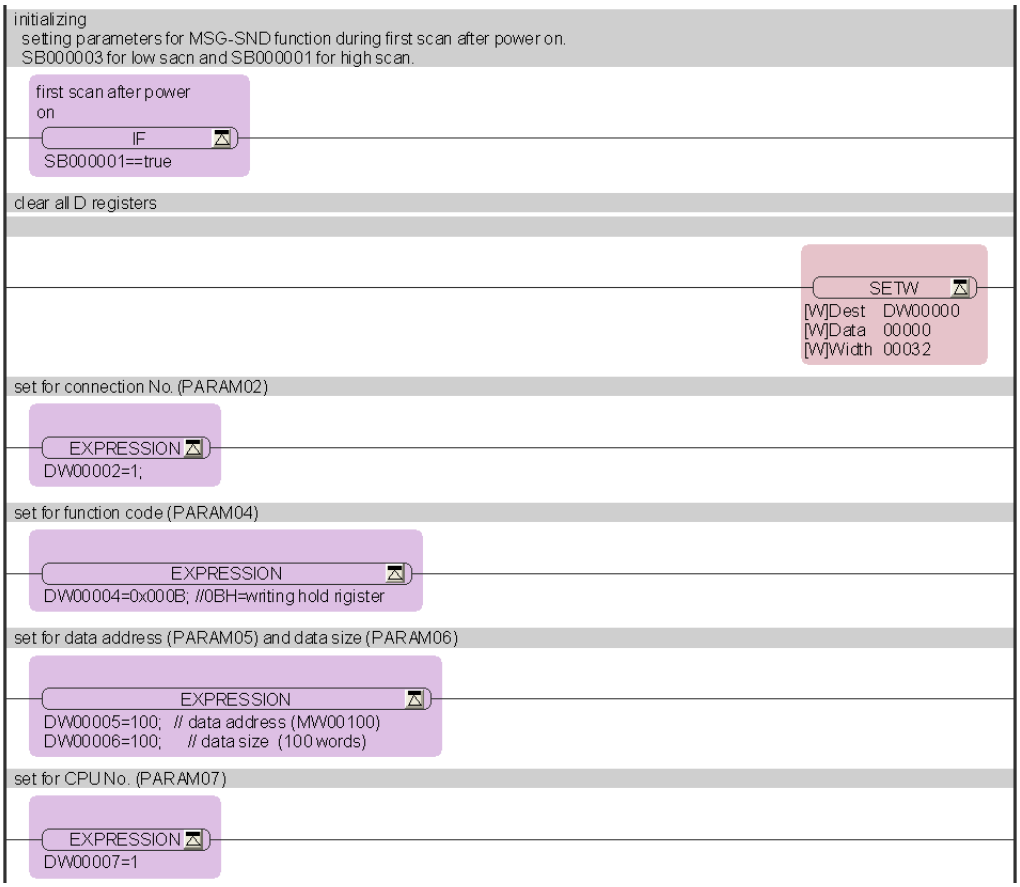
This item turns ON for only one scan when processing is completed normally.

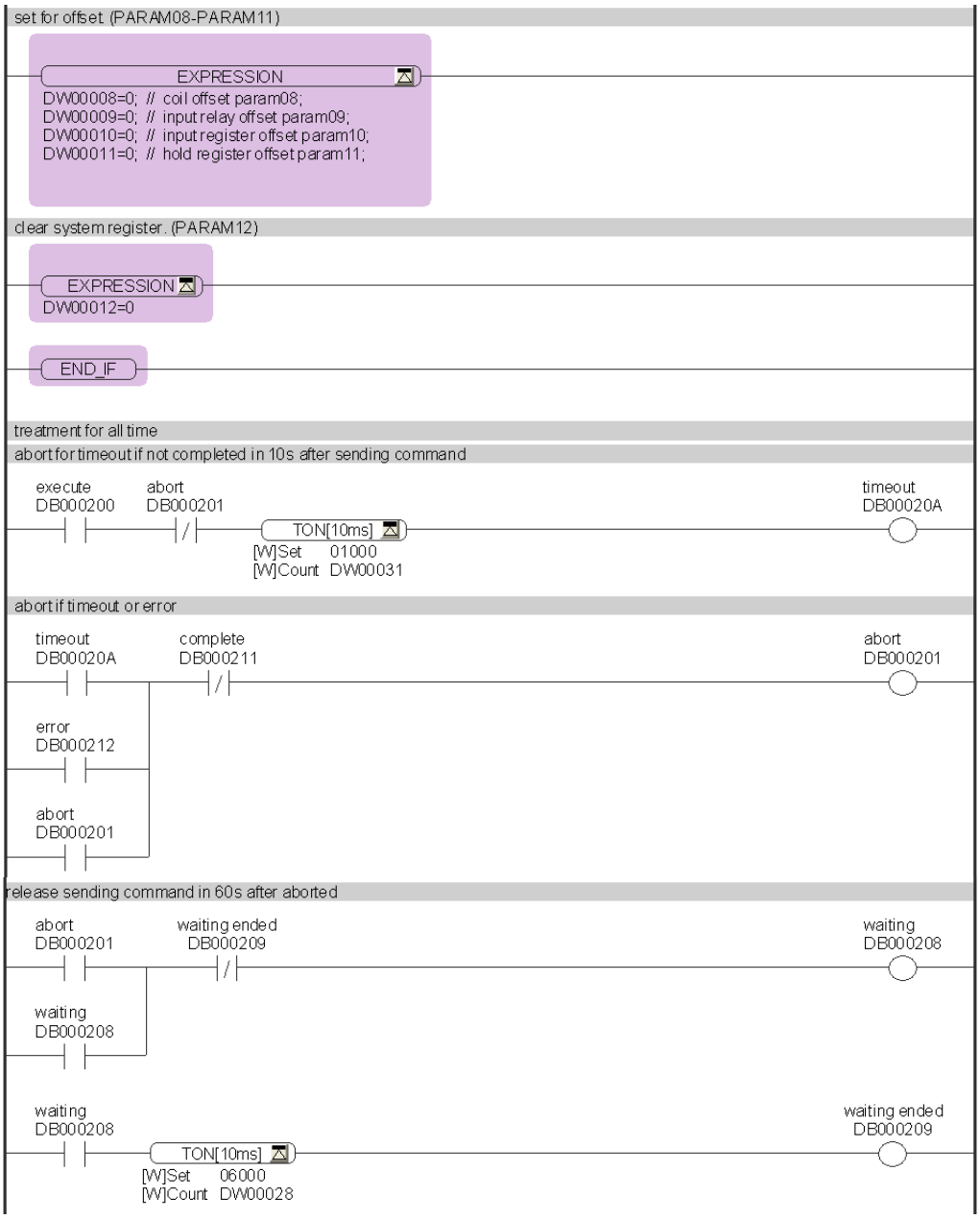
#### ■ Error

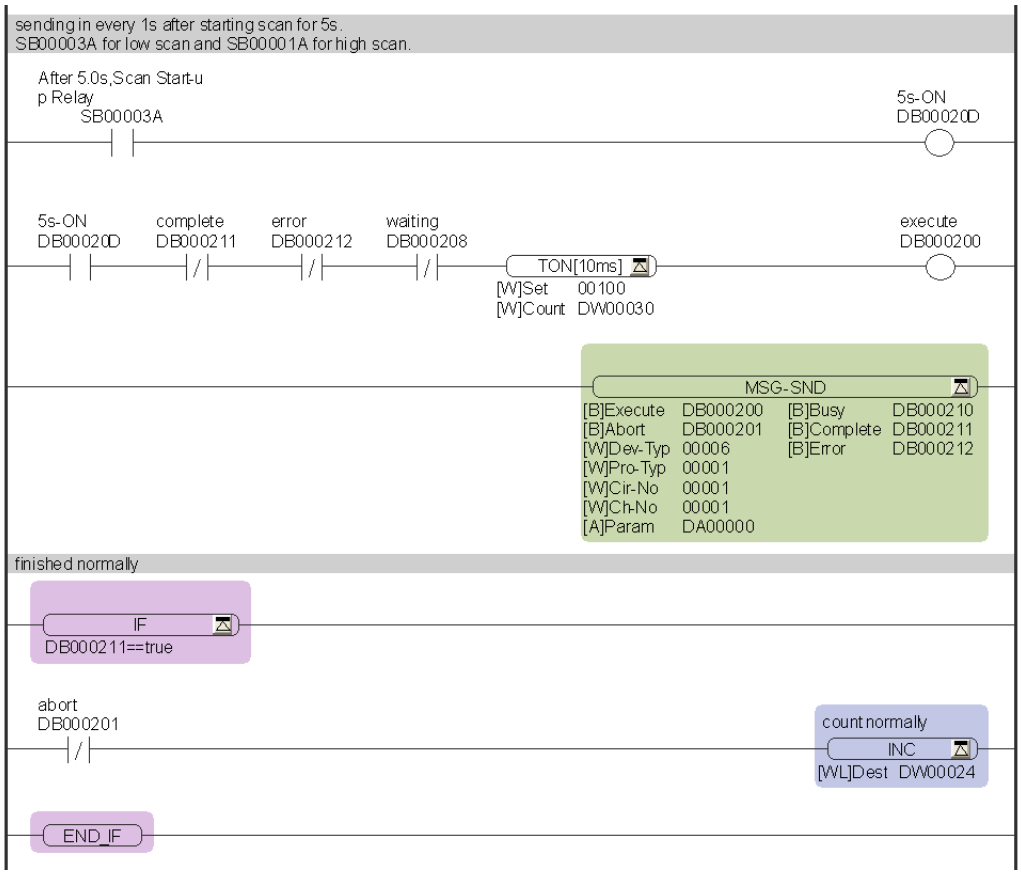
This item turns ON for only one scan when an error occurs. For the causes of the error, refer to information on PARAM00 and PARAM01 in *[ a ] Parameter Details* earlier in this section.

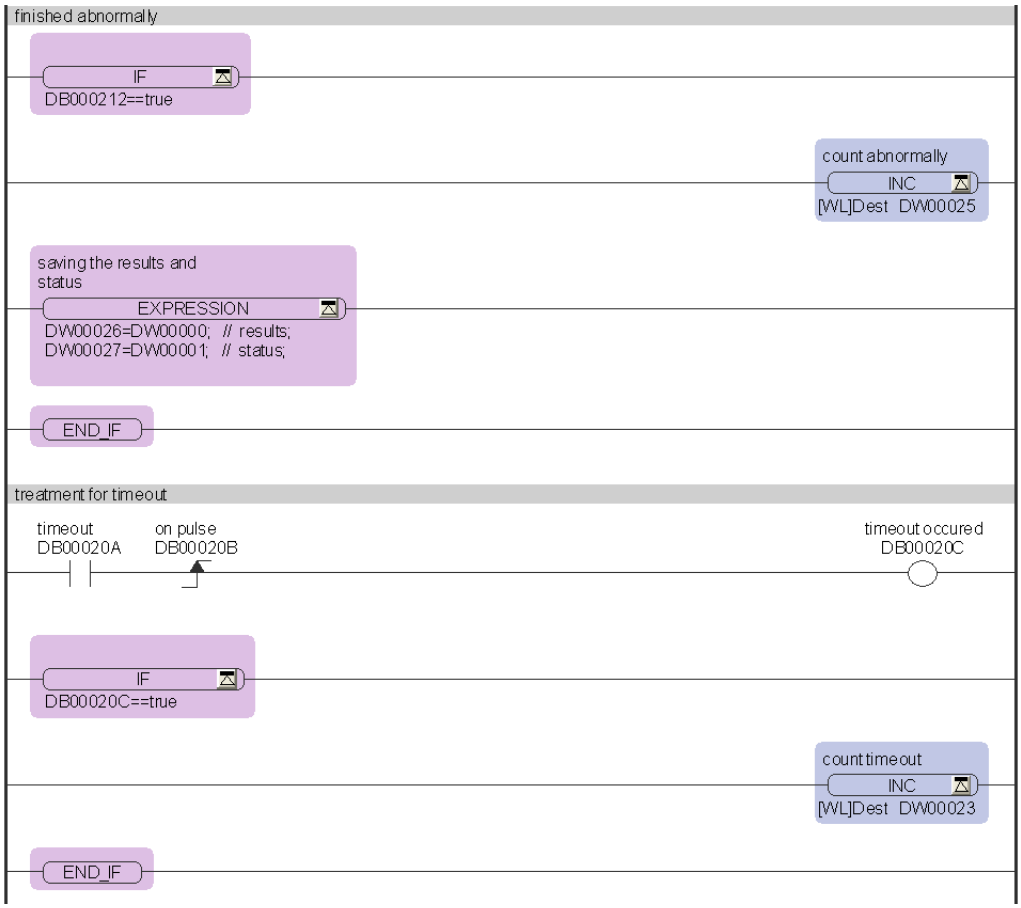
## ( 3 ) Programming Example

In the following programming example, sending data is started according to the set parameters 6.0 seconds after the power supply is turned ON.




set for offset. (PARAM08-PARAM11)
EXPRESSION
DW00008=0; // coil offset param08;
DW00009=0; // input relay offset param09;
DW00010=0; // input register offset param10;
DW00011=0; // hold register offset param11;
clear system register. (PARAM12)
EXPRESSION
DW00012=0
END_IF
treatment for all time
abort for timeout if not completed in 10s after sending command
execute
DB000200
abort
DB000201
TON[10ms]
[W]Set 01000
[W]Count DW00031
timeout
DB00020A
abort if timeout or error
timeout
DB00020A
complete
DB000211
abort
DB000201
error
DB000212
abort
DB000201
release sending command in 60s after aborted
abort
DB000201
waiting ended
DB000209
waiting
DB000208
waiting
DB000208
waiting
DB000208
TON[10ms]
[W]Set 06000
[W]Count DW00028
waiting ended
DB000209


sending in every 1s after starting scan for 5s.
SB00003A for low scan and SB00001A for high scan.
After 5.0s,Scan Start-up Relay
SB00003A
5s-ON
DB00020D
5s-ON
DB00020D
complete
DB000211
error
DB000212
waiting
DB000208
TON[10ms]
[W]Set 00100
[W]Count DW00030
execute
DB000200
MSG-SND
[B]Execute DB000200
[B]Abort DB000201
[W]Dev-Typ 00006
[W]Pro-Typ 00001
[W]Cir-No 00001
[W]Ch-No 00001
[A]Param DA00000
[B]Busy DB000210
[B]Complete DB000211
[B]Error DB000212
finished normally
IF
DB000211==true
abort
DB000201
count normally
INC
[WL]Dest DW00024
END_IF




INFO

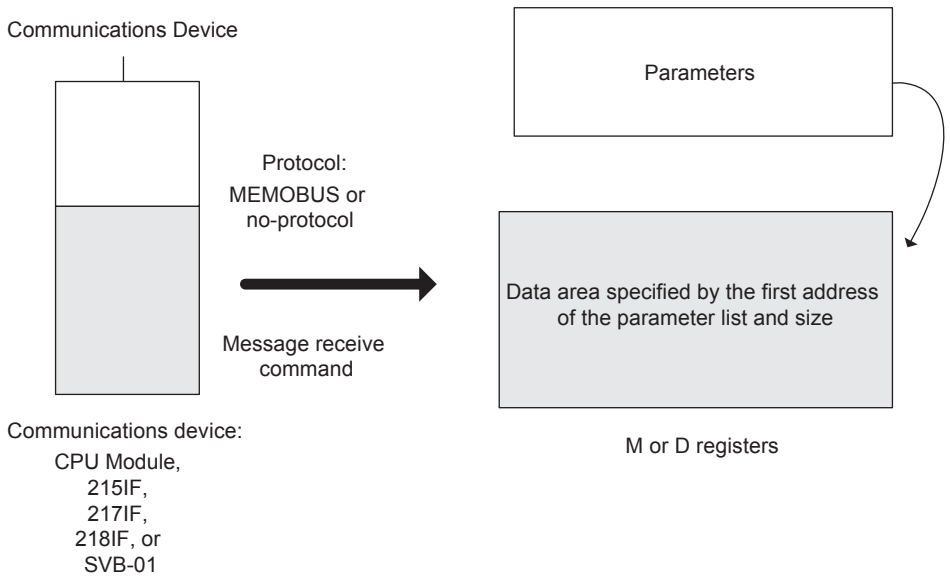
Refer to *Chapter 6 Built-in Ethernet Communications* in the *Machine Controller MP2300S Basic Module User's Manual* (Manual No.: SIEP C880732 00) for application examples of message functions.

# 5.10.7 Receive Message (MSG-RCV)

## ( 1 ) Operation

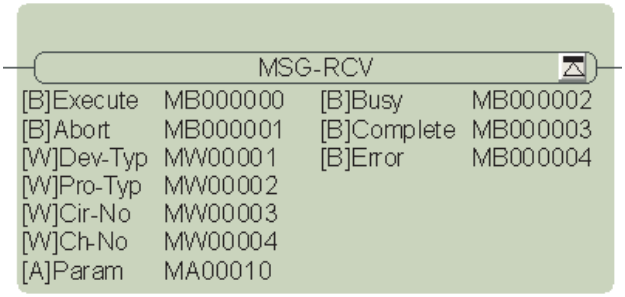
A message is received from a remote station on the specified circuit of the communications device type.
Keep the message receive command ON until the Complete bit turns ON.
This instruction supports the following communications devices and protocols.

Communications devices: CPU Module, 215IF, 217IF, 218IF, and SVB-01
Protocol: MEMOBUS communications or no-protocol




- The Complete bit changes to 1 (ON) when the message reception is completed.
  Until then, keep the receive message command ON.

## ( 2 ) Format




Icon: MSG RCV

Key entry: MSG-RCV

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Receive execution command (Execute) | ○ | × | × | × | × | × | × |
| Receive abort command (Abort) | ○ | × | × | × | × | × | × |
| Communications device type (Dev-Typ) | × | ○ | × | × | × | ○ | ○ |
| Communications protocol (Pro-Typ) | × | ○ | × | × | × | ○ | ○ |
| Circuit number (Cir-No) | × | ○ | × | × | × | ○ | ○ |
| Communications buffer channel number (Ch-No) | × | ○ | × | × | × | ○ | ○ |
| First address of parameter list (Param) | × | × | × | × | ○*1 | × | × |
| Busy | ○*2 | × | × | × | × | × | × |
| Complete | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Receive execution command (Execute) | The message is received when this command turns ON. | IN |
| Receive abort command (Abort) | Receiving the message is aborted when this command turns ON. | IN |
| Communications device type (Dev-Typ) | CPU Module = 8, 215IF = 1, 217IF = 5, 218IF = 6, 218IF-02 = 16, and SVB-01 = 10 | IN |
| Communications protocol (Pro-Typ) | MEMOBUS = 1, No-protocol = 2 | – |
| Circuit number (Cir-No) | CPU Module = 1 or 2, 215IF = 1 to 8, 217IF = 1 to 24, 218IF(-02) = 1 to 8, and SVB-01 = 1 to 16 | IN |
| Communications buffer channel number (Ch-No) | CPU Module = 1 or 2, 215IF = 1 to 13, 217IF = 1, 218IF(-02) = 1 to 10, and SVB-01 = 1 to 8 | IN |
| First address of parameter list (Param) | First address of parameter list (MA, DA, or #A) | IN |
| Busy | Turns ON while receiving the message is in progress. | OUT |
| Complete | Turns ON when receiving the message is completed. | OUT |
| Error | Turns ON when an error occurs. | OUT |

### [ a ] Parameter Details

This section describes the parameters in detail. The parameter number corresponds to the word offset from the first address of the parameter list.

For example, if the first address of the parameter list is MA00100, set the value in MW00110 to set PARAM10.

| Parameter No. | IN/OUT | Description | |
|---|---|---|---|
| | | MEMOBUS | No-protocol |
| PARAM00 | OUT | Processing result | Processing result |
| PARAM01 | OUT | Status | Status |
| PARAM02 | OUT | Remote station number | Remote station number |
| PARAM03 | SYS | Reserved for system. | Reserved for system. |
| PARAM04 | OUT | Function code | – |
| PARAM05 | OUT | Data address | Data address |
| PARAM06 | OUT | Data size | Data size |
| PARAM07 | OUT | Remote CPU number | Remote CPU number |
| PARAM08 | IN | Coil offset | – |
| PARAM09 | IN | Input relay offset | – |
| PARAM10 | IN | Input register offset | – |
| PARAM11 | IN | Hold register offset | Register offset |
| PARAM12 | IN | Writing range low | Register offset |
| PARAM13 | IN | Writing range high | Register offset |
| PARAM14 | SYS | Reserved for system. | Reserved for system. |
| PARAM15 | SYS | Reserved for system. | Reserved for system. |
| PARAM16 | SYS | Reserved for system. | Reserved for system. |

#### ■ Processing Result (PARAM00)

This parameter outputs the result of processing to the upper byte. The lower byte is used for system analysis.

- 00□□ hex: Processing (Busy)
- 10□□ hex: Processing completed (Complete)
- 8□□□ hex: Error

  The following errors can occur.

  - 81□□ hex: Function code error
    An unused function code was received.
  - 82□□ hex: Address setting error
    The data address, coil offset, input relay offset, input register offset, or hold register offset is set outside of the valid range.
  - 83□□ hex: Data size error
    The size of the send or receive data was outside of the valid range.
  - 84□□ hex: Circuit number setting error
    The set circuit number is outside of the valid range.
  - 85□□ hex: Channel number setting error
    The set channel number is outside of the valid range.
  - 86□□ hex: Station address error
    The set station number is outside of the valid range.
  - 88□□ hex: Communications section error
    The communications section returned an error response.
  - 89□□ hex: Device selection error
    A device that cannot be used was selected.

#### ■ Status (PARAM01)

The status of the communications section is output to this parameter. For details, refer to information on the Status parameter (PARAM01) in *5.10.6 Send Message (MSG-SND).*

## ■ Remote Station Number (PARAM02)

The station number of the source is output to this parameter.

## ■ Function Code (PARAM04)

The MEMOBUS function code that was received is output to this parameter.
The MEMOBUS function codes are listed in the following table.

| Function Code | | Setting |
|---|---|---|
| 00 hex | Not used. | × |
| 01 hex | Read Coil Status | ○ |
| 02 hex | Read Input Relay Status | ○ |
| 03 hex | Read Hold Register Contents | ○ |
| 04 hex | Read Input Register Contents | ○ |
| 05 hex | Change Single Coil Status | ○ |
| 06 hex | Write Single Hold Register | ○ |
| 07 hex | Not used. | × |
| 08 hex | Loopback Test | ○ |
| 09 hex | Expanded Read Hold Register Contents | ○ |
| 0A hex | Expanded Read Input Register Contents | ○ |
| 0B hex | Expanded Write Hold Register | ○ |
| 0C hex | Not used. | × |
| 0D hex | Expanded Read Nonconsecutive Hold Registers | ○ |
| 0E hex | Expanded Write Nonconsecutive Hold Registers | ○ |
| 0F hex | Change Multiple Coil Status | ○ |
| 10 hex | Write Multiple Hold Registers | ○ |
| 11 hex to 20 hex | Not used. | × |
| 21 hex to 3F hex | Reserved for system. | × |
| 40 hex to 4F hex | Reserved for system. | × |
| 50 hex and higher | Not used. | × |

- When the target device is operating as a slave, MB, MW, IB, and IW data can be read and written for coils, hold registers, input relays, and input registers, respectively.

## ■ Data Address (PARAM05)

The data address that was requested by the sending node is output to this parameter

## ■ Data Size (PARAM06)

The data size (number of bits or number of words) in the read or write request is output to this parameter.

## ■ Remote CPU Number (PARAM07)

If the remote device is an MP2000-series Machine Controller, 1 is output to this parameter.
If the remote device is any other Yaskawa Controller that consists of more than one CPU Module, the CPU number is output to this parameter.
In all other cases, 0 is output to this parameter.

### ■ Coil Offset (PARAM08)

Set the offset to the word address of the coil.
This setting is valid for function codes 01, 05, and 0F hex.

### ■ Input Relay Offset (PARAM09)

Set the offset to the word address of the input relay.
This setting is valid for function code 02 hex.

### ■ Input Register Offset (PARAM10)

Set the offset to the word address of the input register.
This setting is valid for function codes 04 and 0A hex.

### ■ Hold Register Offset (PARAM11)

Set the offset to the word address of the hold register.
This setting is valid for function codes 03, 06, 09, 0B, 0D, 0E, and 10 hex.

### ■ Writing Range Low (PARAM12) and Writing Range High (PARAM13)

Set the range for which to enable writing for write requests. Any write request that is not within this range will cause an error. This setting is valid for function codes 0B, 0E, 0F, and 10 hex.
0 ≤ Write range low ≤ Write range high ≤ Highest MW address

### ■ Reserved for System (PARAM14)

The channel number that is currently in use is held in this parameter. Set this parameter to 0000 hex from a user program in the first scan after the power supply is turned ON. Do not change the value of the parameter after that. It is used by the system.

### ■ No-protocol Communications

It is not necessary to set PARAM04, PARAM08, PARAM09, PARAM10, and PARAM11.
PARAM12 is also used for the offset to the MW word address at the write destination.

### [ b ] Inputs

#### ■ Execute (Receive Execution Command)

The message is received when this command turns ON.
Keep the Execute bit ON until the Complete or Error bit turns ON.

#### ■ Abort (Receive Abort Command)

This command aborts receiving the message. It takes priority over Execute (Receive Execution Command).

#### ■ Dev-Typ (Communications Device Type)

Specify the communications device type.
CPU Module = 8, 215IF = 1, 217IF = 5, 218IF = 6, 218IF-02 = 16, and SVB-01 = 11

#### ■ Pro-Typ (Communications Protocol)

Specify the communications protocol. For no-protocol communications, a response is not sent to the remote station.
MEMOBUS: Set this input to 1.
No-protocol: Set this input to 2.

#### ■ Cir-No (Circuit Number)

Specify the circuit number.
CPU Module = 1 or 2, 215IF = 1 to 8, 217IF = 1 to 24, 218IF = 1 to 8, and SVB-01 = 1 to 16

#### ■ Ch-No (Channel Number)

Specify the channel number of the communications section. Do not set the same channel number more than once for the same circuit.
CPU Module = 1, 215IF = 1 to 13, 217IF = 1, 218IF = 1 to 10, and SVB-01 = 1 to 8

#### ■ PARAM (First Address of Parameter List)

Set the first address of the parameter list. For details on this parameter, refer to *[ a ] Parameter Details* earlier in this section.

### [ c ] Outputs

#### ■ Busy (Processing)

This item indicates that processing is in progress. Keep Execute ON while Busy is ON.

#### ■ Complete (Processing Completed)

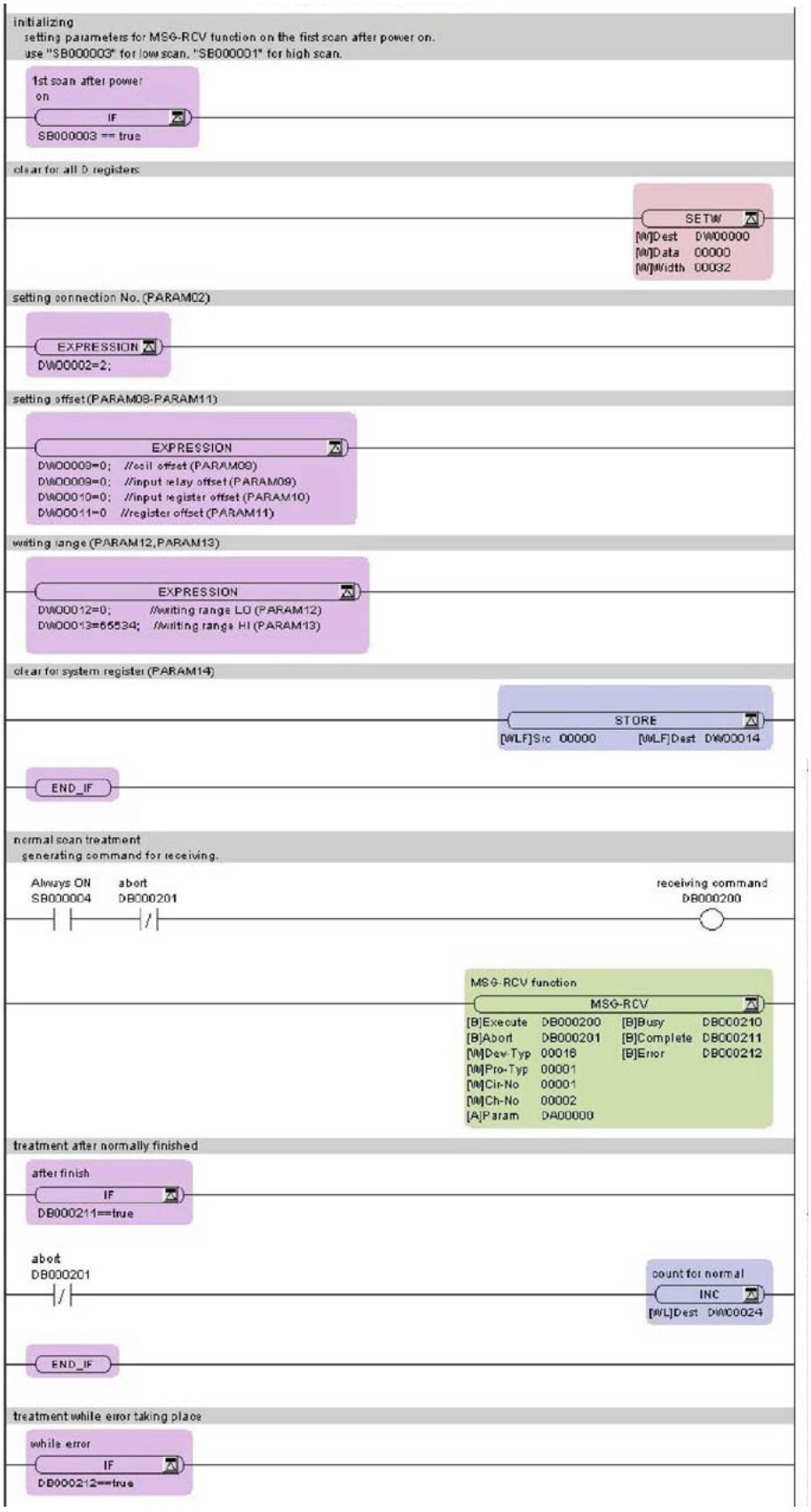
This item turns ON for only one scan when processing is completed normally.

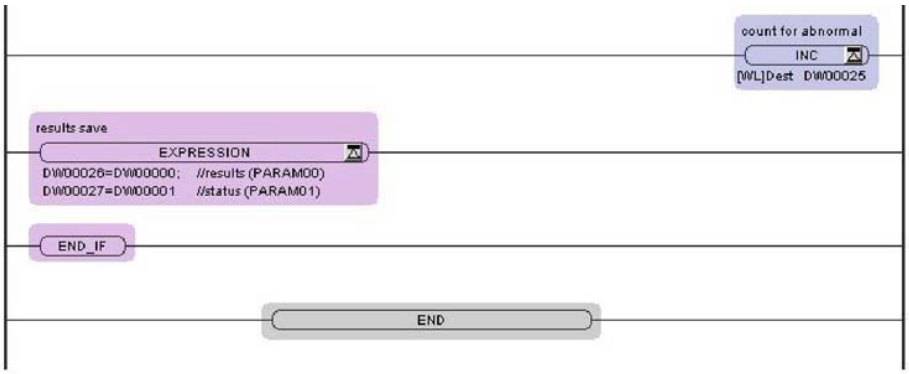
#### ■ Error

This item turns ON for only one scan when an error occurs.
For the causes of the error, refer to information on PARAM00 and PARAM01 in *[ a ] Parameter Details* earlier in this section.

## ( 3 ) Programming Example

In the following programming example, message reception continues with the parameters that are set after the power supply is turned ON.






INFO

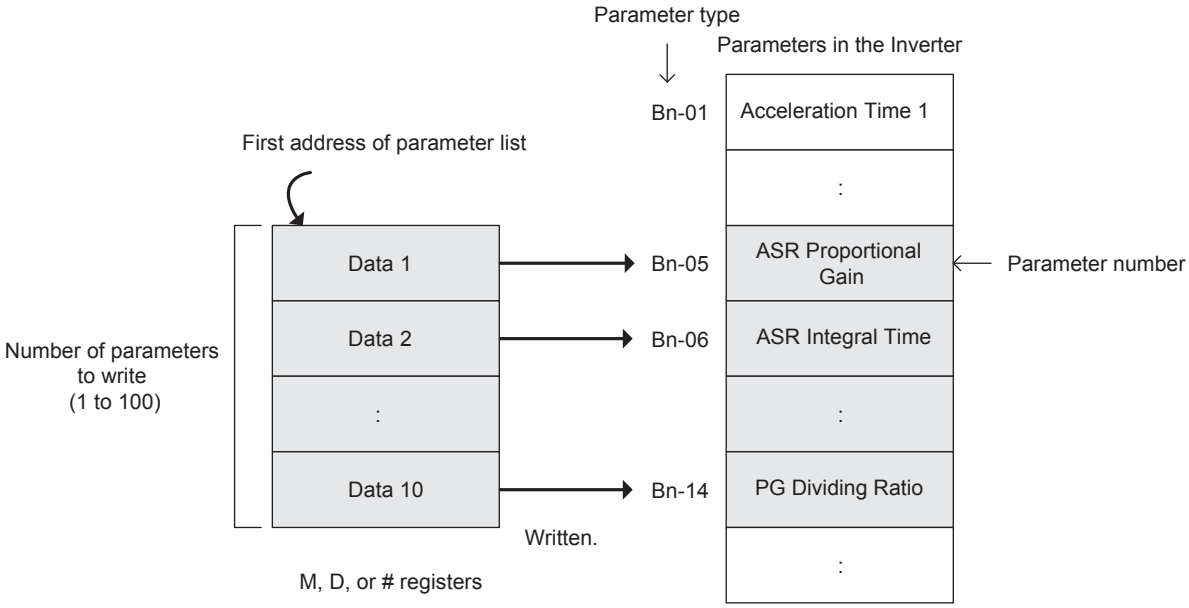
Refer to *Chapter 6 Built-in Ethernet Communications* in the *Machine Controller MP2300S Basic Module User's Manual* (Manual No.: SIEP C880732 00) for application examples of message functions.

## 5.10.8 Write Inverter Parameter (ICNS-WR)

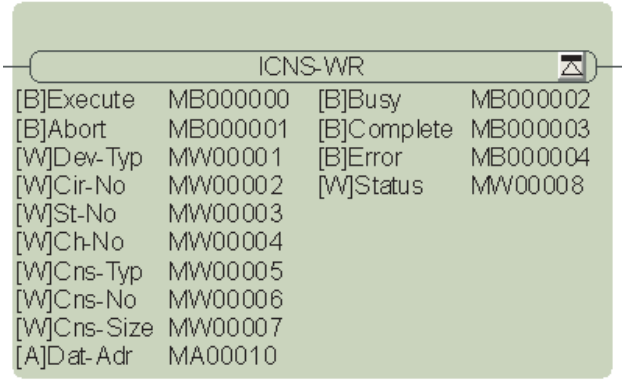
### ( 1 ) Operation

The ICNS-WR instruction writes data to parameters in an Inverter. You can specify the types and range of the parameters in the Inverter.

Applicable Inverters:
This instruction is applicable to Inverters that are connected to the 215IF.



## ( 2 ) Format




Icon: ICNS -WR

Key entry: ICNS-WR

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Write command (Execute) | ○ | × | × | × | × | × | × |
| Write processing abort command (Abort) | ○ | × | × | × | × | × | × |
| Communications device type (Dev-Typ) | × | ○ | × | × | × | ○ | ○ |
| Circuit number (Cir-No) | × | ○ | × | × | × | ○ | ○ |
| Slave station number (St-No) | × | ○ | × | × | × | ○ | ○ |
| Communications buffer channel number (Ch-No) | × | ○ | × | × | × | ○ | ○ |
| Parameter type (Cns-Typ) | × | ○ | × | × | × | ○ | ○ |
| Parameter number (Cns-No) | × | ○ | × | × | × | ○ | ○ |
| Number of parameters to write (Cns-Size) | × | ○ | × | × | × | ○ | ○ |
| First address (Dat-Adr) | × | × | × | × | ○*1 | × | × |
| Writing (Busy) | ○*2 | × | × | × | × | × | × |
| Write completed (Complete) | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |
| Write execution status (Status) | × | ○*2 | × | × | × | ○ | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Write command (Execute) | Writing the Inverter parameters begins when this command turns ON. | IN |
| Write processing abort command (Abort) | The write process is aborted when this command turns ON. | IN |
| Communications device type (Dev-Typ) | 215IF = 1 | IN |
| Circuit number (Cir-No) | 215IF = 1 or 2 | IN |
| Slave station number (St-No) | 215IF = 1 to 64 | IN |
| Communications buffer channel number (Ch-No) | 215IF = 1 to 3 | IN |
| Parameter type (Cns-Typ) | Inverter parameter type:<br>0 = Direct specification of reference number, 1 = An, 2 = Bn, 3 = Cn, 4 = Dn, 5 = En, 6 = Fn, 7 = Hn, 8 = Ln, 9 = On, and 10 = Tn | IN |
| Parameter number (Cns-No) | Inverter parameter number (1 to 99)<br>The upper limit depends on the model of the Inverter.<br>If the parameter type (Cns-Typ) is set to 0, specify the reference number. | IN |
| Number of parameters to write (Cns-Size) | Number of parameters to write to the Inverter (1 to 100) | IN |
| First address (Dat-Adr) | The first register address of the parameters (MA, DA, or #A). | IN |
| Writing (Busy) | Turns ON while writing parameters to the Inverter is in progress. | OUT |
| Write completed (Complete) | Turns ON when writing parameters to the Inverter is completed. | OUT |
| Error | Turns ON when an error occurs. | OUT |
| Write execution status (Status) | Inverter parameter write execution status | OUT |

The status configuration is shown below.

| Bit | Name | Remarks |
|---|---|---|
| 0 to 7 | Reserved for system. | The error code is given here when an Inverter response error is received.<br>01 hex: Function code error<br>02 hex: Reference number error<br>03 hex: Error in the number of data items written<br>21 hex: Error in the upper or lower limit of the write data<br>22 hex: Write error (during operation or under-voltage) |
| 8 | Execution sequence error | The function will not be executed. |
| 9 | Communications parameter error | The function will not be executed. |
| 10 | Specified type error | – |
| 11 | Specified number error | The function will not be executed. |
| 12 | Specified data error | The function will not be executed. |
| 13 | Communications error | The function will not be executed. |
| 14 | Inverter response error | The function will not be executed. |
| 15 | Address input error | The function will not be executed. |

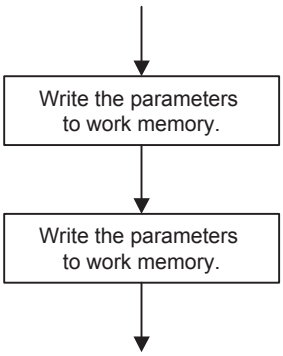
## ( 3 ) Programming Example

In the following programming example, the data in MW00000 is written to parameter Bn-10 in the Inverter that is connected as station 1 to the 215IF on circuit 1.
When DB000000 turns ON, the parameter is written to the Inverter.
After execution, you must check the Complete bit (DB000003) to make sure that it has turned ON, and then write the data to parameter storage memory in the Inverter (EEPROM). (Refer to *( 4 ) Additional Information*.)

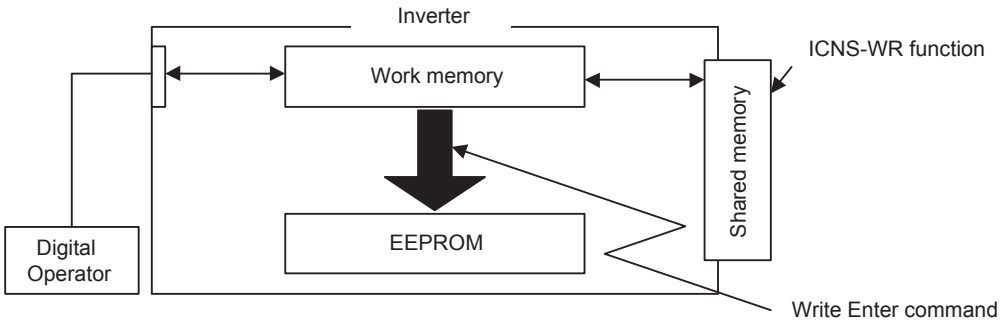
## ( 4 ) Additional Information

### [ a ] Writing Parameters to EEPROM

The procedure to write parameters to the parameter storage memory in the Inverter (EEPROM) is given in the following figure.




The parameters are first written to work memory in the Inverter with the ICNS-WR system function. To write those parameters to EEPROM, you must use the Write Enter command that is shown in the following figure.




### [ b ] Executing the Write Enter Command

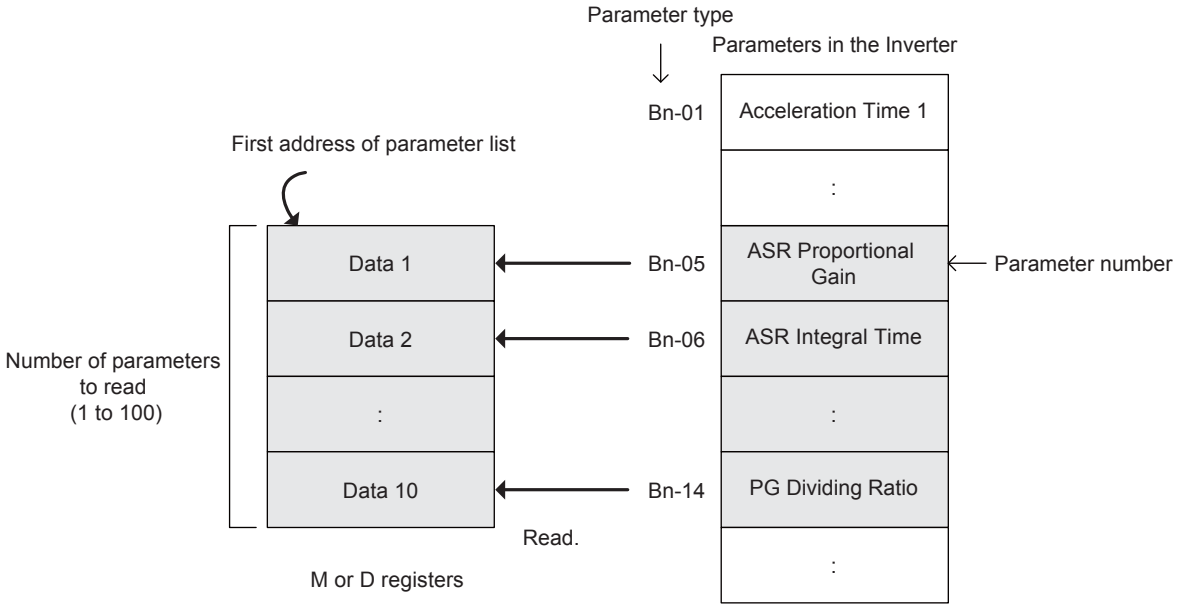
To execute the Write Enter command, use the ICNS-WR function to write data of 0 to the reference “FEED”.

# 5.10.9 Read Inverter Parameter (ICNS-RD)

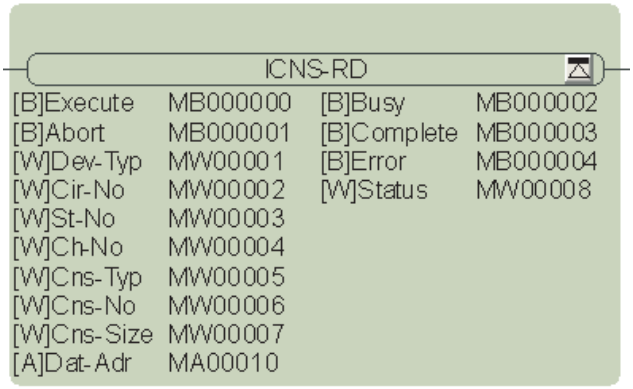
## ( 1 ) Operation

The ICNS-RD instruction reads parameters from an Inverter. You can specify the types and range of the parameters in the Inverter.

Applicable Inverters:
This instruction is applicable to Inverters that are connected to the 215IF.



## ( 2 ) Format




Icon: ICNS -RD

Key entry: ICNS-RD

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Read command (Execute) | ○ | × | × | × | × | × | × |
| Read processing abort command (Abort) | ○ | × | × | × | × | × | × |
| Communications device type (Dev-Typ) | × | ○ | × | × | × | ○ | ○ |
| Circuit number (Cir-No) | × | ○ | × | × | × | ○ | ○ |
| Slave station number (St-No) | × | ○ | × | × | × | ○ | ○ |
| Communications buffer channel number (Ch-No) | × | ○ | × | × | × | ○ | ○ |
| Parameter type (Cns-Typ) | × | ○ | × | × | × | ○ | ○ |
| Parameter number (Cns-No) | × | ○ | × | × | × | ○ | ○ |
| Number of parameters to read (Cns-Size) | × | ○ | × | × | × | ○ | ○ |
| First address (Dat-Adr) | × | × | × | × | ○*1 | × | × |
| Reading (Busy) | ○*2 | × | × | × | × | × | × |
| Read completed (Complete) | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |
| Read execution status (Status) | × | ○*2 | × | × | × | ○ | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

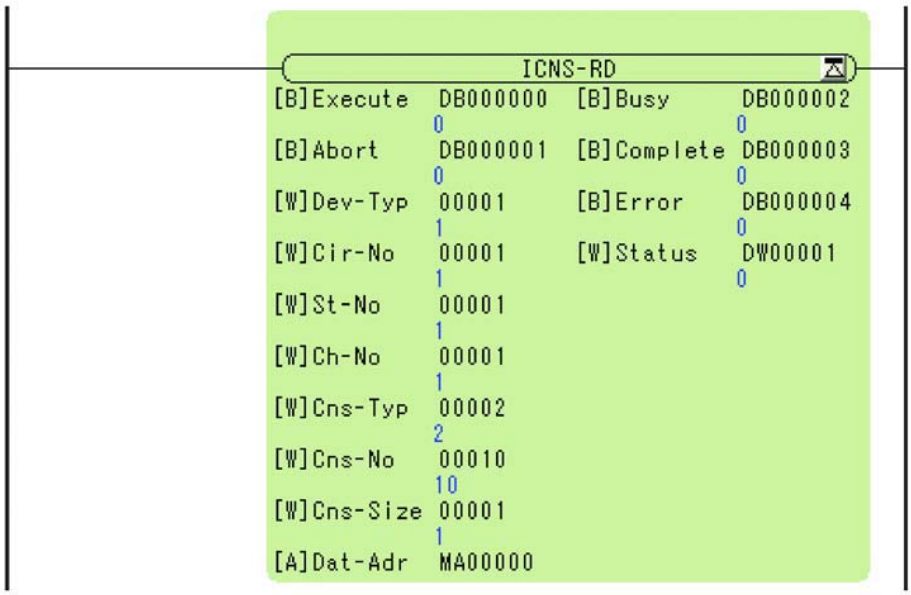
| Parameter Name | Description | I/O |
|---|---|---|
| Read command (Execute) | Reading the Inverter parameters begins when this command turns ON. | IN |
| Read processing abort command (Abort) | The read process is aborted when this command turns ON. | IN |
| Communications device type (Dev-Typ) | 215IF = 1 | IN |
| Circuit number (Cir-No) | 215IF = 1 or 2 | IN |
| Slave station number (St-No) | 215IF = 1 to 64 | IN |
| Communications buffer channel number (Ch-No) | 215IF = 1 to 3 | IN |
| Parameter type (Cns-Typ) | Inverter parameter type:<br>0 = Direct specification of reference number, 1 = An, 2 = Bn, 3 = Cn, 4 = Dn, 5 = En, 6 = Fn, 7 = Hn, 8 = Ln, 9 = On, and 10 = Tn | IN |
| Parameter number (Cns-No) | Inverter parameter number (1 to 99)<br>The upper limit depends on the model of the Inverter.<br>If the parameter type (Cns-Typ) is set to 0, specify the reference number. | IN |
| Number of parameters to read (Cns-Size) | Number of parameters to read from the Inverter (1 to 100) | IN |
| First address (Dat-Adr) | The first register address of the parameters (MA, DA, or #A). | IN |
| Reading (Busy) | Turns ON while reading parameters from the Inverter is in progress. | OUT |
| Read completed (Complete) | Turns ON while reading parameters from the Inverter is completed. | OUT |
| Error | Turns ON when an error occurs. | OUT |
| Read execution status (Status) | Inverter parameter read execution status | OUT |

The status configuration is shown below.

| Bit | Name | Remarks |
|---|---|---|
| 0 to 7 | Reserved for system. | The error code is given here when an Inverter response error is received.<br>01 hex: Function code error<br>02 hex: Reference number error |
| 8 | Execution sequence error | The function will not be executed. |
| 9 | Communications parameter error | The function will not be executed. |
| 10 | Specified type error | – |
| 11 | Specified number error | The function will not be executed. |
| 12 | Specified data error | The function will not be executed. |
| 13 | Communications error | The function will not be executed. |
| 14 | Inverter response error | The function will not be executed. |
| 15 | Address input error | The function will not be executed. |

## ( 3 ) Programming Example

In the following programming example, parameter Bn-10 in the Inverter that is connected as station 1 to the 215IF on circuit 1 is read and the data is stored in MW00000.
When DB000000 turns ON, the parameter is read from the Inverter.



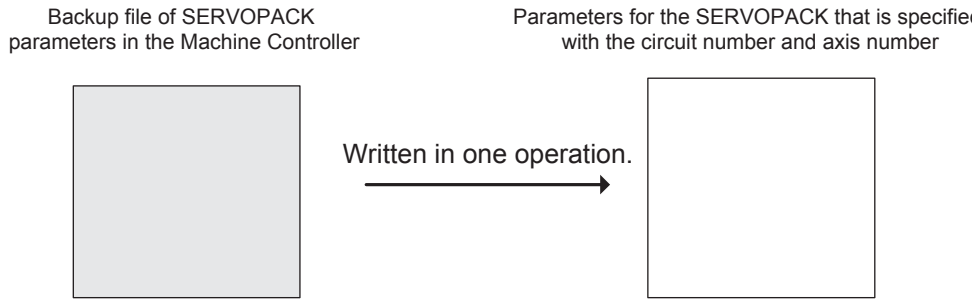
# 5.10.10 Write SERVOPACK Parameter (MLNK-SVW)

## ( 1 ) Operation

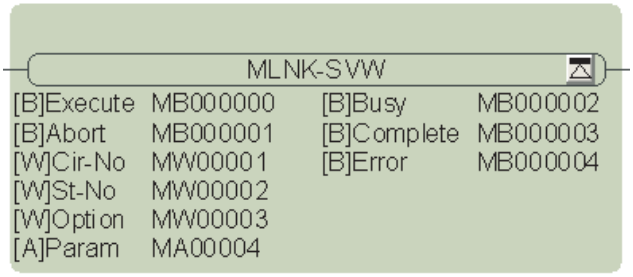
The MLNK-SVW instruction writes all the parameters that are saved in the Machine Controller as a SERVOPACK parameter backup file to the SERVOPACK that is specified with the circuit number and axis number.
This instruction can be used to write SERVOPACK parameters using only a ladder program (i.e., without the use of MPE720 or other tools) when a SERVOPACK is replaced.

**IMPORTANT**

An MP2000-series Machine Controller with software version 2.81 or higher is required to execute the MLNK-SVW instruction for a SERVOPACK connected to an SVC Module.




## ( 2 ) Format




Icon: MLNK -SVW

Key entry: MLNK-SVW

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Write command (Execute) | ○ | × | × | × | × | × | × |
| Write processing abort command (Abort) | ○ | × | × | × | × | × | × |
| Circuit number (Cir-No) | × | ○ | × | × | × | × | × |
| Axis number (St-No) | × | ○ | × | × | × | × | × |
| Option settings (Option) | × | ○ | × | × | × | × | × |
| First address (Param) | × | × | × | × | ○*1 | × | × |
| Writing (Busy) | ○*2 | × | × | × | × | × | × |
| Write completed (Complete) | ○*2 | × | × | × | × | × | × |
| Error | ○*2 | × | × | × | × | × | × |

∗ 1. M or D register only.
∗ 2. C and # registers cannot be used.

The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Write command (Execute) | Writing the SERVOPACK parameters begins when this command turns ON. | IN |
| Write processing abort command (Abort) | The write process is aborted when this command turns ON. | IN |
| Circuit number (Cir-No) | Destination circuit number (1 to 16) | IN |
| Axis number (St-No) | Destination axis number (1 to 16) | IN |
| Option settings (Option) | Command Option Bit Settings<br>Bit E: ID Check Enable/Disable; 0: Enable, 1: Disable<br>Bit F: Version Check Enable/Disable; 0: Enable, 1: Disable<br>The other bits are not used. Any settings in the other bits are ignored. | IN |
| First address (Param) | First address of function workspace | IN |
| Writing (Busy) | Turns ON while writing the parameters is in progress. | OUT |
| Write completed (Complete) | Turns ON for one scan only after the parameters are written. | OUT |
| Error | Turns ON for one scan only when an error occurs.<br>(The error details are output to PARAM00 and PARAM01.) | OUT |

The option settings are described in the following table.

| Bit | Description |
|---|---|
| 0 to D | Not used. (Settings will be ignored.) |
| E | ID Check Enable/Disable (0: Enable, 1: Disable)<br>This option allows you to disable detecting inconsistencies between the ID information in the source file and the ID information where the data is written.<br>Use this option when writing to a stepping motor drive.<br>If this bit is set to 1 (disable), the model information is not checked. This can result in writing parameters for the wrong model, which can cause problems. Take sufficient caution when you disable the check. An inconsistent ID information error will also occur if a SERVOPACK parameters file that was edited or saved offline is used. If that occurs, also disable the ID check. |
| F | Version Check Enable/Disable (0: Enable, 1: Disable)<br>If the version of the source SERVOPACK (communications interface) is not the same as the version at the write destination, an inconsistent version error will occur.<br>SERVOPACK parameters and setting ranges are sometimes different for different versions as the result of changes to specifications. Confirm that no problems will occur before you disable the version check. This will allow you to write the parameters.<br>An inconsistent version error will also occur if a SERVOPACK parameters file that was edited or saved offline is used. If that occurs, also disable the version check. |

### [ a ] Details on Function Workspace

This section provides details on the function workspace. The parameter number corresponds to the word offset from the first address.
For example, if the first address is MA00100, set the value in MW00105 to set PARAM05.

| Parameter No. | IN/OUT | Description |
|---|---|---|
| PARAM00 | OUT | Processing result |
| PARAM01 | OUT | Error code |
| PARAM02 | OUT | Copy of Cir-No |
| PARAM03 | OUT | Copy of St-No |
| PARAM04 | SYS | For system use #1 |
| PARAM05 | SYS | For system use #2 |
| PARAM06 | SYS | For system use #3 |

### ■ Processing Result (PARAM00)

- 0000 hex: Processing (Busy)
- 1000 hex: Processing completed (Complete)
- 8□□□ hex: Error

The following errors can occur.

- 8100 hex: Reserved.
- 8200 hex: Address setting error
  The set data address is outside of the valid range.
- 8300 hex: Reserved.
- 8400 hex: Circuit number setting error
  The set circuit number is outside of the valid range.
- 8500 hex: Reserved.
- 8600 hex: Axis number setting error
  The set axis number is outside of the valid range.
- 8700 hex: Reserved.
- 8800 hex: Communications interface task error
  An error was returned from the communications interface task.
- 8900 hex: Reserved.
- 8A00 hex: Function execution duplication error
  More than one MLNK-SVW function was executed at the same time.

### ■ Error Code (PARAM01)

This parameter outputs the error code from the communications interface task. This parameter is valid only when the processing result (PARAM00) is 8800 hex.

- 0000 hex: Reserved.
- 0001 hex: No SERVOPACK parameter backup file
- 0002 hex: Backup file error
- 0003 hex: Inconsistent ID information
- 0004 hex: Inconsistent version
- 0005 hex: Module error
- 0006 hex: SERVOPACK controller command duplication error
- 0007 hex: Communications error
- 0008 hex: Undefined command
- 0009 hex: Invalid parameter
- 000A hex: Internal system error

### ■ Copy of Cir-No (PARAM02)

This is a copy of the Cir-No input data.

### ■ Copy of St-No (PARAM03)

This is a copy of the St-No input data.

### ■ For System Use #1 (PARAM04)

This parameter is used by the system. Set this parameter to 0000 hex from a user program in the first scan after the power supply is turned ON. Do not modify this parameter at any other time.

### ■ For System Use #2 (PARAM05)

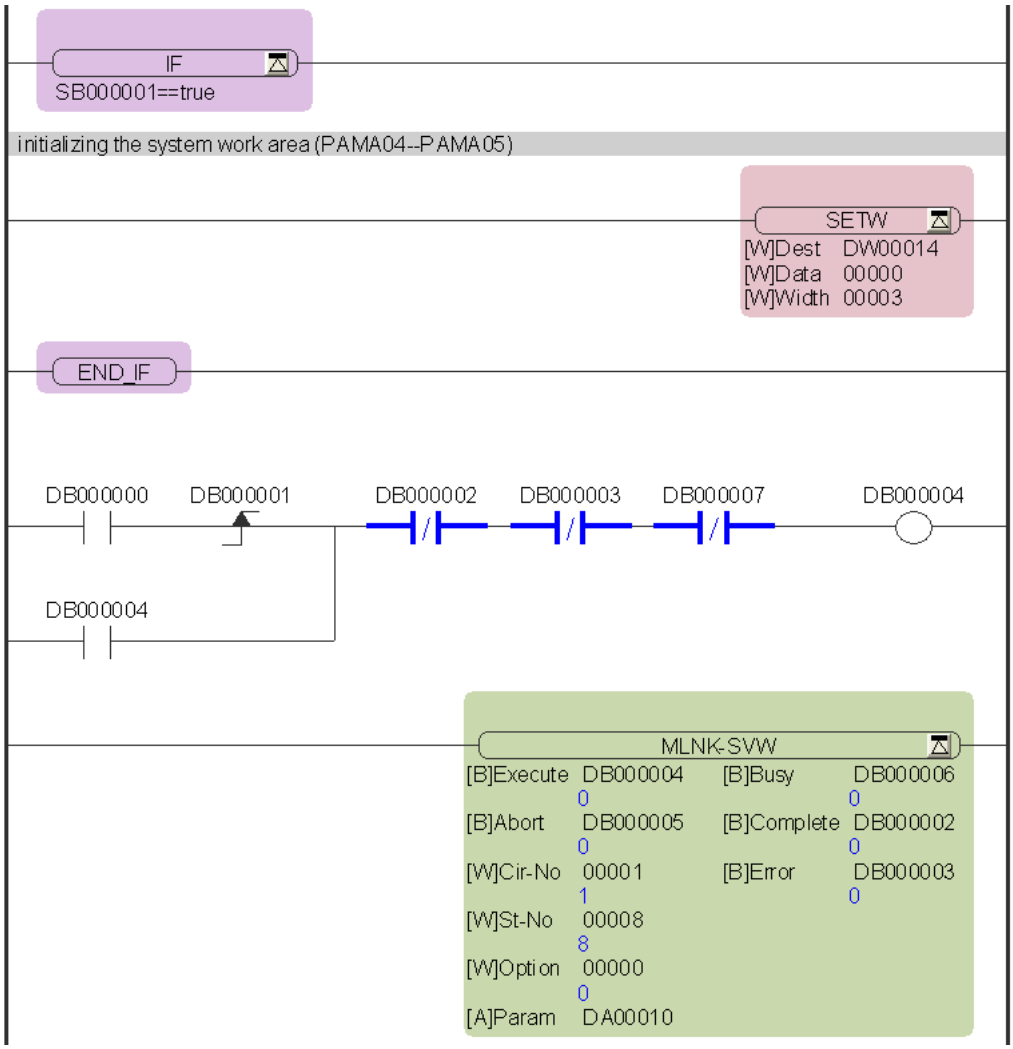
This parameter is used by the system. Set this parameter to 0000 hex from a user program in the first scan after the power supply is turned ON. Do not modify this parameter at any other time.

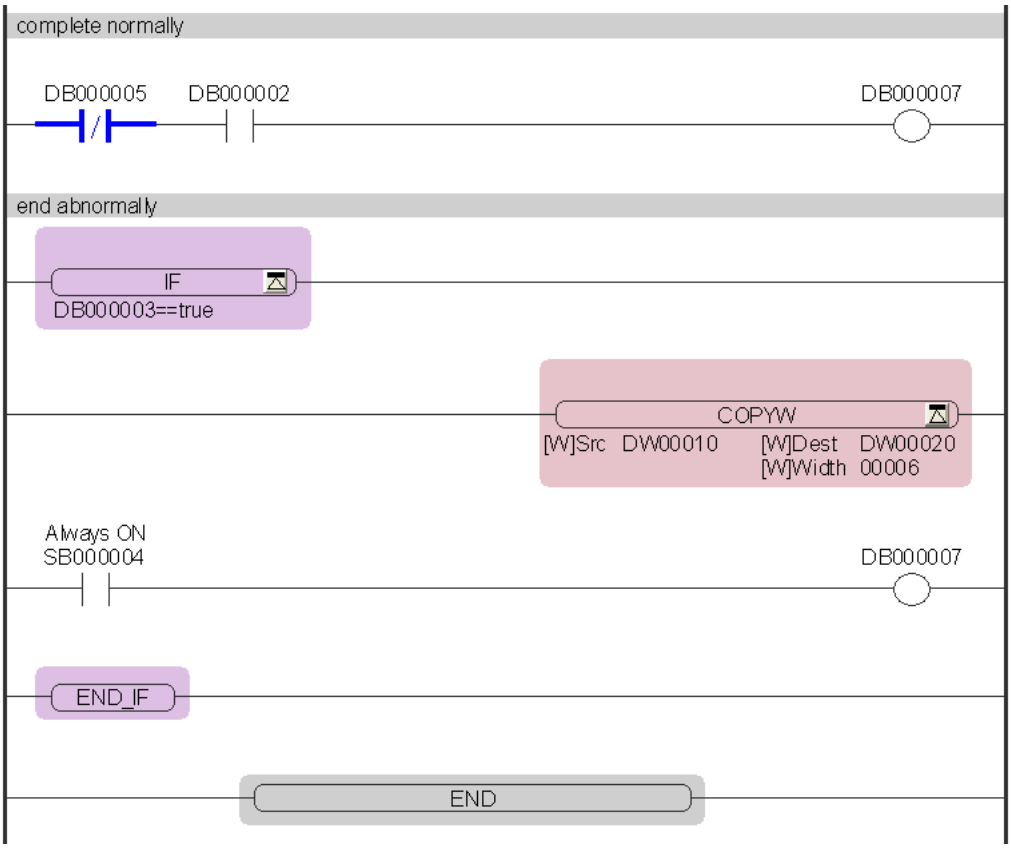
### ■ For System Use #3 (PARAM06)

This parameter is used by the system. Set this parameter to 0000 hex from a user program in the first scan after the power supply is turned ON. Do not modify this parameter at any other time.

## ( 3 ) Programming Example

In the following programming example, the parameters are written to the SERVOPACK.
If a backup file of the SERVOPACK parameters exists in the Machine Controller, the SERVOPACK parameters are written once to the specified SERVOPACK when DB000000 turns ON. The specified SERVOPACK is the one that is defined in the Module configuration definition with a MECHATROLINK circuit number of 1 and defined in the MECHATROLINK detailed definition with ST#8.

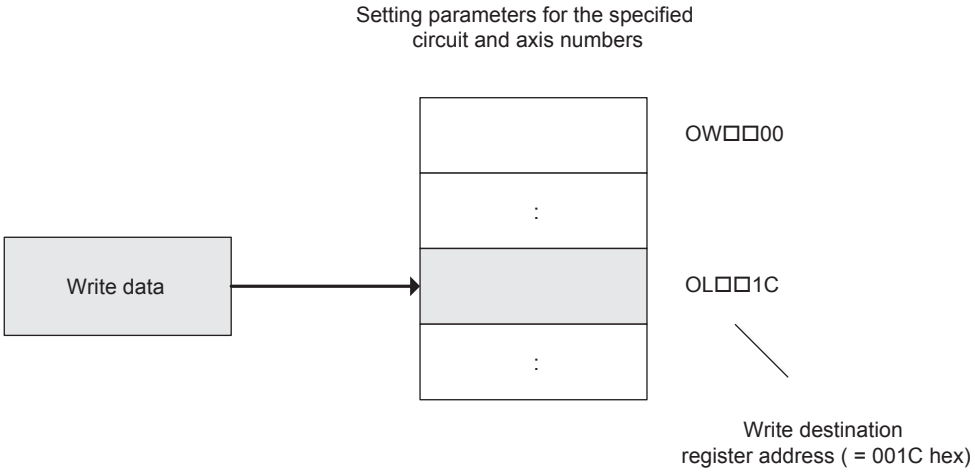



complete normally
DB000005
DB000002
DB000007
end abnormally
IF
DB000003==true
COPYW
[W]Src DW00010
[W]Dest DW00020
[W]Width 00006
Always ON
SB000004
DB000007
END_IF
END

# 5.10.11 Write Motion Register (MOTREG-W)

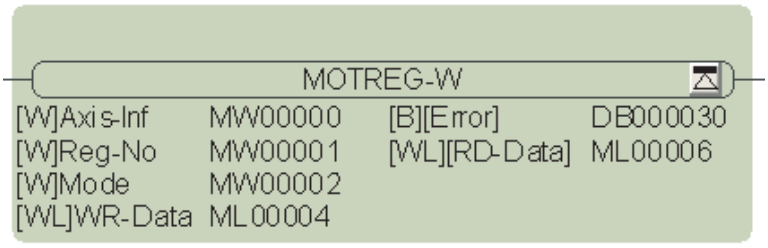
## ( 1 ) Operation

The MOTREG-W instruction calls a function that accesses a specified motion register.
The data is written to the motion register by specifying the circuit number, axis number, and register address. This instruction is used with setting parameters.




- This function is useful to store the same setting parameter for multiple axes with different circuit and axis numbers.
  To store data with the STORE or EXPRESSION instructions, you need to consider an offset to address the circuit and the axis numbers.

## ( 2 ) Format




Icon: MOT REG-W

Key entry: MOTREG-W

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Axis information (Axis-Inf) | × | ○ | × | × | × | × | × |
| Register address (Reg-No) | × | ○ | × | × | × | × | × |
| Mode | × | ○ | × | × | × | × | × |
| Write data (WR-Data) | × | ○ | ○ | × | × | × | × |
| Error | ○* | × | × | × | × | × | × |
| Read data (RD-Data) | × | ○* | ○* | × | × | × | × |

* C and # registers cannot be used. These parameters may be omitted.

The parameters are described in the following table.

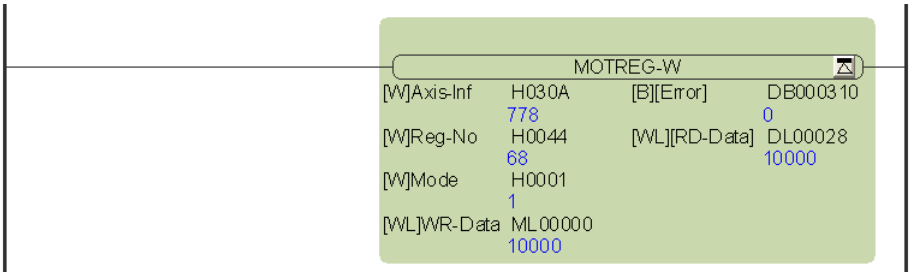
| Parameter Name | Description | I/O |
|---|---|---|
| Axis information (Axis-Inf) | Circuit number and axis number<br>Upper byte: Circuit number (01 to 10 hex)<br>Lower byte: Axis number (01 to 10 hex) | IN |
| Register address (Reg-No) | Integer register: 0000 to 007F hex<br>Double-length integer register: 0000 to 007E hex | IN |
| Mode | Access type and access size<br>Upper byte: Access type<br>0: Write WR-Data to specified register.<br>1: Write inclusive OR of specified register and WR-Data to specified register.<br>2: Write AND of specified register and WR-Data to specified register.<br>Others: Write WR-Data to specified register.<br>Lower byte: Access size<br>0: Integer data<br>1: Double-length integer data<br>Others: Integer data | IN |
| Write data (WR-Data) | If the access size for Mode is an integer and the input data type is a double-length integer, only the lower word will be used. | IN |
| Error | Error cause (Turns ON when an error occurs.)<br>The register cannot be written to or read from because the circuit number, axis number, or register address is outside of the valid range, or because the Module does not exist.<br>When an error occurs, RD-Data is set to 0. | OUT |
| Read data (RD-Data) | This is the data that is read after writing is completed.<br>If integer data is specified, the data is output with the sign. | OUT |

## ( 3 ) Programming Example

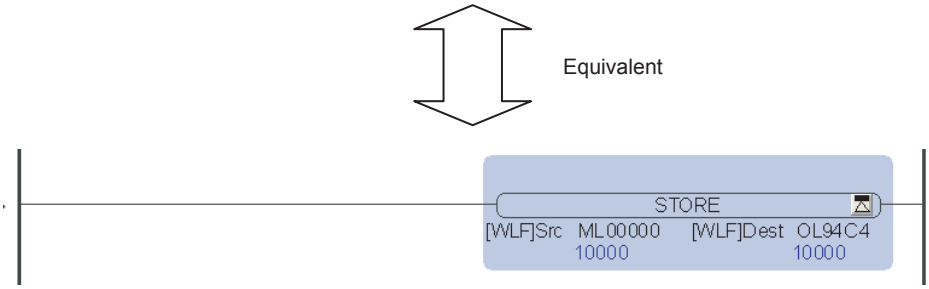
In the following programming example, the value in ML00000 is written to the Step Travel Distance parameter in OW□□44 for axis number 10 on circuit number 3.
Set the following items.

- Axis-Inf = 030A hex (circuit 3, axis 10)
- Register address = 0044 hex
- Mode = 0001 hex (double-length Integer)

For simplicity, this example omits the error and data reading processes.



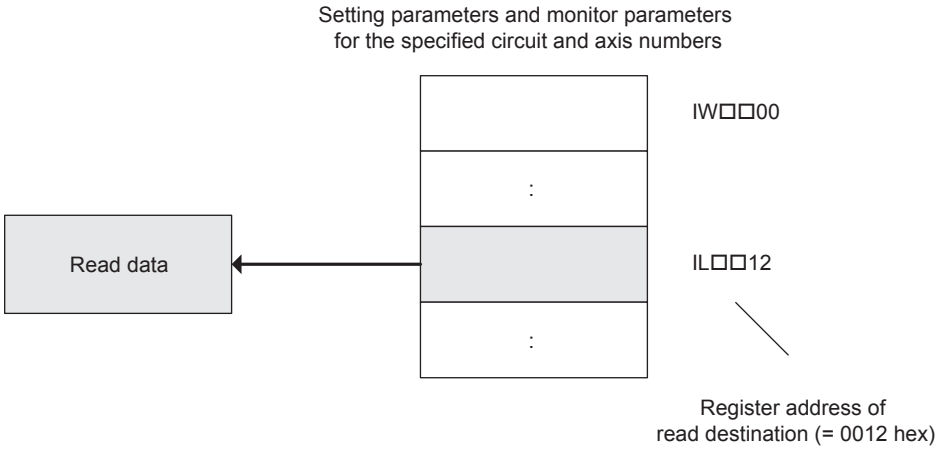

The same result can be achieved by directly specifying the register address and storing data with the STORE instruction.



# 5.10.12 Read Motion Register (MOTREG-R)

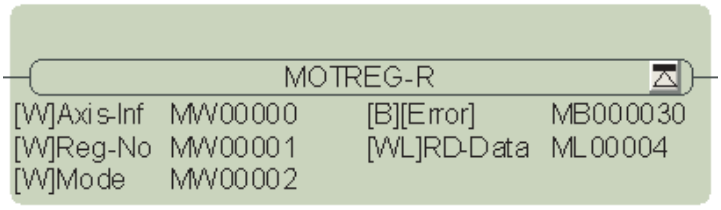
## ( 1 ) Operation

The MOTREG-R instruction calls a function that accesses a specified motion register.
The value is read from the motion register by specifying the circuit number, axis number, and register address.
This instruction is used with setting parameters and monitor parameters.




- This instruction is useful to read the same parameter for multiple axes with different circuit and axis numbers. To read data with the STORE or EXPRESSION instructions, you need to consider an offset to address the circuit and the axis numbers.

## ( 2 ) Format




Icon: MOT REG-R

Key entry: MOTREG-R

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Axis information (Axis-Inf) | × | ○ | × | × | × | × | × |
| Register address (Reg-No) | × | ○ | × | × | × | × | × |
| Mode | × | ○ | × | × | × | × | × |
| Write data (WR-Data) | × | ○ | ○ | × | × | × | × |
| Error | ○* | × | × | × | × | × | × |
| Read data (RD-Data) | × | ○* | ○* | × | × | × | × |

* C and # registers cannot be used. These parameters may be omitted.

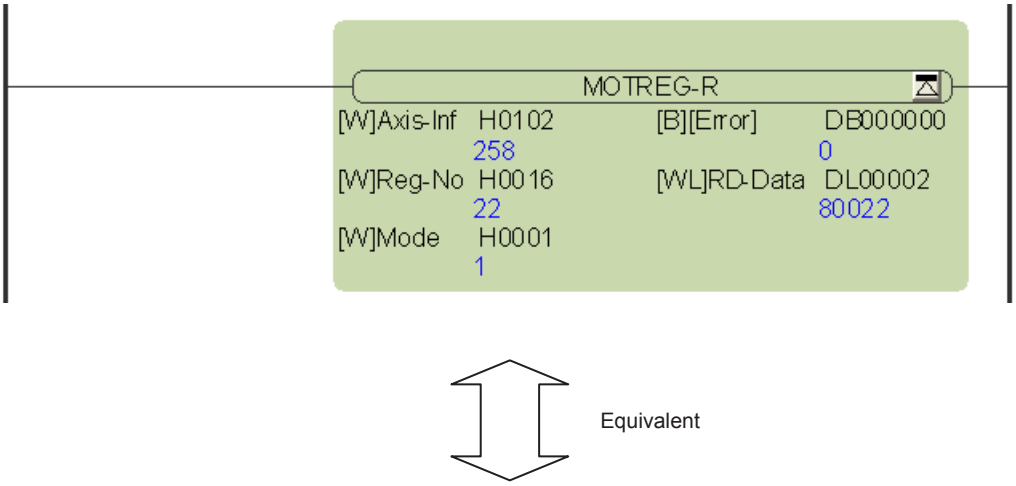
The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Axis information (Axis-Inf) | Circuit number and axis number<br>Upper byte: Circuit number (01 to 10 hex)<br>Lower byte: Axis number (01 to 10 hex) | IN |
| Register address (Reg-No) | Integer register: 0000 to 007F hex<br>Double-length integer register: 0000 to 007E hex | IN |
| Mode | Register type and access size<br>Upper byte: Register type<br>0: I register (monitor parameter)<br>1: O register (setting parameter)<br>Others: I register<br>Lower byte: Access size<br>0: Integer data<br>1: Double-length integer data<br>Others: Integer data | IN |
| Error | Error cause (Turns ON when an error occurs.)<br>The register cannot be written to or read from because the circuit number, axis number, or register address is outside of the valid range, or because the Module does not exist.<br>When an error occurs, RD-Data is set to 0. | OUT |
| Read data (RD-Data) | If integer data is specified, the data is output with the sign. | OUT |

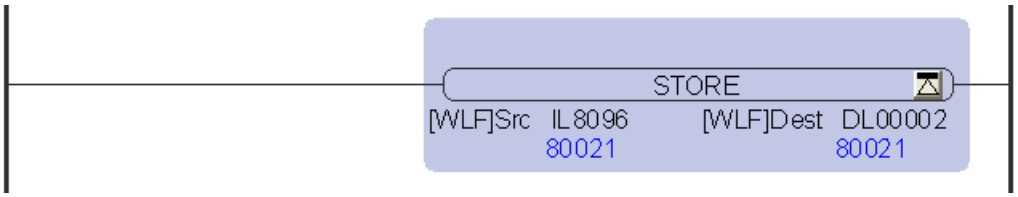
## ( 3 ) Programming Example

In the following programming example, the Machine Coordinate System Feedback Position in IL8096 for axis number 2 on circuit number 1 is read.
Set the following items to read the feedback position and store it in DL00002.

- Axis-Inf = 0102 hex (circuit 1, axis 2)
- Register address = 0016 hex
- Mode = 0001 hex (monitor parameter, double-length integer)




The same result can be achieved by directly specifying the register address and storing data in DL00002 with the STORE instruction.

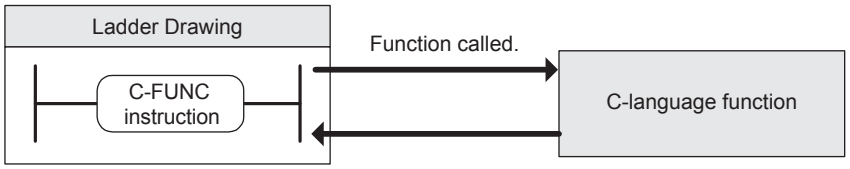


# 5.11 C-language Control Instructions

## 5.11.1 Call C-language Function (C-FUNC)

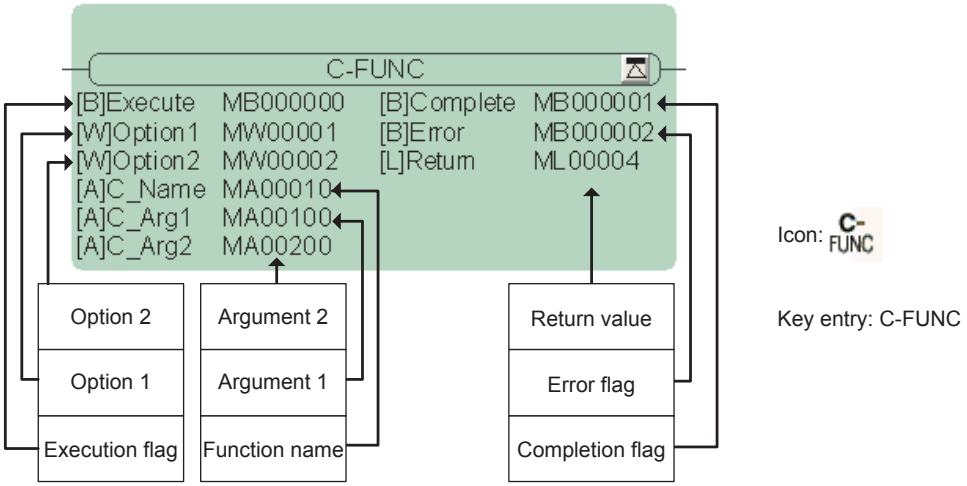
### ( 1 ) Operation

This instruction calls a C-language function from a ladder drawing.




- For details on C-language functions, refer to the *Machine Controller MP2000 Series Embedded C-Language Programming Package Development Guide* (Manual No.: SIEP C880700 25).

### ( 2 ) Format




| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Execution flag (Execute) | ○ | × | × | × | × | × | × |
| Option 1 (Option1) | × | ○ | × | × | × | ○ | ○ |
| Option 2 (Option2) | × | ○ | × | × | × | ○ | ○ |
| Function name (C_Name) | × | × | × | × | ○ | × | × |
| Argument 1 (C_Arg1) | × | × | × | × | ○*3 | × | × |
| Argument 2 (C_Arg2) | × | × | × | × | ○*3 | × | × |
| Completion flag*1 (Complete) | ○*2 | × | × | × | × | × | × |
| Error flag*1 (Error) | ○*2 | × | × | × | × | × | × |
| Return value (Return) | × | × | ○ | × | × | × | × |

∗ 1. Optional.
∗ 2. C and # registers cannot be used.
∗ 3. M or D register only.

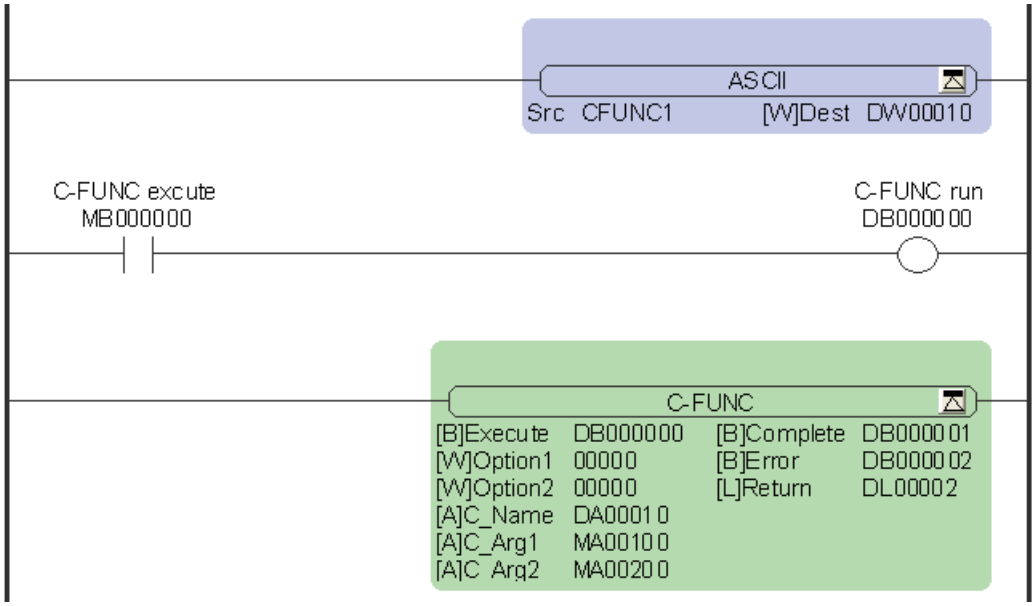
The parameters are described in the following table.

| Parameter Name | Description | I/O |
|---|---|---|
| Execution flag (Execute) | The C-FUNC function is executed when this command turns ON. | IN |
| Option 1 (Option1) | Option specification 1 (for future use) | IN |
| Option 2 (Option2) | Option specification 2 (for future use) | IN |
| Function name (C_Name) | Specify the first register address to pass to argument 1 of the user C-language function. | IN |
| Argument 1 (C_Arg1) | Specify the first register address to pass to argument 1 of the user C-language function. | IN |
| Argument 2 (C_Arg2) | Specify the first register address to pass to argument 2 of the user C-language function. | IN |
| Completion flag (Complete) | Turns ON when execution of the C-FUNC function is completed. | OUT |
| Error flag (Error) | Turns ON when one of the following errors occurs.<br>• Register limit exceeded for C_Name, C_Arg.1, or C_Arg.2.*<br>The sizes of C_Arg.1 and C_Arg.2 are not considered.<br>• The function specified by C_Name does not exist. | OUT |
| Return value (Return) | Stores the value that is returned by the user C-language function. | OUT |

* This error is detected even when Execute is OFF.

## ( 3 ) Programming Example

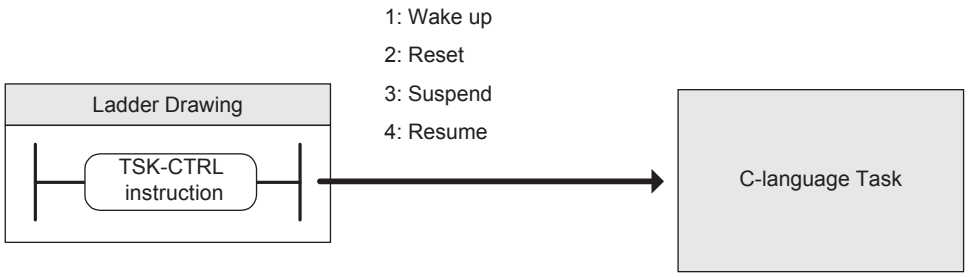
In the following programming example, the CFUNC1 C-language function is executed.
First the CFUNC1 C-language function is loaded into the Controller.
Next, when the C-FUNC execution command (MB000000) turns ON, the CFUNC1 C-language function is executed by the C-FUNC instruction.
The MA00100 and MA00200 addresses are passed to the C-language function as the arguments and the return value from the function is set in DL00002. Options 1 and 2 are not used, so 0 is set for them.



## 5.11.2 C-language Task Control (TSK-CTRL)

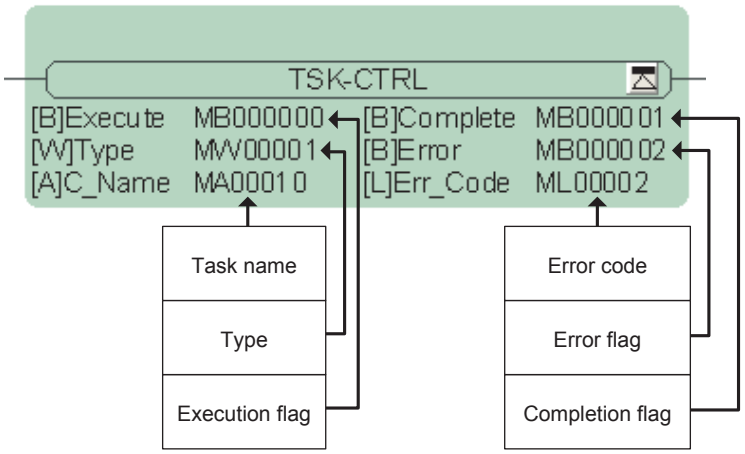
### ( 1 ) Operation

This instruction controls a C-language task from a ladder drawing. The task can be woken up, reset, suspended, or resumed.




- For details on C-language functions, refer to the *Machine Controller MP2000 Series Embedded C-Language Programming Package Development Guide* (Manual No.: SIEP C880700 25).

### ( 2 ) Format




Icon: TSK-CTRL

Key entry: TSK-CTRL

| Parameter Name | Applicable Data Types | | | | | | |
|---|---|---|---|---|---|---|---|
| | B | W | L | F | A | Index | Constant |
| Execution flag (Execute) | ○ | × | × | × | × | × | × |
| Type (Type) | × | ○ | × | × | × | ○ | × |
| Task name (C_Name) | × | × | × | × | ○ | × | × |
| Completion flag (Complete) | ○* | × | × | × | × | × | × |
| Error flag (Error) | ○* | × | × | × | × | × | × |
| Error code (Err_code) | × | × | ○* | × | × | × | × |

* C and # registers cannot be used.

The parameters are described in the following table.

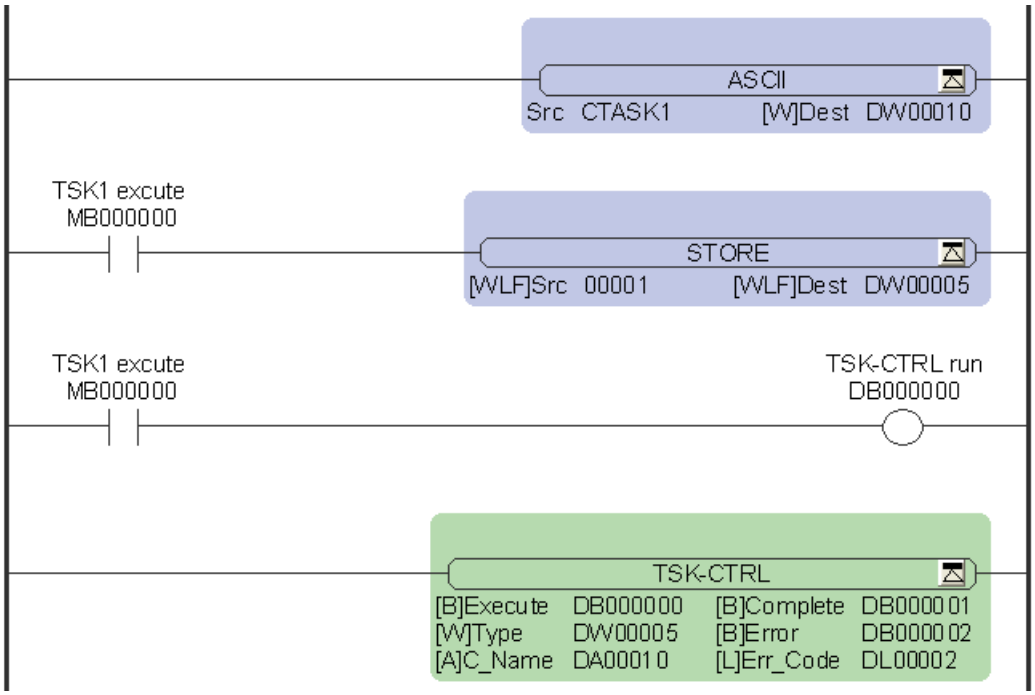
| Parameter Name | Description | I/O |
|---|---|---|
| Execution flag (Execute) | The TSK-CTRL instruction is executed when this command turns ON. | IN |
| Type (Type) | 1: Wake up<br>A task that is in WAIT state is woken up.<br>2: Reset<br>A task is ended, deleted, generated again, and started.<br>The task then executes itself and goes into WAIT state.<br>3: Suspend<br>The task is suspended and placed in SUSPEND state.<br>4: Resume<br>A task in SUSPEND state is changed to READY state. | IN |
| Task name (C_Name) | Specify the first register address of the registers in which the task name of the user C-language task is stored. | IN |
| Completion flag (Complete) | Turns ON when execution of the TSK-CTRL instruction is completed. | OUT |
| Error flag (Error) | Turns ON when an error occurs. (The error is given in Err_code.) | OUT |
| Error code (Err_code) | Error Codes<br>0□00000000<br>No error<br>0□00000091<br>Type setting error<br>• The value of Type is outside of the valid range.<br>• Type was set to Wake up when the state was not WAIT or WAIT-SUSPEND.<br>• Type was set to Suspend when the state was not WAIT or READY.<br>0□00000094<br>The task specified by C_Name does not exist.<br>0□00000096<br>Register upper/lower limit exceeded for C_Name.*1<br>0□FFFFFFDD<br>Error detected by µITRON: Illegal ID number*2<br>0□FFFFFFCC<br>Error detected by µITRON: Task not registered*2<br>0□FFFFFFC1<br>Error detected by µITRON: Illegal object state<br>• The task is in DORMANT state.<br>• Resume was specified when the task was not in SUSPEND state.<br>0□FFFFFFBB<br>Error detected by µITRON: Context error<br>• The task could not be executed in a non-task context.<br>0□FFFFFFB7<br>Error detected by µITRON: Queuing overflow*2 | OUT |

∗ 1. Execute is not treated as a rising edge, but as a level. This allows cyclic execution in the high scan level and low scan level tasks.

∗ 2. The errors that are detected by µITRON normally do not occur because of system management.

## ( 3 ) Programming Example

In the following programming example, the CTASK1 C-language task is executed.
First the CTASK1 C-language task is loaded into the Controller.
Next, when the TSK1 execute command (MB000000) turns ON, the CTASK1 C-language task is executed by the TSK-CTRL instruction.



# 6

# Features of the MPE720 Engineering Tool

This chapter describes the key features of the MPE720 Engineering Tool for ladder programming. Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for details on these features, information on other features, and operating procedures.


6.1 Ladder Program Runtime Monitoring ----------6-2
6.2 Searching/Replacing ----------6-3
6.3 Cross References ----------6-4
6.4 Checking for Multiple Coils ----------6-5
6.5 Forcing Coils ON and OFF ----------6-5
6.6 Viewing Called Programs ----------6-6
6.7 Register Lists ----------6-6
6.8 Tuning Panel ----------6-7
6.9 Enabling and Disabling Ladder Programs ----------6-8
6.10 Compiling for MPE720 Version 5 ----------6-9

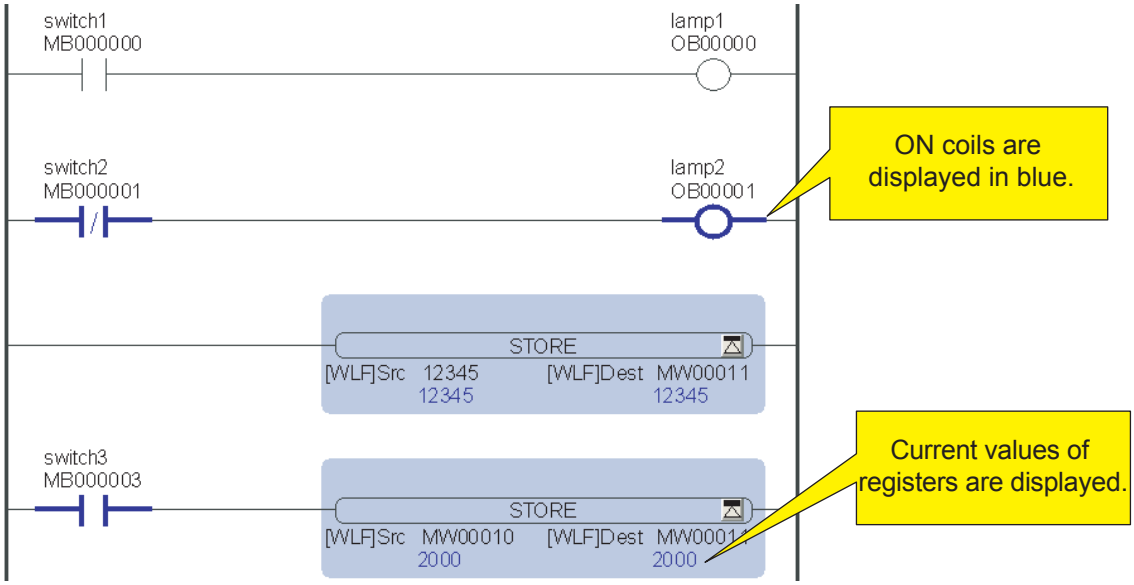
This chapter describes the following ladder programming and debugging features of MPE720 version 6.

- Ladder program runtime monitoring
- Searching/replacing
- Cross references
- Multiple coils
- Forcing coils ON and OFF
- Viewing called programs
- Register lists
- Tuning panel
- Enabling and disabling ladder programs
- Compiling for MPE720 version 5

MPE720 version 6 provides many other features. Refer to the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) for details on these features and information on other features.

# 6.1 Ladder Program Runtime Monitoring

You can monitor the execution status of each instruction. Using runtime monitoring requires a connection to the Machine Controller.
Instructions where the relay output is ON are displayed in blue.
The current values of the parameter registers of the instructions that are being executed are also displayed.

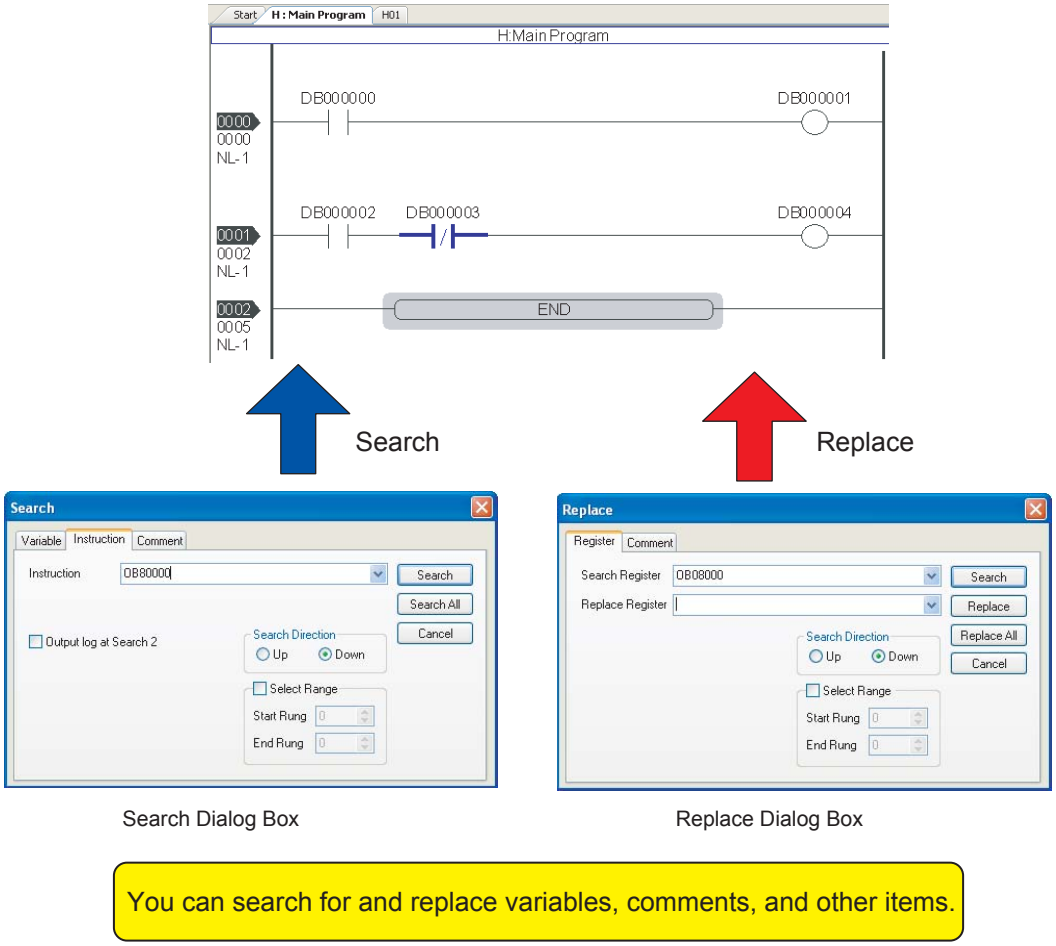


# 6.2 Searching/Replacing

Two different search/replace operations are provided.

- Searching and Replacing in Programs
  You can search for and replace variables, instructions, and comments in the currently active ladder drawing.

- Searching and Replacing in Project Files
  You can search for and replace variables in all ladder drawings in a project file.
  You can use this operation only when the MPE720 is not connected to the Machine Controller.



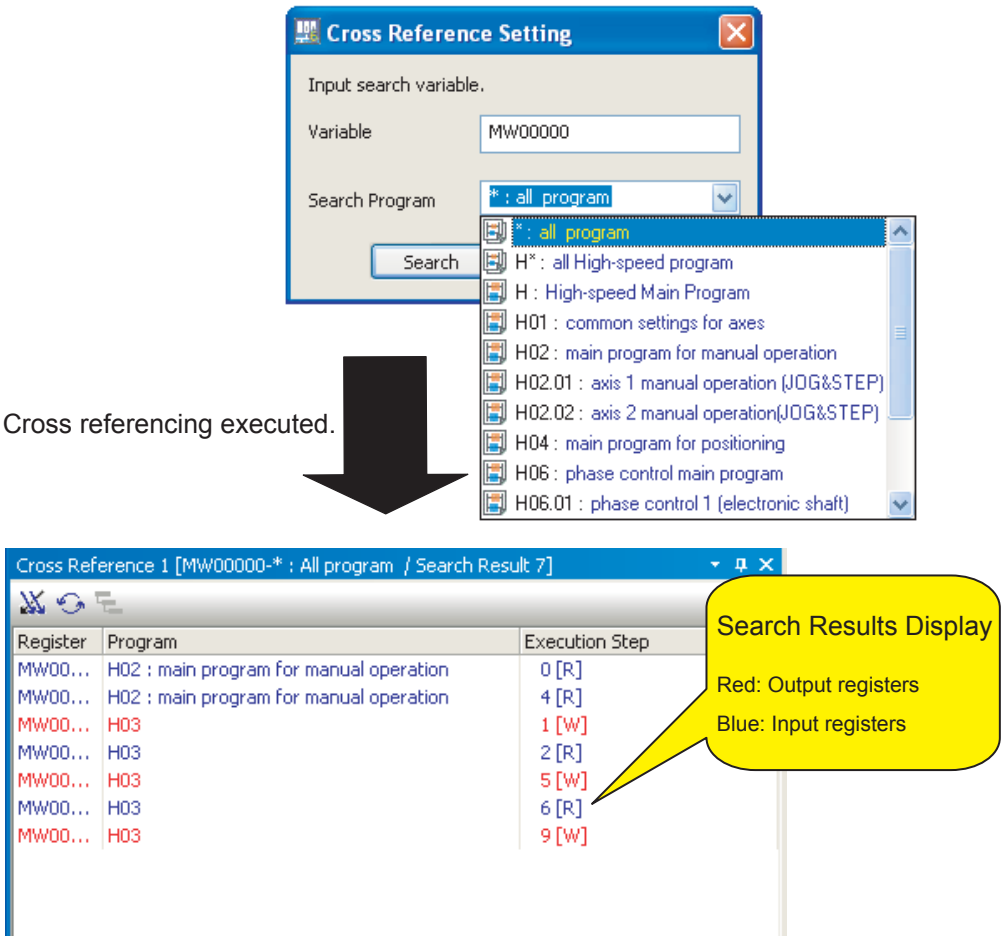

Search Dialog Box

Replace Dialog Box

You can search for and replace variables, comments, and other items.

# 6.3 Cross References

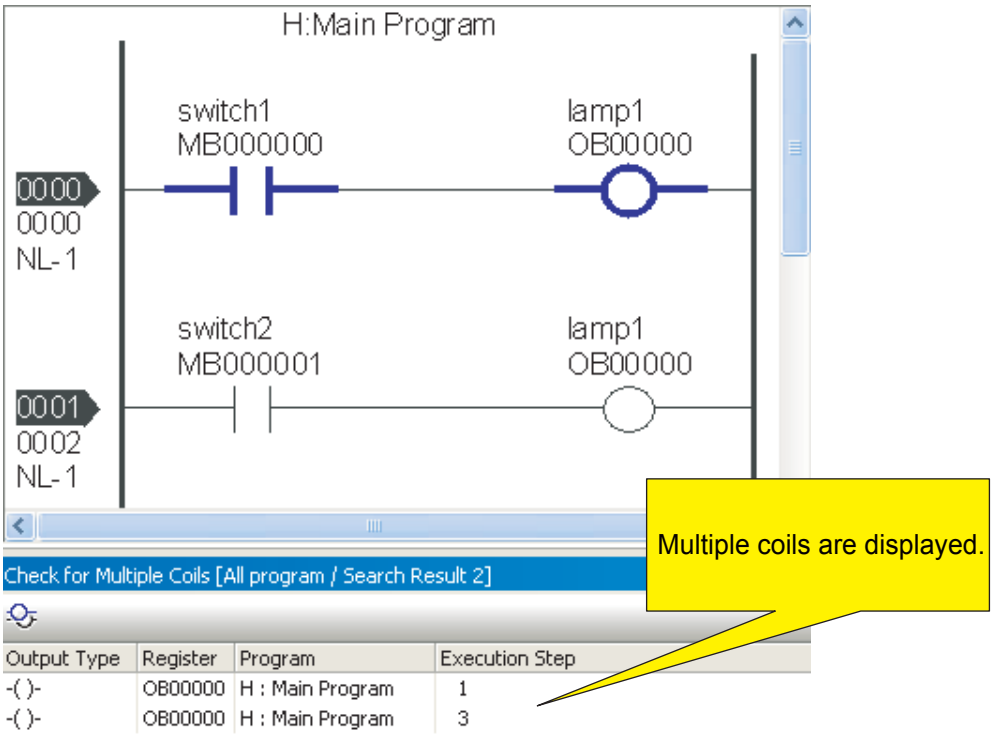
Cross referencing allows you to check whether a register is used in a program, and where it is used.
The search results indicate output registers in red, input registers in blue.




If the value of a register is different from its set value, it means that the value of the register may have been overwritten somewhere in the program. In this case, you can search for the registers using cross references. Check the registers displayed in red to locate the instructions that are overwriting them.

## 6.4 Checking for Multiple Coils

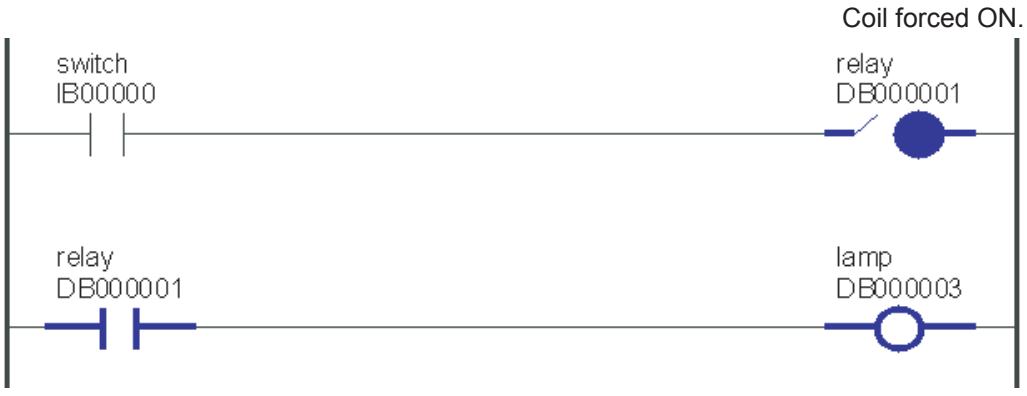
You can check for multiple coils (different coils that use the same register) in an entire ladder program, and display the search results.




## 6.5 Forcing Coils ON and OFF

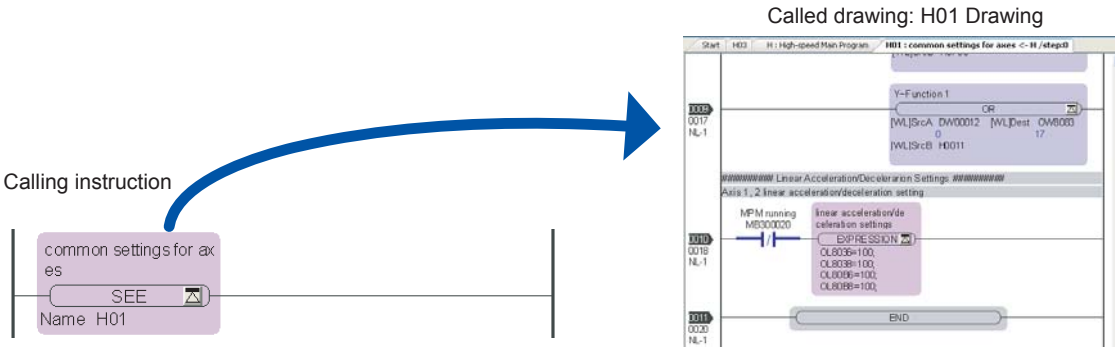
You can force a specified coil ON or OFF from the Ladder Editor.
The coil will output ON or OFF regardless of the output of the instruction to the left of the coil.

In the following example, you can simulate turning ON the switch (IB00000) by forcing the DB000001 relay ON even though the physical switch does not exist.



# 6.6 Viewing Called Programs

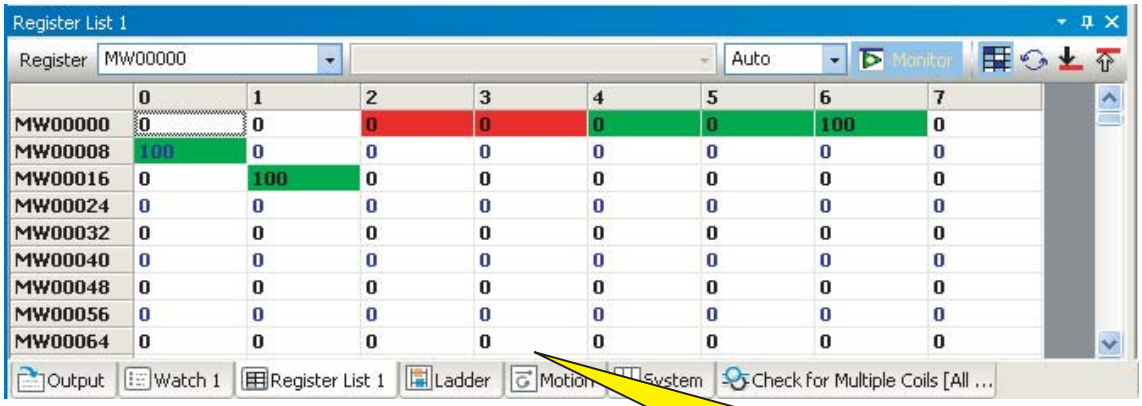
You can open a drawing that is called with an SEE instruction or an FUNC instruction.




You can open called drawings

# 6.7 Register Lists

You can monitor and edit the current values of the registers in a continuous area on a register list. Realtime monitoring and editing are possible if the Machine Controller is connected.
Also, if you turn ON display of the register map, registers that are used in the ladder program are displayed with a green background, and registers that are used for more than one data type are displayed with a red background.

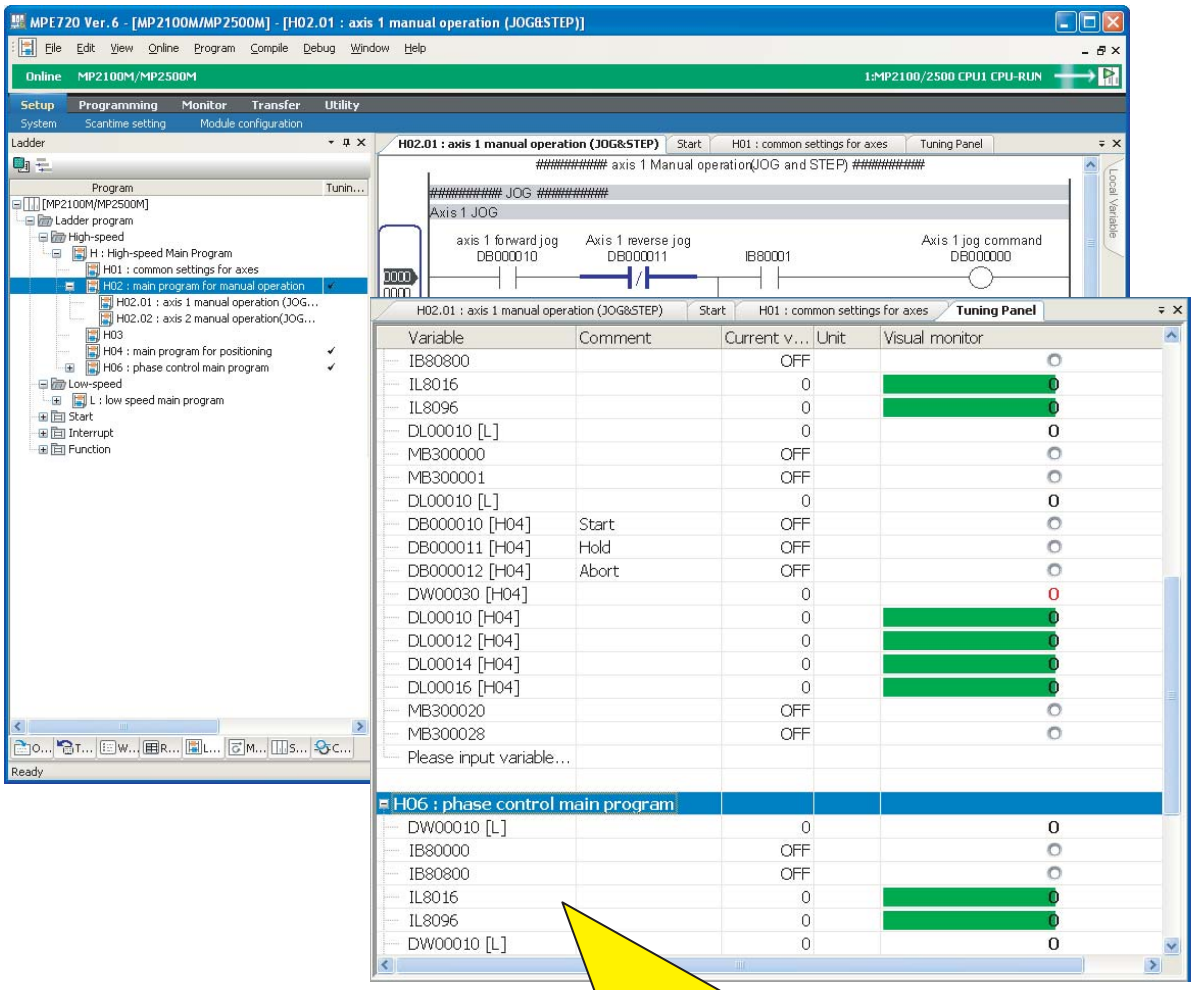



| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| MW00000 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 |
| MW00008 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00016 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00024 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00032 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00040 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00048 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00056 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MW00064 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

You can monitor and edit the current values of registers in a continuous area.

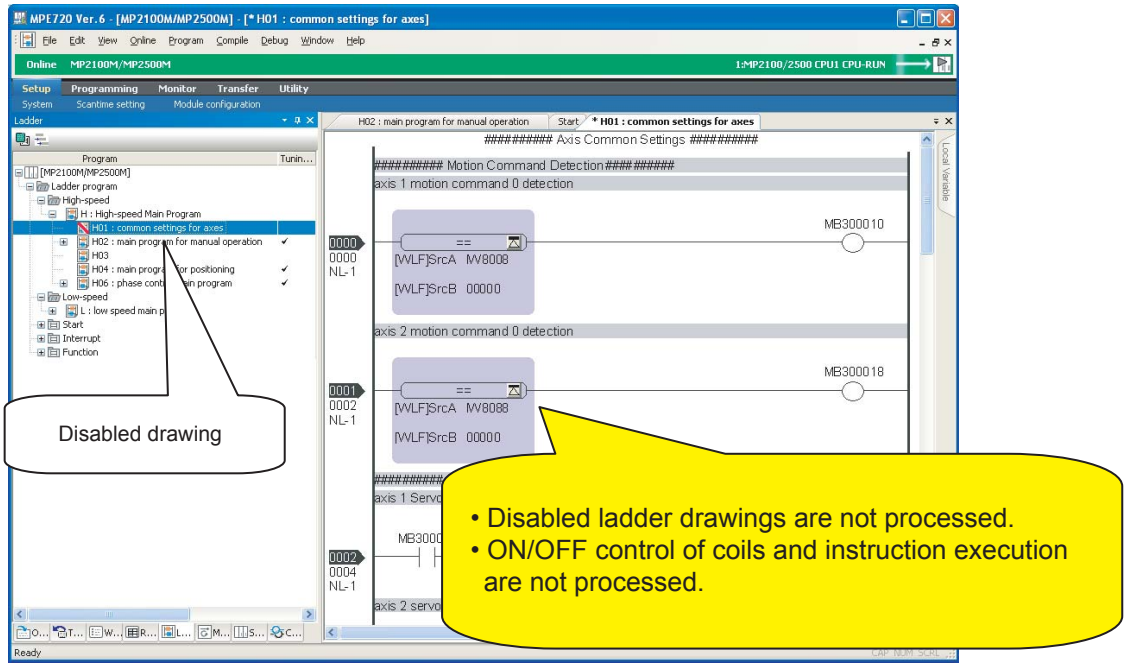
# 6.8 Tuning Panel

The Tuning Panel allows you to display and edit the current values of pre-registered variables.
You can use the Tuning Panel to control and check the operation of your application.
You can adjust the **Visual monitor** Column to display data according to specific conditions.

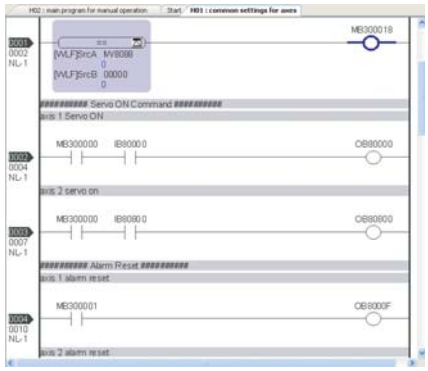


# 6.9 Enabling and Disabling Ladder Programs

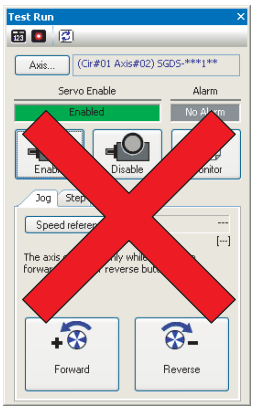
You can enable and disable individual drawings in ladder programs.



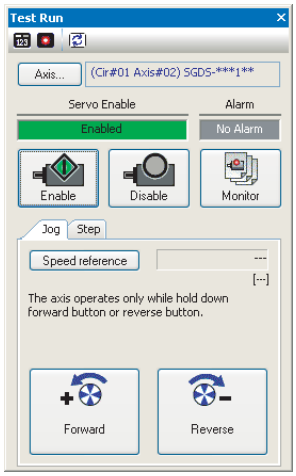

This feature is used to temporarily disable ladder drawings that contain processing to turn ON the power supply to servomotors or jog processing for servomotors. This allows you to check the operation of individual servomotors with the test run operation of the MPE720 or the Module configuration definition.



Ladder drawing disabled.




The required operation is not possible because the ladder drawing is active.

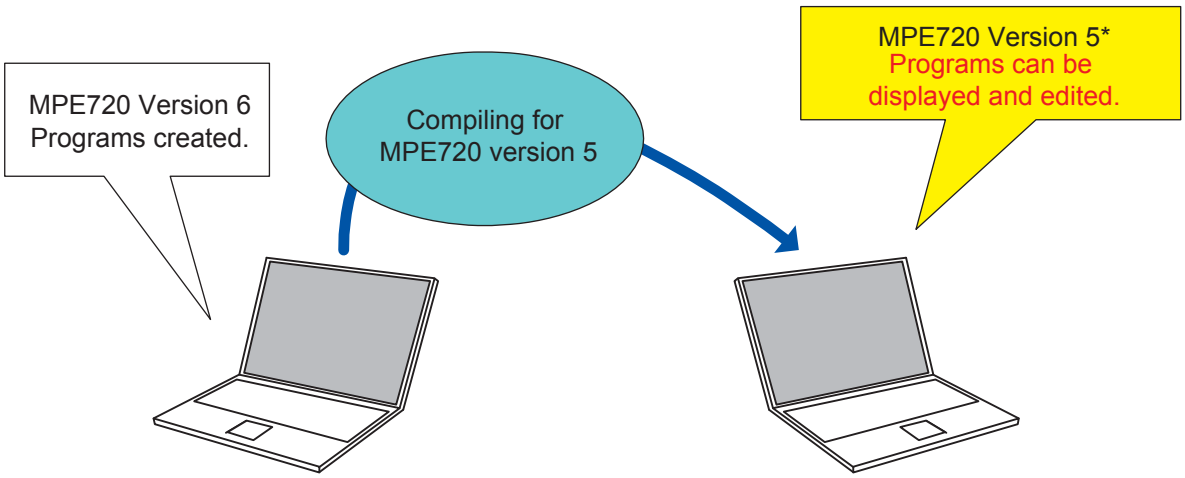



The motor can be controlled from MPE720 as required.

# 6.10 Compiling for MPE720 Version 5

Compiling for MPE720 version 5 allows you to display and edit ladder programs on MPE720 version 5 (version 5.34 or higher) even when you compile them on MPE720 version 6.

However, compiling errors will occur if notation that is supported only on MPE720 version 6 is used.
If you do not compile for MPE720 version5, you will not be able to display and edit the ladder programs that you create on MPE720 version 6 on MPE720 version 5.




* MPE720 version 5.34 or higher is required to display and edit programs that were compiled for MPE720 version 5 on MPE720 version 6.

# 7

# Troubleshooting

This chapter describes troubleshooting.


7.1 Basic Flow of Troubleshooting ----------7-2

7.2 Indicator Status ----------7-3

7.3 Problem Classifications ----------7-4

7.3.1 Overview ---------- 7-4

7.3.2 Error Checking Flowchart for MP2000-series Machine Controllers ---------- 7-5

7.4 Detailed Troubleshooting ----------7-6

7.4.1 Operation Errors ---------- 7-6

7.4.2 I/O Errors ---------- 7-9

7.4.3 Watchdog Timer Errors ---------- 7-10

7.4.4 Module Synchronization Errors ---------- 7-10

7.4.5 System Errors ---------- 7-11

# 7.1 Basic Flow of Troubleshooting

When a problem occurs, it is important to quickly find the cause of the problem and get the system running again as quickly as possible. The basic troubleshooting flow is illustrated below.

| Step 1 | Visually confirm the following items. |
|---|---|
| • Machine movement<br>• Power supply<br>• I/O device status<br>• Wiring status<br>• Indicator status (LED indicators on each Module)<br>• Switch settings (e.g., DIP switches)<br>• Parameter settings and program contents | |

| Step 2 | Monitor the system to see if the problem changes in response to the following operations. |
|---|---|
| • Switching the Controller to STOP status<br>• Resetting alarms<br>• Turning the power supply OFF and ON again | |

| Step 3 | Determine the location of the cause from the results of steps 1 and 2. |
|---|---|
| • Controller or external?<br>• Sequence control or motion control?<br>• Software or hardware? | |

# 7.2 Indicator Status

The pattern of the indicators on the MP2000-series Machine Controller shows the operating status. The following table gives the indicator lighting patterns and corresponding corrective actions.

| Class | Indicator Status | | | | | Meaning | Corrective Action |
|---|---|---|---|---|---|---|---|
| | RDY | RUN | ALM | ERR | BAT | | |
| Normal | Not lit. | Not lit. | Lit. | Lit. | Not lit. | Hardware has been reset. | Normally, the CPU Unit will start within 10 seconds. If more than 10 seconds is required, there is an error in a user program or a hardware failure. |
| | Not lit. | Not lit. | Not lit. | Not lit. | Not lit. | The Machine Controller is being initialized. | |
| | Not lit. | Lit. | Not lit. | Not lit. | Not lit. | Drawing A is being executed. | |
| | Lit. | Not lit. | Not lit. | Not lit. | Not lit. | The user program is stopped.<br>(The Machine Controller is in Offline Stopped Mode.) | This status is entered at the following times. It does not represent an error.<br>• The stop operation was performed from the MPE720.<br>• The STOP switch was turned ON. |
| | Lit. | Lit. | Not lit. | Not lit. | Not lit. | The user programs are being executed normally. | Normal operation is in progress. |
| Error | Not lit. | Not lit. | Not lit. | Lit. | Not lit. | A serious failure, watchdog timer error, or Module synchronization error has occurred. | A hardware failure, watchdog timer error, or Module synchronization error has occurred.<br>Refer to *7.3 Problem Classifications.* |
| | Not lit. | Not lit. | Not lit. | Flashing. | Not lit. | A software error occurred.<br>Number of Flashes<br>3: Read address error exception<br>3: Write address error exception<br>5: FPU exception<br>6: General illegal instruction exception<br>7: Slot illegal instruction exception<br>8: General FPU suppression exception<br>9: Slot FPU suppression exception<br>10: TLB multibit exception<br>11: LTB reading error exception<br>12: LTB writing error exception<br>13: LTB read protection violation exception<br>14: LTB write protection violation exception<br>15: Initial page write exception | A system error occurred.<br>Refer to *7.4.5 System Errors.* |
| | Not lit. | Not lit. | Flashing. | Flashing. | Not lit. | A hardware error occurred.<br>Number of Flashes<br>2: RAM diagnostic error<br>3: ROM diagnostic error<br>4: CPU Function Module diagnostic error<br>5: FPU Function Module diagnostic error | A hardware failure has occurred. Replace the Module. |
| Alarm | – | – | – | – | Lit. | Battery alarm | Replace the Battery. |
| | Lit. | Lit. | Lit. | Not lit. | Not lit. | An operation error occurred.<br>An I/O error occurred. | • Operation Errors<br>Refer to *7.4.1 Operation Errors.*<br>• I/O Errors<br>Refer to *7.4.2 I/O Errors.* |

# 7.3 Problem Classifications

## 7.3.1 Overview

The following table gives the problems that can occur on an MP2000-series Machine Controller and the indicator lighting patterns.

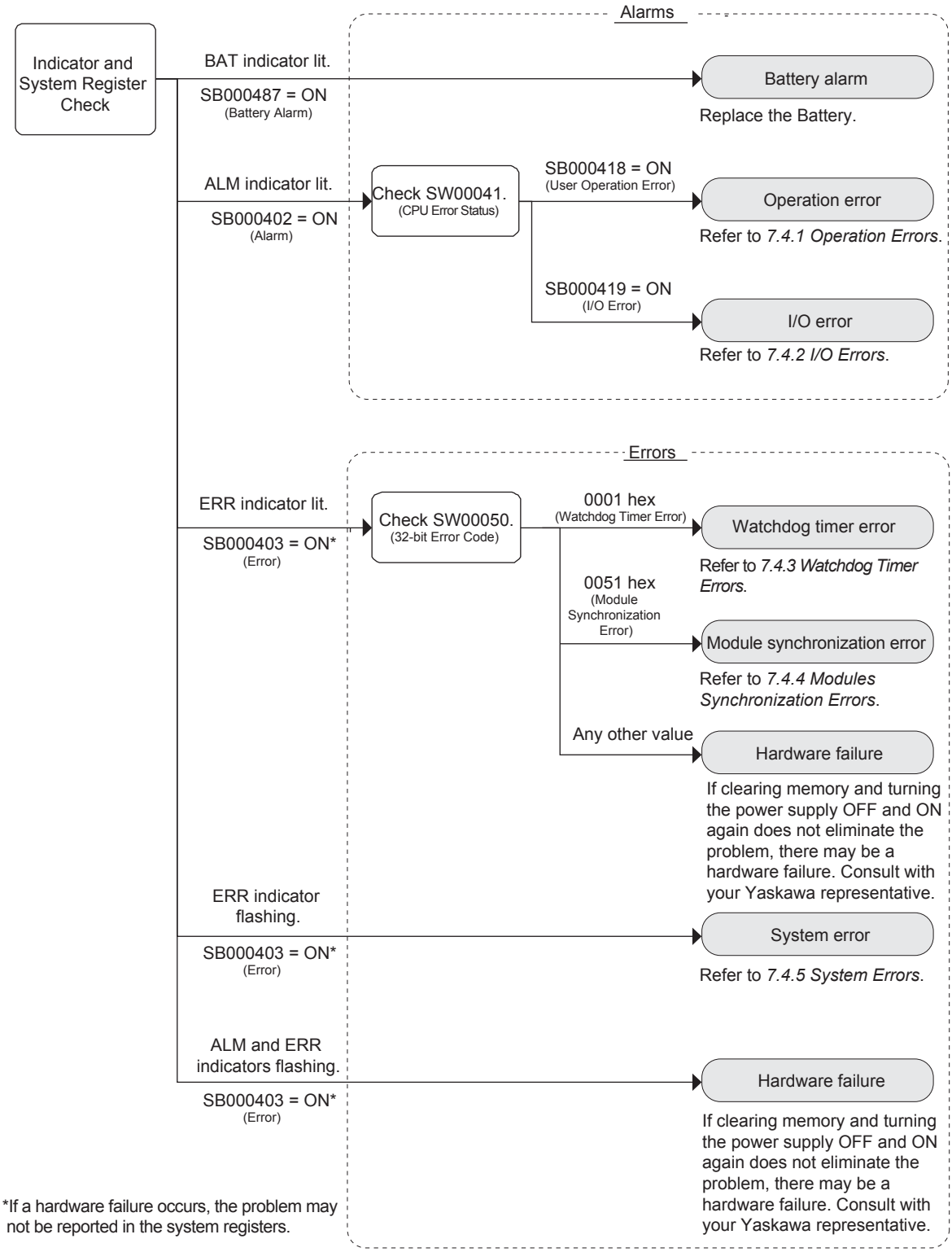
| Classification | Problem | Indicators | | |
|---|---|---|---|---|
| | | ALM | ERR | BAT |
| Alarm | Battery alarm | Not lit. | Not lit. | Lit. |
| | Operation error | Lit. | Not lit. | Not lit. |
| | I/O error | Lit. | Not lit. | Not lit. |
| | Motion program alarm*1 | Not lit. | Not lit. | Not lit. |
| | Axis alarm/warning *2 | Not lit. | Not lit. | Not lit. |
| Error | Watchdog timer error | Not lit. | Lit. | Not lit. |
| | Module synchronization error | Not lit. | Lit. | Not lit. |
| | System error | Not lit. | Flashing. | Not lit. |
| | Hardware failure | Not lit. | Lit. | Not lit. |
| | | Flashing. | Flashing. | Not lit. |

∗ 1. If a motion program alarm occurs, refer to *Chapter 10 Troubleshooting* in the *Machine Controller MP2000 Series User's Manual for Motion Programming* (Manual No.: SIEP C880700 38) and clear the alarm.

∗ 2. If an axis alarm/warning occurs, refer to the user's manual for your Motion Module and clear the alarm.

## 7.3.2 Error Checking Flowchart for MP2000-series Machine Controllers

Use the following flowchart to troubleshoot problems based on the indicators and system registers.



# 7.4 Detailed Troubleshooting

## 7.4.1 Operation Errors

Operation errors can be caused by the following problems.

- An illegal operation was performed in a ladder program.
- An illegal operation was performed in a motion program.
- An illegal operation was performed in a sequence program.

If an operation error occurs, use the following procedure to isolate the error.

**1.** Check the contents of SW00080 to SW00089 to identify the type of drawing and error.

Information on operation errors is stored in the following system registers according to the type of drawing in which the error occurred. Information on errors in motion programs is stored in the system registers for DWG.H.

<table>
<tr><th>Drawing Type</th><th>Error</th><th>Register Address</th><th>Description</th></tr>
<tr><td rowspan="2">DWG.A</td><td>Error Count</td><td>SW00080</td><td rowspan="8">• Error Count<br>Gives the number of errors that have occurred.<br><br>• Error Code<br>Bit 15 - - - - - 12 Bit 11 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 0<br><table><tr><td>Index error</td><td>Error code</td></tr></table><br>Refer to A.9 Interrupt Status for information on error codes.</td></tr>
<tr><td>Error Code</td><td>SW00081</td></tr>
<tr><td rowspan="2">DWG.I</td><td>Error Count</td><td>SW00082</td></tr>
<tr><td>Error Code</td><td>SW00083</td></tr>
<tr><td rowspan="2">DWG.H</td><td>Error Count</td><td>SW00084</td></tr>
<tr><td>Error Code</td><td>SW00085</td></tr>
<tr><td rowspan="2">DWG.L</td><td>Error Count</td><td>SW00088</td></tr>
<tr><td>Error Code</td><td>SW00089</td></tr>
</table>

■ Example: When SW00085 Contains a Value Other Than 0000 Hex

You can tell that an operation error occurred in high-speed scan processing. If the value in SW00084 is continuously incremented, it means that the instruction that is causing the operation error is being executed continuously.

**2.** Check the contents of SW00122, SW00138, SW00154, and SW00186 to identify the drawing number.

| Name | Register Address | Description |
|---|---|---|
| DWG.A Error Drawing No. | SW00122 | Parent drawing: FFFF hex<br>Child drawing: xx00 hex (xx hex: Child drawing number)<br>Grandchild drawing: xxyy hex (yy hex: Grandchild drawing number)<br>Function: 8000 hex<br>Motion program/sequence program: F0xx hex (xx hex: Program number) |
| DWG.I Error Drawing No. | SW00138 | |
| DWG.H Error Drawing No. | SW00154 | |
| DWG.L Error Drawing No. | SW00186 | |

**3.** Identify the instruction that caused the error in the drawing.

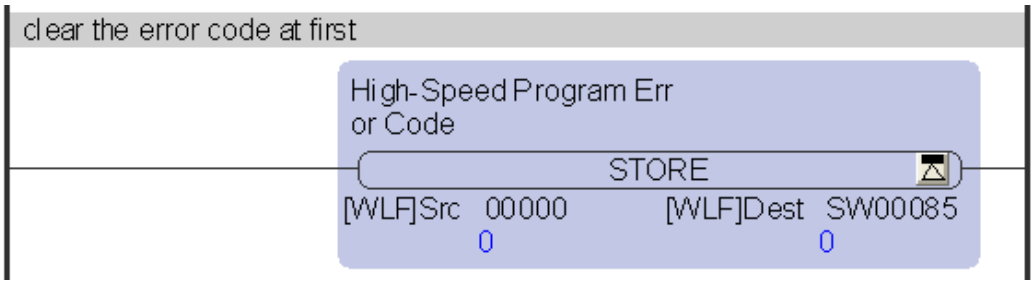
The method to identify operation errors is different for integer operations and real number operations. To identify operation errors for integer operations, refer to ■ *Troubleshooting Method 1.*
To identify operation errors for real number operations, refer to ■ *Troubleshooting Method 2.*

■ Troubleshooting Method 1

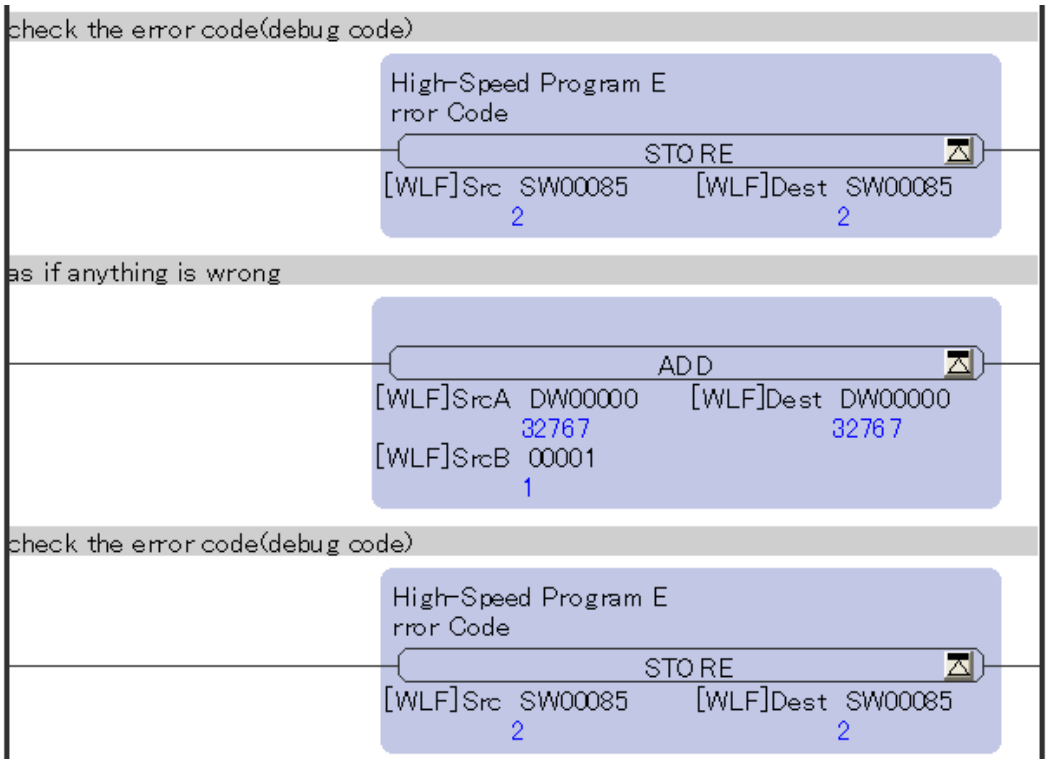
You can use the following procedure to troubleshoot operation errors that occur in DWG.H (0002 hex: Integer operation overflow).

**1.** Identify the error drawing number with the SW00154 system register and open that drawing.

**2.** Add the following code to the beginning of the drawing.




**3.** Add debugging code before and after the instruction that you think is causing the error.




**4.** Check the contents of the register address in the debugging code.
If it changes from 0 (no error) to 2 (integer operation overflow), an integer operation overflow has occurred.

**5.** Repeat steps 3 and 4, above, to isolate the instruction that is causing the error.

INFO

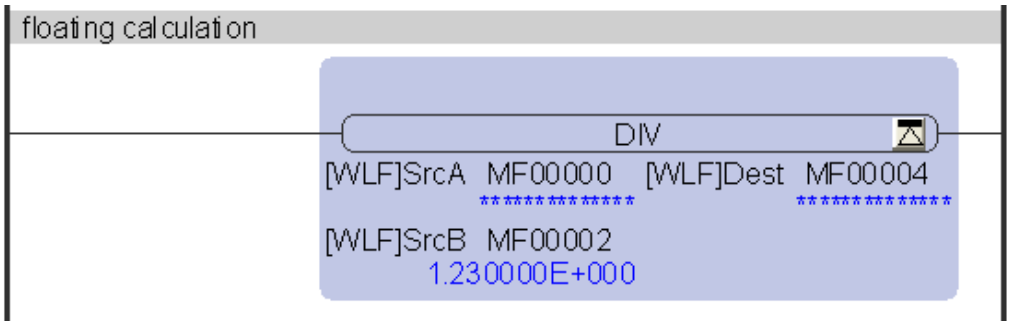
You can use the above debugging method only with an integer or double-length integer operation.
You cannot use the above debugging method with real number operations.

■ Troubleshooting Method 2

You can use the following procedure to troubleshoot operation errors that occur in DWG.H (0030 hex: Invalid real number operation (not a number)).

**1.** Identify the error drawing number with the SW00154 system register and open that drawing.

**2.** Check the value of the real number operation with the online monitor.




In this example, MF000000 in the DIV instruction is "**********".
"**********" indicates an illegal value for a real number (not a number). If you use that value in a real number operation, the system will generate an operation error (0030 hex: Invalid real number operation (not a number)).

INFO

Operation errors can be caused by the following problems.
- A value is not set in a register (undefined data).
- A bit, integer, or double-length integer operation was performed for a register that uses the same address.

To perform real number operations, you must set real number values.

**3.** Repeat step 2, above, to isolate the register that is causing the error.

## 7.4.2 I/O Errors

An I/O error can occur in the following cases.

- Option Module allocations or Module detail definitions were set in the Module Configuration.
- A cable was disconnected or a Module failed while the system was operating.

If an I/O error occurs, you can check the following system registers to check the I/O error.

| Name | Register Address | Description |
|---|---|---|
| I/O Error Count | SW00200 | Number of I/O errors that have occurred (total of SW00201 and SW00203). |
| Input Error Count | SW00201 | Number of input errors that have occurred |
| Output Error Count | SW00203 | Number of output errors that have occurred |

You can also use the following system registers to find the address of the I/O register (IW□□□□/OW□□□□) for which the I/O error occurred.

■ Example: When an I/O Error Was Detected for an I/O Device Assigned to IW1234

A value of 1234 hex will be stored in SW00202.

| Name | Register Address | Meaning |
|---|---|---|
| Input Error Address | SW00202 | The latest input error address (register address in IW□□□□) |
| Output Error Address | SW00204 | The latest output error address (register address in OW□□□□) |

After you find the I/O register address, identify the slot of the Module and then find the I/O status from the following system registers.
For details on I/O status, refer to *5.5.5 System I/O Error Status* in the *Machine Controller MP2000 Troubleshooting Manual* (Manual No.: SIJP C880700 40 (Japanese version)).

| Name | Register Address | Meaning |
|---|---|---|
| Input Error Address | SW00208 to SW00215 | CPU Function Module |
| | SW00216 to SW00223 | Reserved for system. |
| | SW00224 to SW00231 | Error status of Rack 1, Slot 1 |
| | SW00232 to SW00239 | Error status of Rack 1, Slot 2 |
| | SW00240 to SW00247 | Error status of Rack 1, Slot 3 |
| | SW00248 to SW00255 | Error status of Rack 1, Slot 4 |
| | ... | ... |
| | SW00496 to SW00503 | Error status of Rack 4, Slot 9 |

## 7.4.3 Watchdog Timer Errors

Watchdog timer errors can be caused by the following problems.

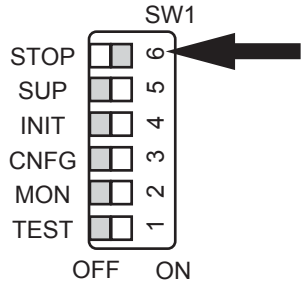
- An infinite loop occurs in a user program.
- The scan time is exceeded by a user program.
- A Motion Module*1 fails.
- A watchdog timer error occurs in an MPU-01 Module.*2

∗ 1. Motion Modules: PO-01, SVA-01, SVB-01, SVC-01, and MPU-01

∗ 2. If a watchdog error occurs when you are using an MPU-01 Module, refer to *Chapter 6 Troubleshooting* in the *Machine Controller MP2000 Series MPU-01 Multiple-CPU Module User's Manual* (Manual No.: SIEP C880781 05).

If a watchdog timer error occurs, it is important to determine if the cause of the error is in the CPU Module or in a Motion Module.

To determine where the cause of the error was, stop the programs in the CPU Module and then restart the CPU Module to see if the problem changes.




Turn ON the STOP switch on the front panel of the CPU Unit and then turn the power supply OFF and ON again to start the CPU Module without executing the ladder programs in the CPU Module.

If a watchdog timer does not occur when the programs in the CPU Module are stopped, it is very likely that the cause of the error is in the CPU Module. Check the programs to see if there are any infinite loops.

If this does not solve the problem, then there is a chance that the Motion Module is faulty. Consult with your Yaskawa representative.

## 7.4.4 Module Synchronization Errors

Module synchronization errors can be caused by the following problems.

- A Motion Module*1 fails.
- A watchdog timer error occurs in an MPU-01 Module.*2

∗ 1. Motion Modules: PO-01, SVA-01, SVB-01, SVC-01, and MPU-01

∗ 2. If a Module synchronization error occurs when you are using an MPU-01 Module, refer to *Chapter 6 Troubleshooting* in the *Machine Controller MP2000 Series MPU-01 Multiple-CPU Module User's Manual* (Manual No.: SIEP C880781 05).

If a Module synchronization error occurs (i.e., if SW00050 = 0051 hex), the slot where the Module synchronization error was detected is reported in the system registers given in the following table.

| Register Address | Description |
|---|---|
| SW00076 | Slot where Module synchronization error was detected*<br>xxyy hex: xx: Rack number (01 to 04), yy: Slot number (01 to 09) |

∗ Module synchronization errors are reported for CPU Modules with a system software version of 2.75 or higher. For version 2.74 or lower, it is reported as a watchdog timer error.

If a Module synchronization error occurs, consult with your Yaskawa representative.

## 7.4.5 System Errors

System errors can be caused by the following problems.

- Illegal processing was performed in a user program.
- A problem occurred in the installation environment.
- A hardware failure occurred.

If you are using embedded C-language programs, a system error that results in the system going down can be caused by illegal pointer access or an illegal operation on floating-point data. The causes of system errors are given in the following table.

| Number of Flashes of ERR Indicator | Error | Cause | Corrective Action |
|---|---|---|---|
| 3 times | Read address error exception | Long word (32-bit) or word (16-bit) data was read from an incorrect address.* | Check for the types of illegal processing given on the left and correct any problems. |
| 4 times | Write address error exception | | |
| 5 times | FPU exception | An illegal operation was performed for floating-point data (not a number, division by 0, overflow, etc.) | |

* For details, refer to *Chapter 10 Precautions* in the *Machine Controller MP2000 Series Embedded C-Language Programming Package Development Guide* (Manual No.: SIEP C880700 25).

If you are not using embedded C-language programs or if you are using embedded C-language programs and none of the above illegal programming problems exist, the cause may be a hardware error.
Hardware errors can be caused by the installation environment or by failures in the hardware itself.

If there are no problems in the installation environment and the error recurs regardless of corrective actions, the hardware itself may have failed. Consult with your Yaskawa representative.

# Appendix A

## System Registers

This appendix describes the registers that are provided by the system of the Machine Controller.


A.1 System Service Registers - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-2

A.2 System Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-6

A.3 System Error Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-7

A.4 Overview of User Operation Error Status - - - - - - - - - - - - - - - - - - - - - - - - A-9

A.5 System Service Execution Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-11

A.6 Detailed User Operation Error Status - - - - - - - - - - - - - - - - - - - - - - - - - - - A-11

A.7 System I/O Error Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-12

A.8 CF Card-related System Registers
(MP2200-series CPU-02 and CPU-03 only) - - - - - - - - - - - - - - - - - - - - - - - A-13

A.9 Interrupt Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-14

A.9.1 Interrupt Status List - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-14

A.9.2 Details on Interrupting Module - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-14

A.10 Module Information - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-15

A.11 MPU-01 System Status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-16

A.12 Motion Program Information - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-17

System registers are provided by the MP2000-series Machine Controller system. They can be used to read system error information, the current operating status, and other information.

| | Contents |
|---|---|
| SW00000 | System Service Registers |
| SW00030 | System Status |
| SW00050 | System Error Status |
| SW00080 | Overview of User Operation Error Status |
| SW00090 | System Service Execution Status |
| SW00110 | Detailed User Operation Error Status |
| SW00190 | Alarm Counter and Alarm Clear |
| SW00200 | System I/O Error Status |
| SW00504 | Reserved for system. |
| SW00652 | CF Card-related System Registers (MP2200-series CPU-02 and CPU-03 only) |
| SW00698 | Interrupt Status |
| SW00800 | Module Information |
| SW01312 | Reserved for system. |
| SW01411 | MPU-01 Module System Status |
| SW02048 | Reserved for system. |
| SW03200 | Motion Program Information |
| SW05200 to SW08191 | Reserved for system. |

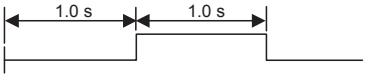
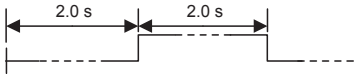
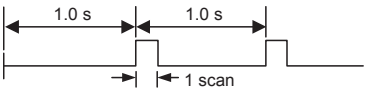
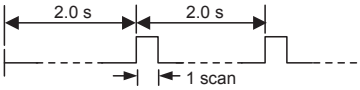
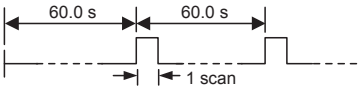
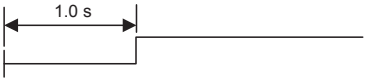
# A.1 System Service Registers

## ( 1 ) Common to All Drawings

| Name | Register Address | Remarks |
|---|---|---|
| Reserved for system. | SB000000 | Not used. |
| High-speed Scan | SB000001 | ON for only the first scan after the high-speed scan starts. |
| Low-speed Scan | SB000003 | ON for only the first scan after low-speed scan starts. |
| Always ON | SB000004 | Always ON (1). |
| Reserved for system. | SB000005 and SB000006 | Not used. |
| High-speed Scan in Progress | SB000007 | ON (1) during execution of the high-speed scan. |
| Reserved for system. | SB000008 to SB00000F | Not used. |

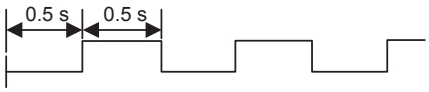
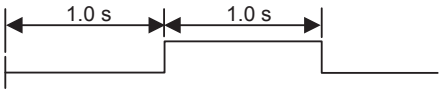
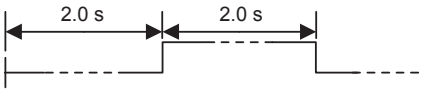
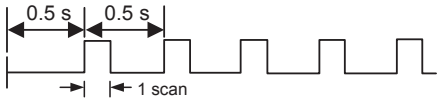
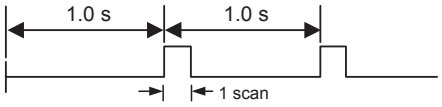
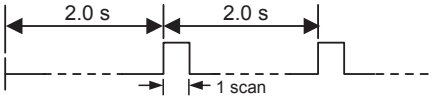
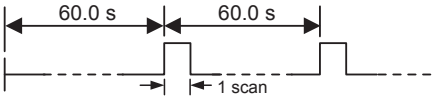
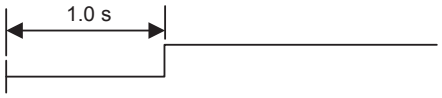
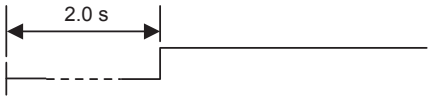
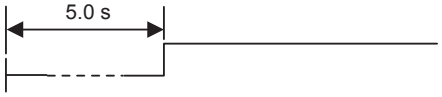
## ( 2 ) Exclusive to DWG.H Only

Operation starts when the high-speed scan starts.

| Name | Register Address | Remarks |
|---|---|---|
| 1-scan Flicker Relay | SB000010 | |
| 0.5-s Flicker Relay | SB000011 | |
| 1.0-s Flicker Relay | SB000012 |  |
| 2.0-s Flicker Relay | SB000013 |  |
| 0.5-s Sampling Relay | SB000014 | |
| 1.0-s Sampling Relay | SB000015 |  |
| 2.0-s Sampling Relay | SB000016 |  |
| 60.0-s Sampling Relay | SB000017 |  |
| 1.0 s After Start of Scan Relay | SB000018 |  |
| 2.0 s After Start of Scan Relay | SB000019 | |
| 5.0 s After Start of Scan Relay | SB00001A | |

## ( 3 ) Exclusive to DWG.L Only

Operation starts when the low-speed scan starts.

| Name | Register Address | Remarks |
|---|---|---|
| 1-scan Flicker Relay | SB000030 | |
| 0.5-s Flicker Relay | SB000031 |  |
| 1.0-s Flicker Relay | SB000032 |  |
| 2.0-s Flicker Relay | SB000033 |  |
| 0.5-s Sampling Relay | SB000034 |  |
| 1.0-s Sampling Relay | SB000035 |  |
| 2.0-s Sampling Relay | SB000036 |  |
| 60.0-s Sampling Relay | SB000037 |  |
| 1.0 s After Start of Scan Relay | SB000038 |  |
| 2.0 s After Start of Scan Relay | SB000039 |  |
| 5.0 s After Start of Scan Relay | SB00003A |  |

## ( 4 ) Scan Execution Status and Calendar

| Name | Register Address | Remarks |
|---|---|---|
| High-speed Scan Set Value | SW00004 | This is the set value of the high-speed scan (0.1 ms). |
| High-speed Scan Current Value | SW00005 | This is the current value of the high-speed scan (0.1 ms). |
| High-speed Scan Maximum Value | SW00006 | This is the maximum value of the high-speed scan (0.1 ms). |
| High-speed Scan Set Value 2 | SW00007 | This is the set value of the high-speed scan (1 μs). |
| High-speed Scan Current Value 2 | SW00008 | This is the current value of the high-speed scan (1 μs). |
| High-speed Scan Maximum Value 2 | SW00009 | This is the maximum value of the high-speed scan (1 μs). |
| Low-speed Scan Set Value | SW00010 | This is the set value of the low-speed scan (0.1 ms). |
| Low-speed Scan Current Value | SW00011 | This is the current value of the low-speed scan (0.1 ms) |
| Low-speed Scan Maximum Value | SW00012 | This is the maximum value of the low-speed scan (0.1 ms) |
| Reserved for system. | SW00013 | Not used. |
| Current Scan Time | SW00014 | This is the current value of the scan that is currently being executed (0.1 ms). |
| Calendar: Year | SW00015 | 1999: 0099 (BCD) (last two digits only) |
| Calendar: Month Day | SW00016 | December 31: 1231 (BCD) |
| Calendar: Hours and Minutes | SW00017 | 23: 59: 2359 (BCD) |
| Calendar: Seconds | SW00018 | 59 s: 59 (BCD) |
| Calendar: Week | SW00019 | 0: Sunday, 1: Monday, 2: Tuesday, 3: Wednesday, 4: Thursday, 5: Friday, 6: Saturday |

## ( 5 ) System Program Software Numbers and Remaining Program Memory Capacity

| Name | Register Address | Remarks |
|---|---|---|
| System Program Software Version | SW00020 | S□□□□ (□□□□ is replaced with the BCD value.) |
| System Number | SW00021 to SW00025 | Not used. |
| Remaining Program Memory Capacity | SL00026 | Bytes |
| Total Memory Capacity | SL00028 | Bytes |

# A.2 System Status

The system operating status and errors are stored in registers SW00040 to SW00048. You can check the system status to determine whether the cause of the error is hardware or software related.

| Name | Register Address | Contents | | |
|---|---|---|---|---|
| Reserved for system. | SW00030 to SW00039 | – | | |
| CPU Status | SW00040 | SB000400 | READY | 0: Error, 1: Ready |
| | | SB000401 | RUN | 0: Stopped, 1: Running |
| | | SB000402 | ALARM | 0: Normal, 1: Alarm |
| | | SB000403 | ERROR | 0: Normal, 1: Error |
| | | SB000404 | Reserved for system. | – |
| | | SB000405 | Reserved for system. | – |
| | | SB000406 | FLASH | 1: Flash operation |
| | | SB000407 | WEN | 0: Writing disabled, 1: Writing enabled |
| | | SB000408 to SB00040D | Reserved for system. | – |
| | | SB00040E | Operation Stop Request | 0: RUN selected, 1: STOP selected |
| | | SB00040F | Run Switch Status at Power ON | 0: STOP, 1: RUN |
| CPU Error Status | SW00041 | SB000410 | Serious Failure | 1: WDGE, undefined instruction Refer to SW00050 for details. |
| | | SB000411 | Reserved for system. | – |
| | | SB000412 | Reserved for system. | – |
| | | SB000413 | Exception Error | – |
| | | SB000414 to SB000417 | Reserved for system. | – |
| | | SB000418 | User Operation Error | 1: User operation error |
| | | SB000419 | I/O Error | 1: I/O error |
| | | SB00041A to SB00041F | Reserved for system. | – |
| H Scan Exceeded Counter | SW00044 | – | – | – |
| L Scan Exceeded Counter | SW00046 | – | – | – |
| Reserved for system. | SW00047 | | Reserved for system. | – |
| Hardware Configuration Status | SW00048 | SB000480 | TEST | DIP switch status 0: ON, 1: OFF |
| | | SB000481 | MON | |
| | | SB000482 | CNFG | |
| | | SB000483 | INIT | |
| | | SB000484 | SUP | |
| | | SB000485 | STOP | |
| | | SB000486 | | |
| | | SB000487 | Battery Alarm | – |
| | | SB000488 to SB00048F | Reserved for system. | – |
| Reserved for system. | SW00049 | SB000490 to SB00049F | Reserved for system. | – |

# A.3 System Error Status

Details on the system errors are stored in registers SW00050 to SW00079.

| Name | Register Address | Contents | |
|---|---|---|---|
| 32-bit Error Code | SW00050 | 0001 hex | Watchdog timer error |
| | | 0041 hex | ROM diagnostic error |
| | | 0042 hex | RAM diagnostic error |
| | | 0043 hex | CPU diagnostic error |
| | | 0044 hex | FPU diagnostic error |
| | | 0050 hex | EXIO error |
| | | 0051 hex | Module synchronization error*1 |
| | | 00E0 hex | Read address exception error |
| | | 0100 hex | Write address exception error |
| | | 0120 hex | FPU exception error |
| | | 0180 hex | General illegal instruction exception error |
| | | 01A0 hex | Slot illegal instruction exception error |
| | | 01E0 hex | User break after instruction execution |
| | | 0800 hex | General FPU suppression exception error |
| | | 0820 hex | Slot FPU suppression exception error |
| | SW00051 | For system error analysis | |
| 32-bit Error Address | SW00052 | For system error analysis | |
| | SW00053 | | |
| Error Task | SW00054 | 0000 hex: System, 0001 hex: DWG.A, 0002 hex: DWG.I,<br>0003 hex: DWG.H, 0005 hex: DWG.L | |
| Program Type | SW00055 | 0000 hex: System, 0001 hex: DWG.A, 0002 hex: DWG.I,<br>0003 hex: DWG.H, 0005 hex: DWG.L, 0008 hex: Function,<br>000F hex: Motion program/sequence program | |
| Error Drawing No. | SW00056 | Ladder program parent drawing: FFFF hex<br>Ladder program function: 8000 hex<br>Ladder program child drawing: xx00 hex (xx hex: Child drawing number)<br>Ladder program grandchild drawing: xxyy hex (yy hex: Grandchild drawing number)<br>Motion program/sequence program: F0xx hex (xx hex: Program number) | |
| Calling Drawing Type | SW00057 | Type of the calling drawing in which the error occurred | |
| | | 0001 hex: DWG.A, 0002 hex: DWG.I, 0003 hex: DWG.H,<br>005 hex: DWG.L, 8000 hex: Ladder program function,<br>000F hex: Motion program/sequence program,<br>0010 hex: Reserved for system, 0011 hex: Reserved for system. | |
| Calling Drawing No. | SW00058 | Number of the calling drawing in which the error occurred | |
| | | Parent drawing:<br>FFFF hex<br>Function: 0100 hex | Child drawing: xx00 hex<br>(xx hex: Child drawing number)<br>Grandchild drawing: xxyy hex<br>(yy hex: Grandchild drawing number) |
| Calling Drawing Step No. | SW00059 | Step number in the calling drawing in which the error occurred<br>This number is set to 0 if the error occurred in the parent drawing. | |

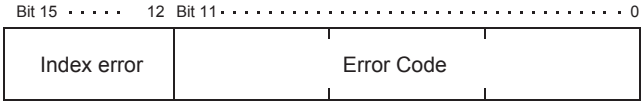
| Name | Register Address | Contents |
|---|---|---|
| Error Data | SW00060 and SW00061 | Reserved for system. |
| | SW00062 to SW00065 | Name of task that caused the error |
| | SW00066 and SW00067 | Reserved for system. |
| | SW00068 | Year when error occurred |
| | SW00069 | Month when error occurred |
| | SW00070 | Day of week when error occurred |
| | SW00071 | Day when error occurred |
| | SW00072 | Hour when error occurred |
| | SW00073 | Minutes when error occurred |
| | SW00074 | Seconds when error occurred |
| | SW00075 | Milliseconds when error occurred (Not used.) |
| | SW00076 | Slot where module synchronization error was detected*2<br>xxyy hex: xx: Rack number (01 to 04), yy: Slot number (01 to 09) |
| | SW00078 and SW00079 | Reserved for system. |

∗ 1. This error is reported for CPU Modules with a system software version of 2.75 or higher. For version 2.74 or lower, it is reported as a watchdog timer error (0001 hex).

∗ 2. This error is reported for CPU Modules with a system software version of 2.75 or higher.

# A.4 Overview of User Operation Error Status

Details are given in registers SW00080 to SW00089 when a user operation error occurs in a program.

| Name | | Register Address | Contents |
|---|---|---|---|
| DWG.A | Error Count | SW00080 | • Error Count<br>Gives the number of errors that have occurred.<br><br>• Error Code<br>  <br><br>Refer to *( 1 ) User Operation Error Code -1* or *( 2 ) User Operation Error Code -2* for information on error codes. |
| | Error Code | SW00081 | |
| DWG.I | Error Count | SW00082 | |
| | Error Code | SW00083 | |
| DWG.H | Error Count | SW00084 | |
| | Error Code | SW00085 | |
| DWG.L | Error Count | SW00088 | |
| | Error Code | SW00089 | |

## ( 1 ) User Operation Error Code -1

| | Error Code | Error Description | System Default |
|---|---|---|---|
| Integer Operations | 0001 hex | Integer operation underflow | -32,768 |
| | 0002 hex | Integer operation overflow | 32,767 |
| | 0003 hex | Integer operation division error | The A register stays the same. |
| | 0009 hex | Double-length integer operation underflow | -2,147,483,648 |
| | 000A hex | Double-length integer operation overflow | 2,147,483,647 |
| | 000B hex | Double-length integer operation division error | The A register stays the same. |
| Real Number Operations | 0010 hex | Non-numerical integer storage error | Data is not stored. [00000] |
| | 0011 hex | Integer storage underflow | Data is not stored. [-32,768] |
| | 0012 hex | Integer storage overflow | Data is not stored. [ + 32,767] |
| | 0021 hex | Real number storage underflow | Data is not stored. [-1.0E + 38] |
| | 0022 hex | Real number storage overflow | Data is not stored. [1.0E + 38] |
| | 0023 hex | Real number operation division by zero error | Data is not stored. [F register stays the same.] |
| | 0030 hex | Invalid real number operation (not a number) | Data is not stored. |
| | 0031 hex | Real number operation exponent underflow | 0.0 |
| | 0032 hex | Real number operation exponent overflow | Maximum Value |
| | 0033 hex | Real number operation division error (0/0) | Operation is not executed. |
| | 0034 hex | Real number storage exponent underflow | A value of 0.0 is stored. |
| | 0035 hex | Real number operation stack error | – |
| | 0040 to 0059 hex | Real number operation error in standard system function | Operation is aborted and output is set to 0.0. |

| 0040 to 0059 hex | | | | |
|---|---|---|---|---|
| | 0040 hex: SQRT | 0041 hex: SIN | 0042 hex: COS | 0043 hex: TAN |
| | 0044 hex: ASIN | 0045 hex: ACOS | 0046 hex: ATAN | 0047 hex: EXP |
| | 0048 hex: LN | 0049 hex: LOG | 004A hex: DZA | 004B hex: DZB |
| | 004C hex: LIM | 004D hex: PI | 004E hex: PD | 004F hex: PID |
| | 0050 hex: LAG | 0051 hex: LLAG | 0052 hex: FGN | 0053 hex: IFGN |
| | 0054 hex: LAU | 0055 hex: SLAU | 0056 hex: REM | 0057 hex: RCHK |
| | 0058 hex: BSRCH | 0059 hex: SORT | | |
| | For an index error, 1000, 2000, or 3000 hex is added. | | | |

## ( 2 ) User Operation Error Code -2

| | Error Code | Error Description | System Default |
|---|---|---|---|
| Integer and Real Number Operations | 1000 hex | Index error in drawing | Re-executed as if i and j were set to 0. (Both i and j registers stay the same.) |
| | 2000 hex | Index error in function | Re-executed as if i and j were set to 0. (Both i and j registers stay the same.) |
| | 3000 hex | Index error in motion program or sequence program | Re-executed as if i and j were set to 0. (Both i and j registers stay the same.) |
| Integer Operations | □060 to □0C9 hex (□ = 1, 2, or 3) | Index error in integer system function | Operation is aborted and output is set to input. |

| | | | | |
|---|---|---|---|---|
| | □06D hex: PI | □06E hex: PD | □06F hex: PID | □070 hex: LAG |
| | □071 hex: LLAG | □072 hex: FGN | □073 hex: IFGN | □074 hex: LAU |
| | □075 hex: SLAU | □076 hex: FGN | □077 hex: IFGN | □08E hex: INS |
| | □08F hex: OUTS | □090 hex: ROTL | □091 hex: ROTR | □092 hex: MOVB |
| | □093 hex: MOVW | □094 hex: SETW | □095 hex: XCHG | □096 hex: LIMIT |
| | □097 hex: LIMIT | □098 hex: DZA | □099 hex: DZA | □09A hex: DZB |
| | □09B hex: DZB | □09C hex: PWM | □09E hex: SHFTL | □09F hex: SHFTR |
| | □0A0 hex: BEXTEND | □0A1 hex: BPRESS | □0A2 hex: SORT | □0A4 hex: SORT |
| | □0A6 hex: RCHK | □0A7 hex: RCHK | □0A8 hex: COPYW | □0A9 hex: ASCII |
| | □0AA hex: BINASC | □0AB hex: ASCBIN | □0AC hex: BSRC H | □0AD hex: BSRC H |
| | □0AE hex: TIMEADD | □0AF hex: TIMSUB | □0B1 hex: SPEND | □0C0 hex: TBLBR |
| | □0C1 hex: TBLBW | □0C2 hex: TBLSRL | □0C3 hex: TBLSRC | □0C4 hex: TBLCL |
| | □0C5 hex: TBLMW | □0C6 hex: QTBLR | □0C7 hex: QTBLRI | □0C8 hex: QTBLW |
| | □0C9 hex: QTBLWI | | | |

# A.5 System Service Execution Status

The execution status of system services is stored in registers SW00090 to SW00103.

| Name | Register Address | Remarks |
|---|---|---|
| Reserved for system. | SW00090 | – |
| Reserved for system. | SW00091 | |
| Reserved for system. | SW00092 | |
| Reserved for system. | SW00093 | |
| Reserved for system. | SW00094 to SW00097 | – |
| Data Trace Definition Existence | SW00098 | Bits 0 to 3: Groups 1 to 4<br>Defined: 1, Not defined: 0 |
| Data Trace Execution Status | SW00099 | Bits 0 to 3: Groups 1 to 4<br>Trace stopped: 1, Trace in progress: 0 |

Latest Record Numbers in Data Trace

| Name | Register Address | Remarks |
|---|---|---|
| Data Trace Group 1 | SW00100 | Latest record number |
| Data Trace Group 2 | SW00101 | Latest record number |
| Data Trace Group 3 | SW00102 | Latest record number |
| Data Trace Group 4 | SW00103 | Latest record number |

# A.6 Detailed User Operation Error Status

Detailed information is given in registers SW00110 to SW00189 when a user operation error occurs in a program.

| Name | Register Address | | | | Contents |
|---|---|---|---|---|---|
| | DWG.A | DWG.I | DWG.H | DWG.L | |
| Error Count | SW00110 | SW00126 | SW00142 | SW00174 | • Error Counts and Error Codes<br>Same as in *Appendix A.4 Overview of User Operation Error Status.*<br>• Error Drawing No.<br>Parent drawing: FFFF hex<br>Child drawing: xx00 hex<br>(xx hex: Child drawing number)<br>Grandchild drawing: xxyy hex<br>(yy hex: Grandchild drawing number)<br>Function: 8000 hex<br>Motion program/sequence program:<br>F0xx hex (xx hex: Program number)<br>• Calling Drawing No.<br>Number of the calling drawing in which the operation error occurred<br>• Calling Drawing Step No.<br>Step number in the calling drawing in which the operation error occurred<br>This number is set to 0 if the error occurred in the parent drawing. |
| Error Code | SW00111 | SW00127 | SW00143 | SW00175 | |
| Reserved for system. | SW00112 to SW00121 | SW00128 to SW00137 | SW00144 to SW00153 | SW00176 to SW00185 | |
| Error Drawing No. | SW00122 | SW00138 | SW00154 | SW00186 | |
| Calling Drawing No. | SW00123 | SW00139 | SW00155 | SW00187 | |
| Calling Drawing Step No. | SW00124 | SW00140 | SW00156 | SW00188 | |
| Reserved for system. | SW00125 | SW00141 | SW00157 | SW00189 | |

# A.7 System I/O Error Status

Details on the system I/O errors are stored in registers SW00200 to SW00503.

| Name | Register Address | Contents |
|---|---|---|
| I/O Error Count | SW00200 | Number of I/O error occurrences |
| Input Error Count | SW00201 | Number of input error occurrences |
| Input Error Address | SW00202 | The latest input error address (register address in IW□□□□) |
| Output Error Count | SW00203 | Output error count |
| Output Error Address | SW00204 | The latest output error address (register address in OW□□□□) |
| Reserved for system. | SW00205 | Not used. |
| | SW00206 | |
| | SW00207 | |
| I/O Error Status | SW00208 to SW00215 | Error status for CPU Module |
| | SW00216 to SW00223 | Reserved for system. |
| | SW00224 to SW00231 | Error status of Rack 1, Slot 1 |
| | SW00232 to SW00239 | Error status of Rack 1, Slot 2 |
| | SW00240 to SW00247 | Error status of Rack 1, Slot 3 |
| | SW00248 to SW00255 | Error status of Rack 1, Slot 4 |
| | … | … |
| | SW00496 to SW00503 | Error status of Rack 4, Slot 9 |

# A.8 CF Card-related System Registers (MP2200-series CPU-02 and CPU-03 only)

The status of the CF Card is reported in registers SW00652 to SW00659.
These registers can be used only when a CF card is supported (the MP2200 with the CPU-02 or CPU-03).
For all other models, they are reserved for the system.

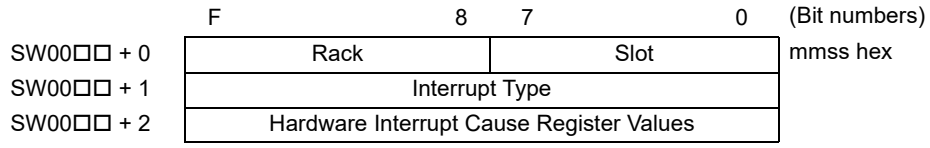
| Name | Register Address | | Contents |
|---|---|---|---|
| Total capacity of CF card | SL00652 | | Unit: Bytes |
| Card status | SW00654 | SB006540 | 0: CF card not mounted, 1: CF card mounted |
| | | SB006541 | 0: Not supplying power, 1: Supplying power |
| | | SB006542 | 0: Cannot detect a CF card, 3: CF card detected |
| | | SB006543 | 0: Not accessing CF card, 1: Accessing CF card |
| | | SB006544 | 0: –, 1: Checking FAT file system |
| | | SB006545 to SB00654F | Reserved for system. |
| FAT type | SW00655 | 0001 hex | FAT12 |
| | | 0002 hex | FAT16 |
| | | 0003 hex | FAT32 |
| Reserved for system. | SW00656 | | – |
| Reserved for system. | SW00657 | | – |
| Batch load and batch save | SW00658 | SB006580 | Batch load in progress |
| | | SB006581 | CF card reading error |
| | | SB006582 | Load file model mismatch error |
| | | SB006583 | Load file write error |
| | | SB006584 | Save to flash memory error |
| | | SB006585 | Folder for batch loading does not exist. |
| | | SB006586 | Loading error due to program write protection |
| | | SB006587 | Reserved for system. |
| | | SB006588 | Batch save in progress |
| | | SB006589 | CF card writing error |
| | | SB00658A | Save file read error |
| | | SB00658B | Security error |
| | | SB00658C to SB00658F | Reserved for system. |
| Reserved for system. | SW00659 | | – |

# A.9 Interrupt Status

## A.9.1 Interrupt Status List

| Name | Register Address | Remarks |
|---|---|---|
| Interrupt information | SW00698 | Interrupt detection count |
| | SW00699 | Number of interrupting methods |
| | SW00700<br>SW00701<br>SW00702 | Interrupting module 1 |
| | SW00703<br>SW00704<br>SW00705 | Interrupting module 2 |
| | :<br>: | |
| | SW00787<br>SW00788<br>SW00789 | Interrupting module 30 |

## A.9.2 Details on Interrupting Module




### ( 1 ) Rack

mm = 01 to 04
The rack number where the Module in which the interrupt occurred is mounted is reported.

### ( 2 ) Slot

ss = 01 to 09
The slot number where the Module in which the interrupt occurred is mounted is reported.

### ( 3 ) Interrupt Type

1: DI interrupt for CPU IO (MP2100, MP2100M, MP2101, MP2101M, MP2101T, MP2101TM, or MP2300)
2: DI interrupt for LIO-01, LIO-02, LIO-04, or LIO-05
3: Counter interrupt for LIO-01, LIO-02, LIO-06, or CNTR-01

### ( 4 ) Hardware Interrupt Cause Register Values

For the hardware interrupt cause register values, refer to *2.5.6 Interrupt Status* in the *Machine Controller MP2000 Troubleshooting Manual* (Manual No.: SIJP C880700 40 (Japanese version)).

# A.10 Module Information

The Module information is reported as shown in this section.

- The contents of the registers depends on the model. Refer to the manuals for your Machine Controller.

## ( 1 ) CPU Function Module

| Name | Register Address | Remarks |
|---|---|---|
| CPU Unit Information | SW00800 | CPU Module ID |
| | SW00801 | Hardware version (BCD) |
| | SW00802 | Software version (BCD) |
| | SW00803 | Number of sub-slots (hex) |
| | SW00804 | Function Module 1 ID (hex) |
| | SW00805 | Function Module 1 Status |
| | SW00806 | Function Module 2 ID (hex) |
| | SW00807 | Function Module 2 Status |
| | SW00808 | Function Module 3 ID (hex) |
| | SW00809 | Function Module 3 Status |
| | SW00810 | Function Module 4 ID (hex) |
| | SW00811 | Function Module 4 Status |
| | SW00812 | Function Module 5 ID (hex) |
| | SW00813 | Function Module 5 Status |
| | SW00814 | Function Module 6 ID (hex) |
| | SW00815 | Function Module 6 Status |
| Option Module Information | SW00816 to SW01095 | Option Module Information<br>(Depends on the CPU model and mounted Option Modules.) |

## ( 2 ) Option Modules

| Name | Register Address | Remarks |
|---|---|---|
| Module Information | SW00□□□ + 0 | Option Module ID |
| | SW00□□□ + 1 | Hardware version (BCD) |
| | SW00□□□ + 2 | Software version (BCD) |
| | SW00□□□ + 3 | Number of sub-slots (hex) |
| | SW00□□□ + 4 | Function Module 1 ID (hex) |
| | SW00□□□ + 5 | Function Module 1 Status |
| | SW00□□□ + 6 | Function Module 2 ID (hex) |
| | SW00□□□ + 7 | Function Module 2 Status |

## ( 3 ) Function Module Status Details

| Value | Text Displayed in MPE720 Module Configuration Definition | Status |
|---|---|---|
| 0 | None | There is no Module Definition and the Module is not mounted. |
| 1 | Empty | There is a Function Module Definition, but the Module is not mounted. |
| 2 | Operating (Driving) | The Module is operating normally. |
| 3 | Standby<br>(Reserved for system.) | The Module is on standby. |
| 4 | Failure | An error was detected in the Module. |
| 5 | × Module name | The mounted Module does not match the definition. |
| 6 | Waiting for initialization | The Module is mounted, but there is no Detailed Function Module Definition. |
| 7 | Driving stop | Local I/O is stopped. |
| 8 or higher | – | Reserved for system. |

# A.11 MPU-01 System Status

| Name | Register Address | Remarks |
|---|---|---|
| MPU-01 #1 Status | SW01411 | Status of MPU-01 Module circuit number 1 |
| MPU-01 #1 Error Status | SW01412 | Error status of MPU-01 Module circuit number 1 |
| MPU-01 #2 Status | SW01413 | Status of MPU-01 Module circuit number 2 |
| MPU-01 #2 Error Status | SW01414 | Error status of MPU-01 Module circuit number 2 |
| MPU-01 #3 Status | SW01415 | Status of MPU-01 Module circuit number 3 |
| MPU-01 #3 Error Status | SW01416 | Error Status of MPU-01 Module circuit number 3 |
| MPU-01 #4 Status | SW01417 | Status of MPU-01 Module circuit number 4 |
| MPU-01 #4 Error Status | SW01418 | Error Status of MPU-01 Module circuit number 4 |
| MPU-01 #5 Status | SW01419 | Status of MPU-01 Module circuit number 5 |
| MPU-01 #5 Error Status | SW01420 | Error Status of MPU-01 Module circuit number 5 |
| MPU-01 #6 Status | SW01421 | Status of MPU-01 Module circuit number 6 |
| MPU-01 #6 Error Status | SW01422 | Error Status of MPU-01 Module circuit number 6 |
| MPU-01 #7 Status | SW01423 | Status of MPU-01 Module circuit number 7 |
| MPU-01 #7 Error Status | SW01424 | Error Status of MPU-01 Module circuit number 7 |
| MPU-01 #8 Status | SW01425 | Status of MPU-01 Module circuit number 8 |
| MPU-01 #8 Error Status | SW01426 | Error Status of MPU-01 Module circuit number 8 |
| MPU-01 #9 Status | SW01427 | Status of MPU-01 Module circuit number 9 |
| MPU-01 #9 Error Status | SW01428 | Error Status of MPU-01 Module circuit number 9 |
| MPU-01 #10 Status | SW01429 | Status of MPU-01 Module circuit number 10 |
| MPU-01 #10 Error Status | SW01430 | Error Status of MPU-01 Module circuit number 10 |
| MPU-01 #11 Status | SW01431 | Status of MPU-01 Module circuit number 11 |
| MPU-01 #11 Error Status | SW01432 | Error status of MPU-01 Module circuit number 11 |
| MPU-01 #12 Status | SW01433 | Status of MPU-01 Module circuit number 12 |
| MPU-01 #12 Error Status | SW01434 | Error status of MPU-01 Module circuit number 12 |
| MPU-01 #13 Status | SW01435 | Status of MPU-01 Module circuit number 13 |
| MPU-01 #13 Error Status | SW01436 | Error status of MPU-01 Module circuit number 13 |
| MPU-01 #14 Status | SW01437 | Status of MPU-01 Module circuit number 14 |
| MPU-01 #14 Error Status | SW01438 | Error status of MPU-01 Module circuit number 14 |
| MPU-01 #15 Status | SW01439 | Status of MPU-01 Module circuit number 15 |
| MPU-01 #15 Error Status | SW01440 | Error status of MPU-01 Module circuit number 15 |
| MPU-01 #16 Status | SW01441 | Status of MPU-01 Module circuit number 16 |
| MPU-01 #16 Error Status | SW01442 | Error status of MPU-01 Module circuit number 16 |

# A.12 Motion Program Information

## ( 1 ) System Work Numbers 1 to 8

| System Work Number | | System Work 1 | System Work 2 | System Work 3 | System Work 4 | System Work 5 | System Work 6 | System Work 7 | System Work 8 |
|---|---|---|---|---|---|---|---|---|---|
| Executing Main Program No. | | SW03200 | SW03201 | SW03202 | SW03203 | SW03204 | SW03205 | SW03206 | SW03207 |
| Status | | SW03264 | SW03322 | SW03380 | SW03438 | SW03496 | SW03554 | SW03612 | SW03670 |
| Control Signals | | SW03265 | SW03323 | SW03381 | SW03439 | SW03497 | SW03555 | SW03613 | SW03671 |
| Fork 0 | Program Number | SW03266 | SW03324 | SW03382 | SW03440 | SW03498 | SW03556 | SW03614 | SW03672 |
| | Block Number | SW03267 | SW03325 | SW03383 | SW03441 | SW03499 | SW03557 | SW03615 | SW03673 |
| | Alarm Code | SW03268 | SW03326 | SW03384 | SW03442 | SW03500 | SW03558 | SW03616 | SW03674 |
| Fork 1 | Program Number | SW03269 | SW03327 | SW03385 | SW03443 | SW03501 | SW03559 | SW03617 | SW03675 |
| | Block Number | SW03270 | SW03328 | SW03386 | SW03444 | SW03502 | SW03560 | SW03618 | SW03676 |
| | Alarm Code | SW03271 | SW03329 | SW03387 | SW03445 | SW03503 | SW03561 | SW03619 | SW03677 |
| Fork 2 | Program Number | SW03272 | SW03330 | SW03388 | SW03446 | SW03504 | SW03562 | SW03620 | SW03678 |
| | Block Number | SW03273 | SW03331 | SW03389 | SW03447 | SW03505 | SW03563 | SW03621 | SW03679 |
| | Alarm Code | SW03274 | SW03332 | SW03390 | SW03448 | SW03506 | SW03564 | SW03622 | SW03680 |
| Fork 3 | Program Number | SW03275 | SW03333 | SW03391 | SW03449 | SW03507 | SW03565 | SW03623 | SW03681 |
| | Block Number | SW03276 | SW03334 | SW03392 | SW03450 | SW03508 | SW03566 | SW03624 | SW03682 |
| | Alarm Code | SW03277 | SW03335 | SW03393 | SW03451 | SW03509 | SW03567 | SW03625 | SW03683 |
| Fork 4 | Program Number | SW03278 | SW03336 | SW03394 | SW03452 | SW03510 | SW03568 | SW03626 | SW03684 |
| | Block Number | SW03279 | SW03337 | SW03395 | SW03453 | SW03511 | SW03569 | SW03627 | SW03685 |
| | Alarm Code | SW03280 | SW03338 | SW03396 | SW03454 | SW03512 | SW03570 | SW03628 | SW03686 |
| Fork 5 | Program Number | SW03281 | SW03339 | SW03397 | SW03455 | SW03513 | SW03571 | SW03629 | SW03687 |
| | Block Number | SW03282 | SW03340 | SW03398 | SW03456 | SW03514 | SW03572 | SW03630 | SW03688 |
| | Alarm Code | SW03283 | SW03341 | SW03399 | SW03457 | SW03515 | SW03573 | SW03631 | SW03689 |
| Fork 6 | Program Number | SW03284 | SW03342 | SW03400 | SW03458 | SW03516 | SW03574 | SW03632 | SW03690 |
| | Block Number | SW03285 | SW03343 | SW03401 | SW03459 | SW03517 | SW03575 | SW03633 | SW03691 |
| | Alarm Code | SW03286 | SW03344 | SW03402 | SW03460 | SW03518 | SW03576 | SW03634 | SW03692 |
| Fork 7 | Program Number | SW03287 | SW03345 | SW03403 | SW03461 | SW03519 | SW03577 | SW03635 | SW03693 |
| | Block Number | SW03288 | SW03346 | SW03404 | SW03462 | SW03520 | SW03578 | SW03636 | SW03694 |
| | Alarm Code | SW03289 | SW03347 | SW03405 | SW03463 | SW03521 | SW03579 | SW03637 | SW03695 |
| Logical Axis 1 Program Current Position | | SL03290 | SL03348 | SL03406 | SL03464 | SL03522 | SL03580 | SL03638 | SL03696 |
| Logical Axis 2 Program Current Position | | SL03292 | SL03350 | SL03408 | SL03466 | SL03524 | SL03582 | SL03640 | SL03698 |
| Logical Axis 3 Program Current Position | | SL03294 | SL03352 | SL03410 | SL03468 | SL03526 | SL03584 | SL03642 | SL03700 |
| Logical Axis 4 Program Current Position | | SL03296 | SL03354 | SL03412 | SL03470 | SL03528 | SL03586 | SL03644 | SL03702 |
| Logical Axis 5 Program Current Position | | SL03298 | SL03356 | SL03414 | SL03472 | SL03530 | SL03588 | SL03646 | SL03704 |
| Logical Axis 6 Program Current Position | | SL03300 | SL03358 | SL03416 | SL03474 | SL03532 | SL03590 | SL03648 | SL03706 |
| Logical Axis 7 Program Current Position | | SL03302 | SL03360 | SL03418 | SL03476 | SL03534 | SL03592 | SL03650 | SL03708 |
| Logical Axis 8 Program Current Position | | SL03304 | SL03362 | SL03420 | SL03478 | SL03536 | SL03594 | SL03652 | SL03710 |
| Logical Axis 9 Program Current Position | | SL03306 | SL03364 | SL03422 | SL03480 | SL03538 | SL03596 | SL03654 | SL03712 |
| Logical Axis 10 Program Current Position | | SL03308 | SL03366 | SL03424 | SL03482 | SL03540 | SL03598 | SL03656 | SL03714 |
| Logical Axis 11 Program Current Position | | SL03310 | SL03368 | SL03426 | SL03484 | SL03542 | SL03600 | SL03658 | SL03716 |
| Logical Axis 12 Program Current Position | | SL03312 | SL03370 | SL03428 | SL03486 | SL03544 | SL03602 | SL03660 | SL03718 |
| Logical Axis 13 Program Current Position | | SL03314 | SL03372 | SL03430 | SL03488 | SL03546 | SL03604 | SL03662 | SL03720 |
| Logical Axis 14 Program Current Position | | SL03316 | SL03374 | SL03432 | SL03490 | SL03548 | SL03606 | SL03664 | SL03722 |
| Logical Axis 15 Program Current Position | | SL03318 | SL03376 | SL03434 | SL03492 | SL03550 | SL03608 | SL03666 | SL03724 |
| Logical Axis 16 Program Current Position | | SL03320 | SL03378 | SL03436 | SL03494 | SL03552 | SL03610 | SL03668 | SL03726 |

## ( 2 ) System Work Numbers 9 to 16

| System Work Number | | System Work 9 | System Work 10 | System Work 11 | System Work 12 | System Work 13 | System Work 14 | System Work 15 | System Work 16 |
|---|---|---|---|---|---|---|---|---|---|
| Executing Main Program No. | | SW03208 | SW03209 | SW03210 | SW03211 | SW03212 | SW03213 | SW03214 | SW03215 |
| Status | | SW03728 | SW03786 | SW03844 | SW03902 | SW03960 | SW04018 | SW04076 | SW04134 |
| Control Signals | | SW03729 | SW03787 | SW03845 | SW03903 | SW03961 | SW04019 | SW04077 | SW04135 |
| Fork 0 | Program Number | SW03730 | SW03788 | SW03846 | SW03904 | SW03962 | SW04020 | SW04078 | SW04136 |
| | Block Number | SW03731 | SW03789 | SW03847 | SW03905 | SW03963 | SW04021 | SW04079 | SW04137 |
| | Alarm Code | SW03732 | SW03790 | SW03848 | SW03906 | SW03964 | SW04022 | SW04080 | SW04138 |
| Fork 1 | Program Number | SW03733 | SW03791 | SW03849 | SW03907 | SW03965 | SW04023 | SW04081 | SW04139 |
| | Block Number | SW03734 | SW03792 | SW03850 | SW03908 | SW03966 | SW04024 | SW04082 | SW04140 |
| | Alarm Code | SW03735 | SW03793 | SW03851 | SW03909 | SW03967 | SW04025 | SW04083 | SW04141 |
| Fork 2 | Program Number | SW03736 | SW03794 | SW03852 | SW03910 | SW03968 | SW04026 | SW04084 | SW04142 |
| | Block Number | SW03737 | SW03795 | SW03853 | SW03911 | SW03969 | SW04027 | SW04085 | SW04143 |
| | Alarm Code | SW03738 | SW03796 | SW03854 | SW03912 | SW03970 | SW04028 | SW04086 | SW04144 |
| Fork 3 | Program Number | SW03739 | SW03797 | SW03855 | SW03913 | SW03971 | SW04029 | SW04087 | SW04145 |
| | Block Number | SW03740 | SW03798 | SW03856 | SW03914 | SW03972 | SW04030 | SW04088 | SW04146 |
| | Alarm Code | SW03741 | SW03799 | SW03857 | SW03915 | SW03973 | SW04031 | SW04089 | SW04147 |
| Fork 4 | Program Number | SW03742 | SW03800 | SW03858 | SW03916 | SW03974 | SW04032 | SW04090 | SW04148 |
| | Block Number | SW03743 | SW03801 | SW03859 | SW03917 | SW03975 | SW04033 | SW04091 | SW04149 |
| | Alarm Code | SW03744 | SW03802 | SW03860 | SW03918 | SW03976 | SW04034 | SW04092 | SW04150 |
| Fork 5 | Program Number | SW03745 | SW03803 | SW03861 | SW03919 | SW03977 | SW04035 | SW04093 | SW04151 |
| | Block Number | SW03746 | SW03804 | SW03862 | SW03920 | SW03978 | SW04036 | SW04094 | SW04152 |
| | Alarm Code | SW03747 | SW03805 | SW03863 | SW03921 | SW03979 | SW04037 | SW04095 | SW04153 |
| Fork 6 | Program Number | SW03748 | SW03806 | SW03864 | SW03922 | SW03980 | SW04038 | SW04096 | SW04154 |
| | Block Number | SW03749 | SW03807 | SW03865 | SW03923 | SW03981 | SW04039 | SW04097 | SW04155 |
| | Alarm Code | SW03750 | SW03808 | SW03866 | SW03924 | SW03982 | SW04040 | SW04098 | SW04156 |
| Fork 7 | Program Number | SW03751 | SW03809 | SW03867 | SW03925 | SW03983 | SW04041 | SW04099 | SW04157 |
| | Block Number | SW03752 | SW03810 | SW03868 | SW03926 | SW03984 | SW04042 | SW04100 | SW04158 |
| | Alarm Code | SW03753 | SW03811 | SW03869 | SW03927 | SW03985 | SW04043 | SW04101 | SW04159 |
| Logical Axis 1 Program Current Position | | SL03754 | SL03812 | SL03870 | SL03928 | SL03986 | SL04044 | SL04102 | SL04160 |
| Logical Axis 2 Program Current Position | | SL03756 | SL03814 | SL03872 | SL03930 | SL03988 | SL04046 | SL04104 | SL04162 |
| Logical Axis 3 Program Current Position | | SL03758 | SL03816 | SL03874 | SL03932 | SL03990 | SL04048 | SL04106 | SL04164 |
| Logical Axis 4 Program Current Position | | SL03760 | SL03818 | SL03876 | SL03934 | SL03992 | SL04050 | SL04108 | SL04166 |
| Logical Axis 5 Program Current Position | | SL03762 | SL03820 | SL03878 | SL03936 | SL03994 | SL04052 | SL04110 | SL04168 |
| Logical Axis 6 Program Current Position | | SL03764 | SL03822 | SL03880 | SL03938 | SL03996 | SL04054 | SL04112 | SL04170 |
| Logical Axis 7 Program Current Position | | SL03766 | SL03824 | SL03882 | SL03940 | SL03998 | SL04056 | SL04114 | SL04172 |
| Logical Axis 8 Program Current Position | | SL03768 | SL03826 | SL03884 | SL03942 | SL04000 | SL04058 | SL04116 | SL04174 |
| Logical Axis 9 Program Current Position | | SL03770 | SL03828 | SL03886 | SL03944 | SL04002 | SL04060 | SL04118 | SL04176 |
| Logical Axis 10 Program Current Position | | SL03772 | SL03830 | SL03888 | SL03946 | SL04004 | SL04062 | SL04120 | SL04178 |
| Logical Axis 11 Program Current Position | | SL03774 | SL03832 | SL03890 | SL03948 | SL04006 | SL04064 | SL04122 | SL04180 |
| Logical Axis 12 Program Current Position | | SL03776 | SL03834 | SL03892 | SL03950 | SL04008 | SL04066 | SL04124 | SL04182 |
| Logical Axis 13 Program Current Position | | SL03778 | SL03836 | SL03894 | SL03952 | SL04010 | SL04068 | SL04126 | SL04184 |
| Logical Axis 14 Program Current Position | | SL03780 | SL03838 | SL03896 | SL03954 | SL04012 | SL04070 | SL04128 | SL04186 |
| Logical Axis 15 Program Current Position | | SL03782 | SL03840 | SL03898 | SL03956 | SL04014 | SL04072 | SL04130 | SL04188 |
| Logical Axis 16 Program Current Position | | SL03784 | SL03842 | SL03900 | SL03958 | SL04016 | SL04074 | SL04132 | SL04190 |

# Appendix B

## CP (Previous) Ladder Instructions and New Ladder Instructions

This appendix describes some CP (previous) ladder instructions and new ladder instructions.

B.1 Correspondence between CP (Previous) Ladder Instructions and New Ladder Instructions - - - - - - - - - - - - - - - - - - B-2

B.2 Converting CP (Previous) Ladder Programs to New Ladder Programs - - B-3

# B.1 Correspondence between CP (Previous) Ladder Instructions and New Ladder Instructions

Changing from CP ladder programs to new ladder programs involves changes to some instructions and the addition of new instructions.

This section tells you what to do in new ladder programs for instructions that can be used only in CP ladder programs.

It also provides a list of instructions that can be used only in new ladder instructions.

## ( 1 ) Handling Instructions Supported Only by CP (Previous) Ladder Programs in New Ladder Programs

| Instruction Name | Function Outline | Procedure in New Ladder Programs |
|---|---|---|
| Instructions in [ ] Brackets | Instructions in [ ] brackets are executed only when the value of the B register is ON. | Use the IF instruction. |
| IFON instruction | Processing up to the IEND instruction is executed only when the value of the B register is ON. | |
| IFOFF instruction | Processing up to the IEND instruction is executed only when the value of the B register is OFF. | |
| Call User Function instruction | A user function is called. | Use the FUNC instruction. |
| Function Input instruction | The input data is stored in the function input register. | |
| Function Output instruction | The data in the function output register is stored in the specified register. | |
| Comment instruction | Text with double quotation marks (“ ”) is treated as a comment. | Use rung comments. |
| Integer Replacement instruction | Data is replaced in an A register and the integer operation is started. | Use the STORE instruction. |
| Real Number Replacement instruction | Data is replaced in an F register and the real number operation is started. | |
| Store instruction | The contents of the A or F register is stored in the specified register. | |

## ( 2 ) Instructions That You Can Use Only in New Ladder Programs

| Instruction Name | Function Outline |
|---|---|
| IF instruction | The programming between the IF and END_IF instructions is executed while the conditional expression for the IF instruction is satisfied. |
| FUNC instruction | A user function is called. |
| STORE instruction | Integer, double-length integer, or real number data is stored in a register. |
| EXPRESSION instruction | A numeric expression is written. |
| MLINK-SVW instruction | The specified SERVOPACK parameter is written. |
| MOTREG-W instruction | The specified motion register is written. |
| MOTREG-R instruction | The specified motion register is read. |

# B.2 Converting CP (Previous) Ladder Programs to New Ladder Programs

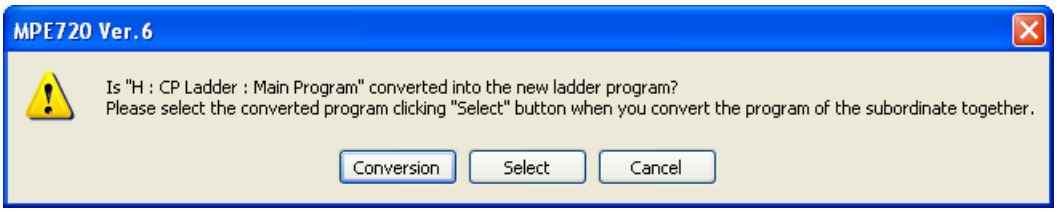
You can use the CP ladder program conversion function on MPE720 version 6 to convert CP ladder programs to new ladder programs. When converting a program, DWG properties and comments in the program will be converted at the same time.

You must be offline to convert CP ladder programs. (You cannot convert CP ladder programs while connected to the Machine Controller.) Refer to *3.6 Converting CP Ladder Programs to Ordinary Ladder Programs* in the *Engineering Tool for MP2000 Series Machine Controller MPE720 Version 6 User's Manual* (SIEP C880700 30) on converting CP ladder programs.

## ■ Procedure to Convert CP Ladder Programs

**1.** Select a program folder (High-speed, Low-speed, Start, Interrupt, or Function) or select a program that contains programs of the lower hierarchical levels. Then, right-click the selected folder or program and select ***Conversion of CP ladder*** from the pop-up menu.

When the selected program contains programs of lower hierarchical levels, the following message will appear asking for confirmation.




**Conversion**: Click this button to convert the current program and all lower level programs to new ladder programs.

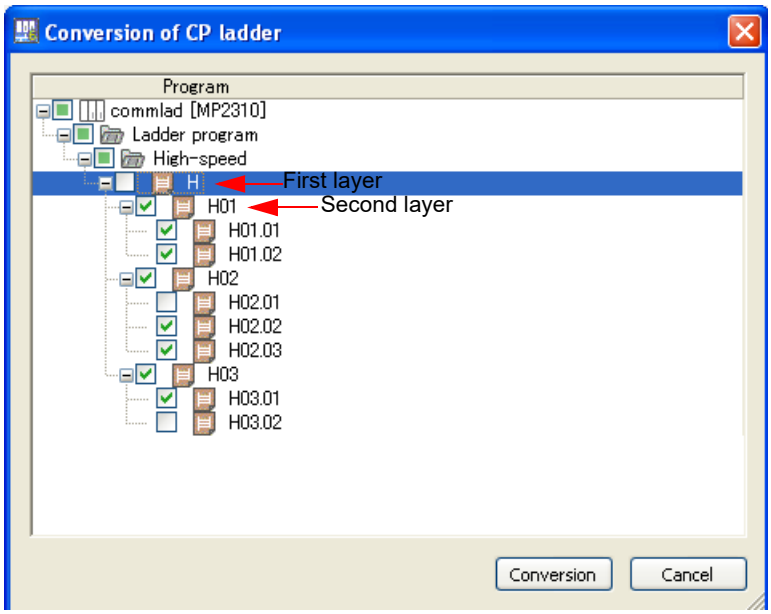
**Select**: Click this button to display the **Conversion of CP ladder** Dialog Box. See step 2 for details on setting.

**Cancel**: Click this button to cancel program conversion.

**2.** Click **Select**.

The **Conversion of CP ladder** Dialog Box will appear.
The check box of the CP ladder program specified in step 1 or the check boxes of the CP ladder programs that are displayed under the ladder program specified in step 1 will be selected.
When a program from the second hierarchical level is specified in step 1, the check box of the programs of first hierarchical level will also be selected.

- After deselecting the check boxes of the CP ladder programs not to be converted in step 3, the check boxes of the project file, ladder program folder, High-speed, Low-speed, Start, Interrupt, or Function folder to which non-selected CP ladder programs belong will be shaded. This is because some of CP ladder programs in the file or folder are selected and the rest are not selected.




**3.** Clear the check boxes of CP ladder programs not to be converted.

- New Ladder programs are shaded, and cannot be selected.

**4.** Click ***Conversion***. All the selected CP ladder programs will be converted to new ladder programs.

If you click the **Cancel** Button, the CP ladder program conversion will be cancelled.

An error code (0xAxxxxxxx) and an error name may be displayed in the Output Subwindow in accordance with the changed program.
Select the error code (0xAxxxxxxx) and press the **F1** Key. ***Error Generating Information*** will appear.
Check the error causes and take corrective action.

# Appendix C

## Sample Programming

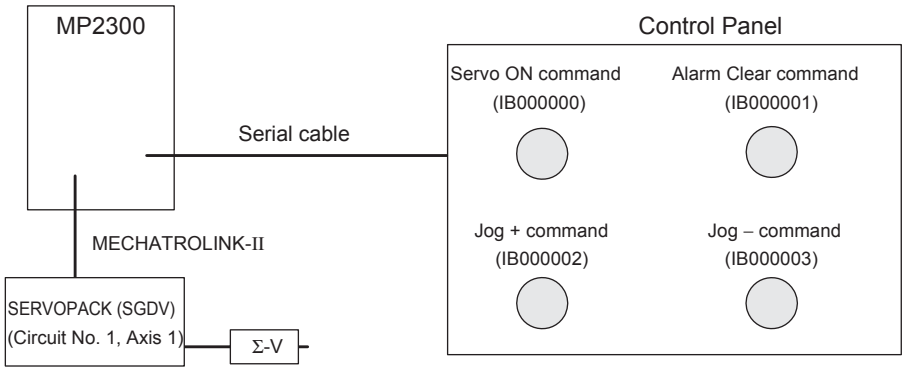
This appendix describes ladder programming examples that perform test runs.

C.1 Jogging from the Control Panel ---------- C-2
C.2 Motion Program Control ---------- C-3
C.3 Simple Synchronized Operation of Two Axes with a Virtual Axis ------ C-4
C.4 Transferring Project Files to Different Models ---------- C-6

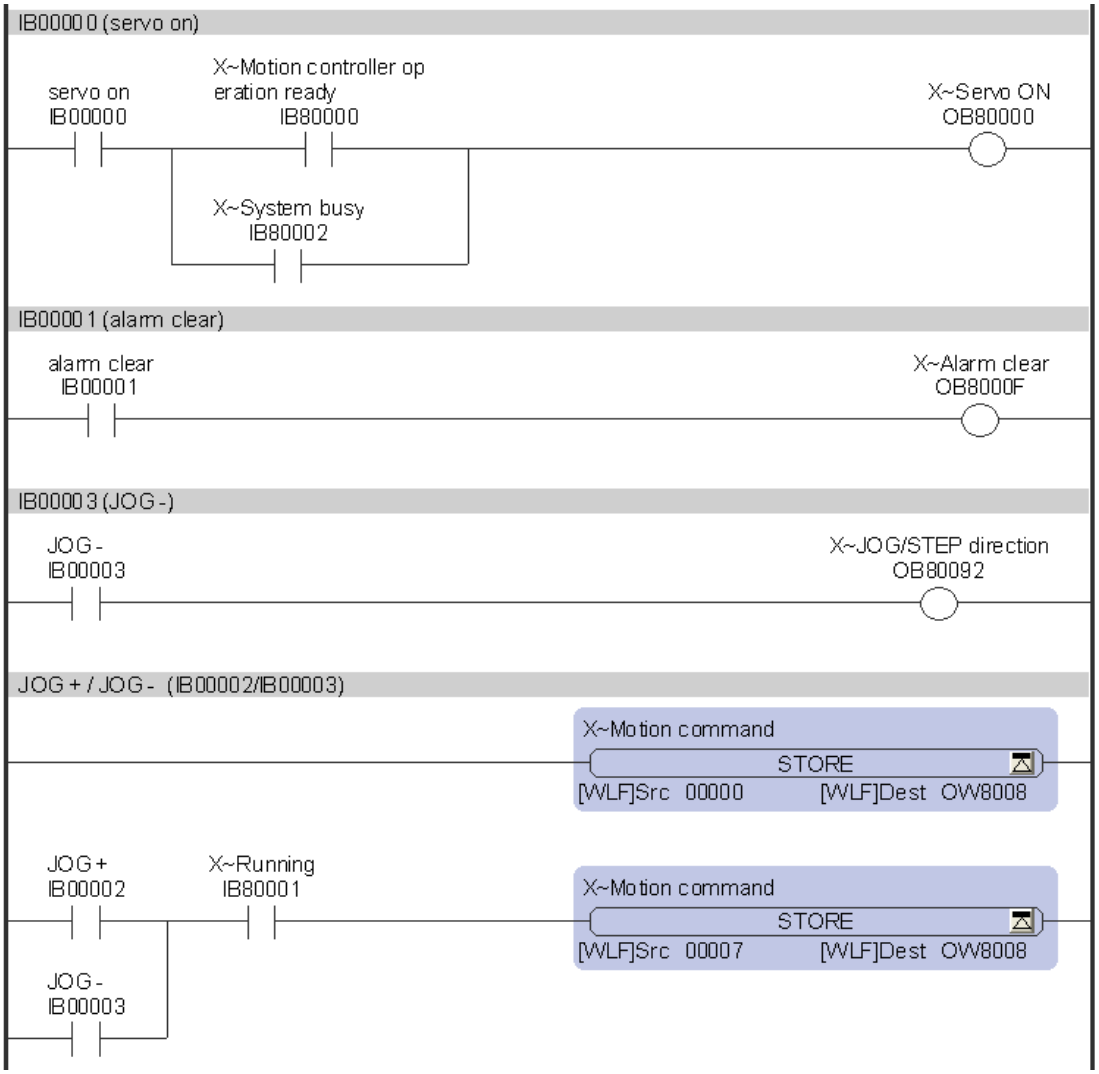
# C.1 Jogging from the Control Panel

The following configuration and ladder programming example illustrate how to control a motor from switches on a control panel when the motor and control panel are connected to the MP2300.

## ■ Configuration Example



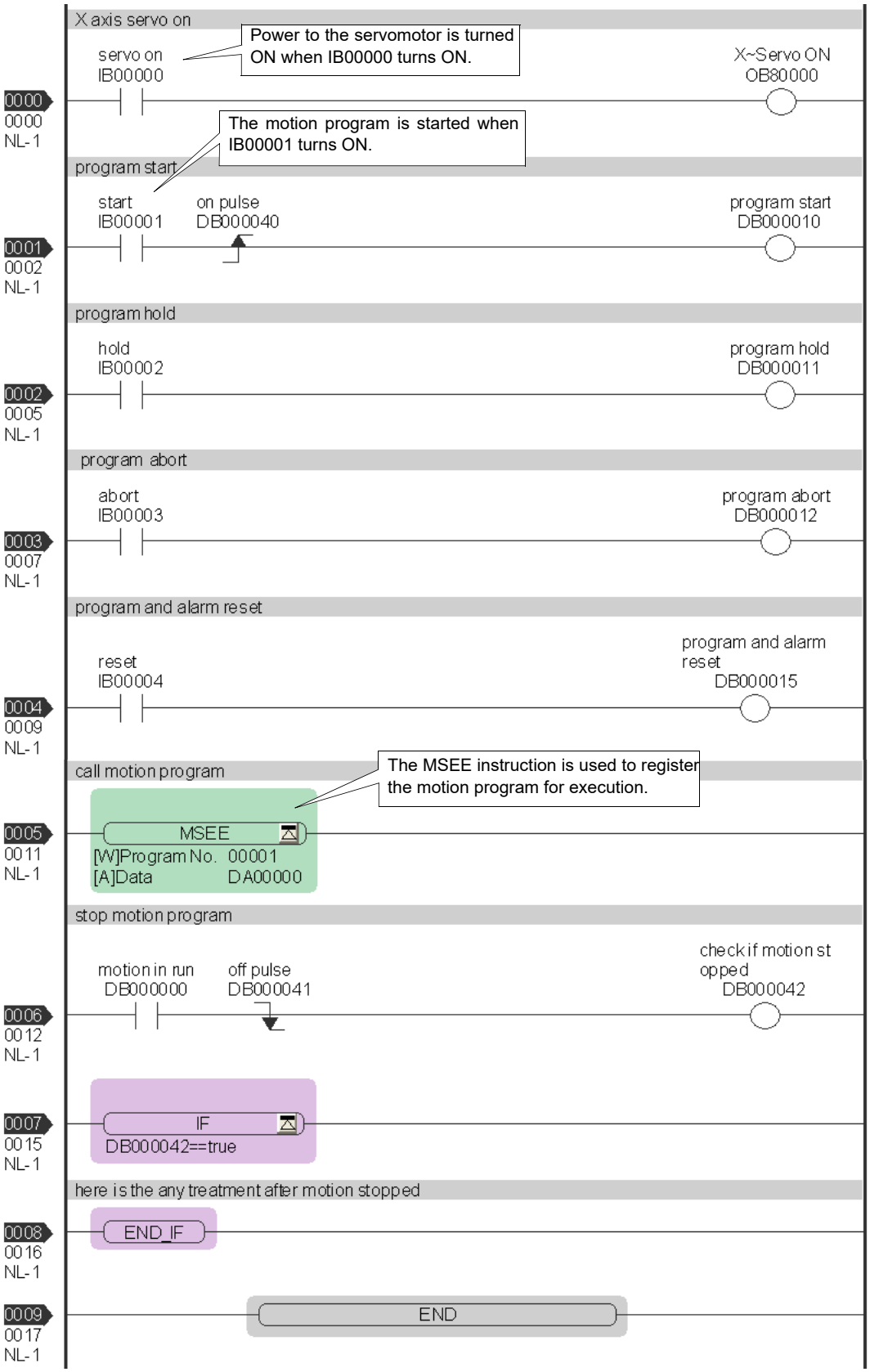

## ■ Ladder Programming Example



# C.2 Motion Program Control

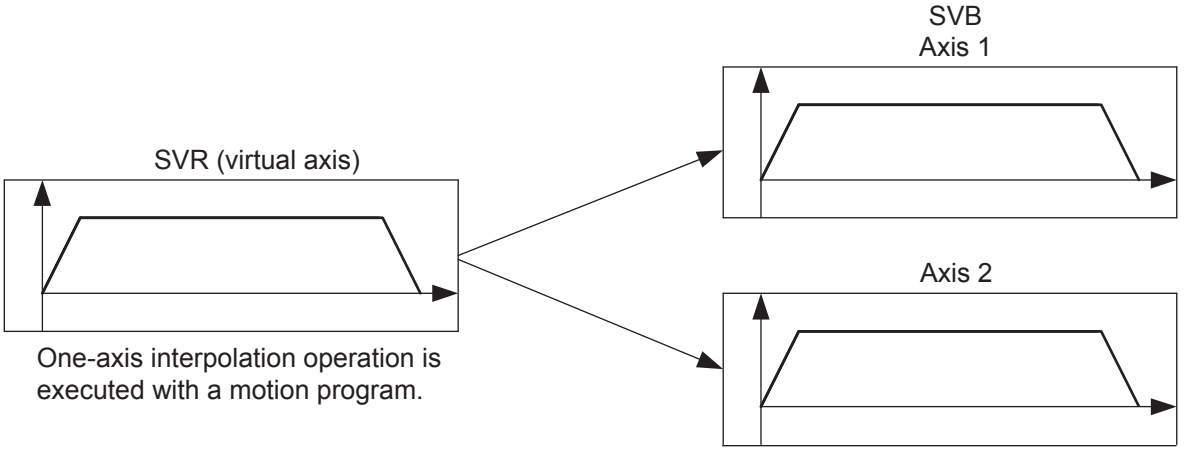
The following ladder programming example demonstrates how to control execution of a motion program.

## ■ Ladder Programming Example



# C.3 Simple Synchronized Operation of Two Axes with a Virtual Axis

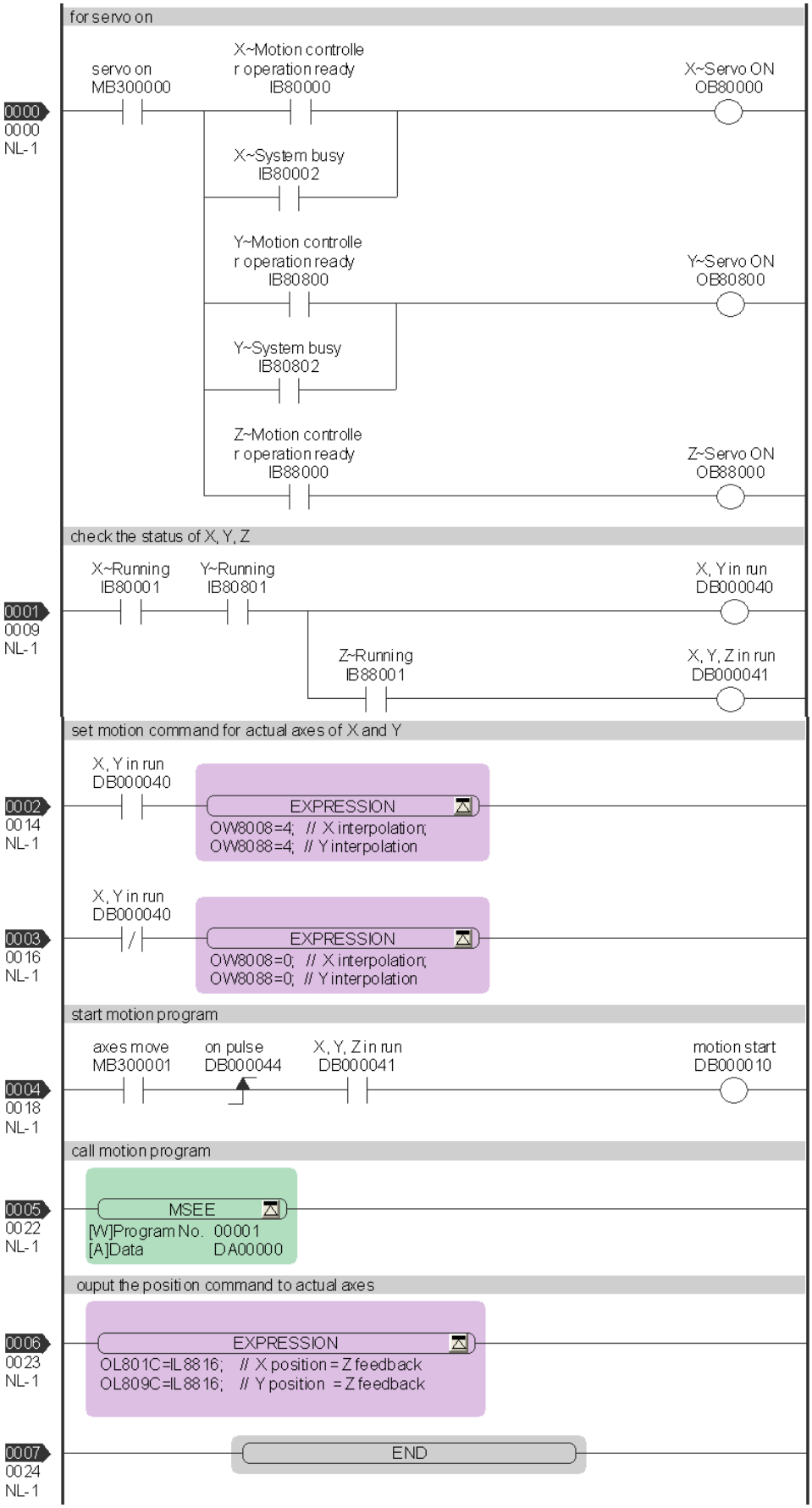
With the following sample programs, a motion program moves an SVR (virtual axis) and a ladder program distributes the feedback position of the SVR to two physical axes to perform synchronized operation with two axes.




■ Motion Programming Example

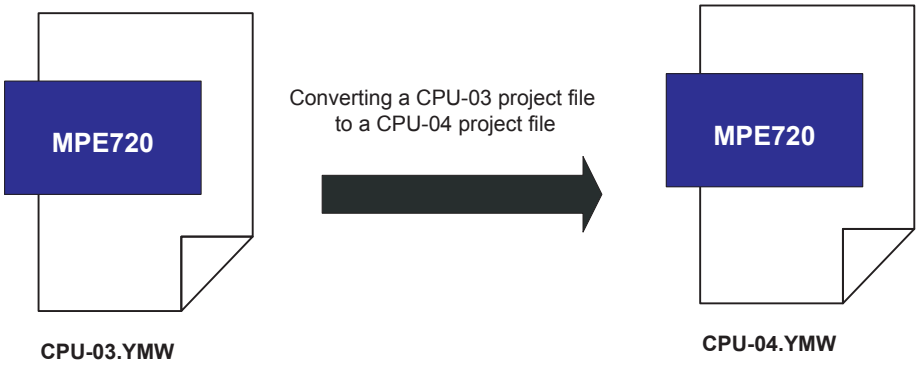
```
FMX T10000K;                  "Set maximum interpolation speed K = 1,000.
INC;                          "Incremental Mode
IAC T500;                     "Interpolation acceleration time = 500 ms
IDC T500;                     "Interpolation deceleration time = 500 ms
MVS [SVR] 1000K F10000K;      "Interpolation for travel distance of 1,000,000
END;
```

## ■ Ladder Programming Example



# C.4 Transferring Project Files to Different Models

Use the following procedure to transfer a project file to a different model.
This example shows how to convert a CPU-03 project file to a CPU-04 project file.




## ■ Procedure

1. Create a new project file for the CPU-04.
2. Select ***Online - Transfer - Read from Project.***
3. Select the CPU-03 project file and transfer it to the CPU-04 project.
4. Manually set the Module configuration definitions.

INFO

The Module configuration definition will be lost when you transfer a project file to a different model.
Set the Module configuration definitions and parameters manually.
You cannot use the axis data copy function for Module configuration definitions between different project files.

# Appendix D

## Format for EXPRESSION Instruction

This appendix describes the format for the EXPRESSION instruction.


D.1 Elements That You Can Use in Numeric Expressions ---------- D-2

D.2 National Limitations ---------- D-5

D.2.1 Arithmetic Operators ---------- D-5
D.2.2 Comparison Operators ---------- D-5
D.2.3 Logic Operators ---------- D-5
D.2.4 Substitution Operator ---------- D-6
D.2.5 Functions ---------- D-6
D.2.6 Others ---------- D-6

# D.1 Elements That You Can Use in Numeric Expressions

Numeric expressions can include operators, operands (constants and variables), and functions.
This section describes each of these elements.

## ( 1 ) Operators

### [ a ] Types of Operators and Usable Operators

The following table gives the types of operators and usable operators.

| Type | Usable Operators | |
|---|---|---|
| Arithmetic and Logic Operators | + | Add |
| | − | Subtract |
| | * | Multiply |
| | / | Divide |
| | % | Remainder |
| | & | Bit-wise AND |
| | \| | Bit-wise OR |
| Logic Operators (Usable only with bit data) | && | Inclusive AND |
| | \|\| | Inclusive OR |
| | ! | Logical NOT |
| Comparison Operators | == | Equal to right-side value |
| | != | Unequal to right-side value |
| | > | Greater than right-side value |
| | >= | Greater than or equal to right-side value |
| | < | Less than right-side value |
| | <= | Less than or equal to right-side value |
| Substitution Operator | = | Substitutes left-side value with right-side value |
| Reserved Words | true | TRUE for a logical expression |
| | false | FALSE for a logical expression |

### [ b ] Order of Evaluation

Operators are evaluated according to their processing priority and the order in which operands are grouped, as listed below.

| Priority* | Operators | Description | Grouping Order |
|---|---|---|---|
| High | [] () | Expression | Left to right |
| ↑ | – ! | Unary | Right to left |
| | * / % | Multiplication, division, and remainder | Left to right |
| | + – | Addition and subtraction | |
| | < > <= >= | Relational | |
| | == != | Equivalence | |
| | & | Bit-wise AND | |
| ↓ | \| | Bit-wise OR | |
| | && | Inclusive AND | |
| Low | \|\| | Inclusive OR | |

- Operators on the same line have the same processing priority and are evaluated according to their grouping order.

## ( 2 ) Operands

### [ a ] Constants

Integers or real numbers may be used as a constant.

- An integer may be any number that can be expressed within the range of a 32-bit integer.
  (-2,147,483,648 to 2,147,483,647)
- A real number may be any number that can be expressed within the range of 32-bit floating point data.
  ± (1.175494351e–38F to 3.402823466e+38F)

INFO

Hexadecimal numbers must be expressed using the 0x□□□□ notation when used in the EXPRESSION, IF, or WHILE instruction.

The H□□□□ notation will result in an error.

Example: H012F ... NG  0x012F ... OK

The H□□□□ notation must be used for all other instructions, such as the STORE instruction.

### [ b ] Variables

The EXPRESSION instruction allows you to assign arbitrary variable names that are allowed in C language to registers in the Machine Controller.
Although the C language does not have Boolean variables, bit registers in the Machine Controller are treated as Boolean variables. Boolean variables are either TRUE or FALSE and can be used only in logical expressions.

#### ■ Limitations on Variable Names

The following limitations apply to variable names.

- Variable names must start with a non-numeric character.
- For ASCII characters, only alphabetic characters, underscores, and numbers may be used.
- The following variable names cannot be used because they are already used as function names.

◀ EXAMPLE ▶

| | |
|---|---|
| Abc | OK |
| Get_input() | OK |
| 1ab | NG |
| Sin | NG |

## ( 3 ) Instructions That You Can Use with EXPRESSION Instructions

| Instruction | Description | Example | Reserved Word |
|---|---|---|---|
| + | Add | MW00001 = MW00002 + MW00003 | √ |
| – | Subtract | MW00001 = MW00002 – MW00003 | √ |
| * | Multiply | MW00001 = MW00002 × MW00003 | √ |
| / | Divide | MW00001 = MW00002 / MW00003 | √ |
| % | Remainder | MW00001 = MW00002 % MW00003 | √ |
| & | Bit-wise AND | MW00001 = MW00002 & 4096 | √ |
| \| | Bit-wise OR | MW00001 = MW00002 \| 4096 | √ |
| && | Inclusive AND | MB000010 = MB000011 && MB000012 | √ |
| // | Inclusive OR | MB000010 = MB000011 // MB000012 | √ |
| ! | Logical NOT | MB000010 = !MB000011 | √ |
| == | Equal to right-side value | MB000010 = MB000011 == true | √ |
| >= | Right-side value is less than or equal to left-side value | MB000010 = MW00020 >= MW00021 | √ |
| > | Right-side value is less than left-side value | MB000010 = MW00020 > MW00021 | √ |
| < | Right-side value is greater than left-side value | MB000010 = MW00020 < MW00021 | √ |
| <= | Right-side value is greater than or equal to left-side value | MB000010 = MW00020 <= MW00021 | √ |
| = | Substitute left-side value with right-side value | MW00001 = MW00002 | √ |
| true | TRUE | MB000010 = MB000011 == true | √ |
| false | FALSE | MB000010 = MB000011 == false | √ |
| sin() | SIN | MW00001 = sin(MW00002) | √ |
| cos() | COS | MF00002 = cos(MF00004) | √ |
| atan() | ARCTAN | MF00002 = atan(MF00004) | √ |
| tan() | TAN | MF00002 = tan(MF00004) | √ |
| () | Parentheses | MW00001 = (MW00002 + MW00003) / MW00004 | √ |
| asin() | ARCSIN | MF00002 = asin(MF00004) | √ |
| acos() | ARCCOS | MF00002 = acos(MF00004) | √ |
| sqrt() | AQRT | MW00001 = sqrt(MW00002) | √ |
| abs() | ABS | MW00001 = abs(MW00002) | √ |
| exp() | EXP | MF00002 = exp(MF00004) | √ |
| log() | LOG natural logarithm | MF00002 = log(MF00004) | √ |
| log10() | LOG10 common logarithm | MF00002 = log10(MF00004) | √ |
| IF ELSE construct | IF condition judgment | IF MW00000 == MW00001;<br>ELSE;<br>IEND;<br>(ELSE can be omitted) | √ |
| WHILE construct | WHILE repeat | WHILE MW00000 == MW00001;<br>WEND; | √ |

IF-ELSE and WHILE constructs may result in a compile error in MPE720 version 6.
These constructs can be used with MPE720 version 7.

# D.2 National Limitations

Several limitations apply when combining operands and operators to form numeric expressions. An expression is not recognized as a numeric expression unless it meets these conditions.
This section describes these limitations.

## D.2.1 Arithmetic Operators

These operators can be used with integer and real number operands. The unary minus operator can be used only once. Bit operations can be performed only on integer data. Bit operands cannot be used for arithmetic operations. No automatic data type conversion is performed even if the calculation result exceeds the range of the assigned register. Therefore, the user must assign the appropriate data type to the variable.

◀ EXAMPLE ▶

| | |
|---|---|
| MW00001 = MW00002 + MW00003 | OK |
| MW00001 = MW00002 / 345 | OK |
| MF00002 = (MW00004 + MF00002) / (ML00018 + MW00008) | OK |
| MW00001 = MW00002 & 4096 | OK |
| MB000010 = MB000011 - MB000012 | NG |
| MW00001 = MB000011 * MW00001 | NG |

## D.2.2 Comparison Operators

These operators can be used with integer and real number operands. The left side must be a bit data register. To use an integer bit operand in a comparison operation with the == or != operator, compare it with TRUE or FALSE.

◀ EXAMPLE ▶

| | |
|---|---|
| MB000010 = MW00002 != MW00003 | OK |
| MB000010 = MF00002 < 99.99 | OK |
| MB000010 = MW00002 >= MW00003 | OK |
| MB000010 = MB000011 == true | OK |
| MB000010 = MB000011 != 0 | NG |
| MB000010 = MB000011 == 1 | NG |

## D.2.3 Logic Operators

These operators can be used with bit operands.

◀ EXAMPLE ▶

| | |
|---|---|
| MB000010 = MB000011 && MB000012 | OK |
| MB000010 = !MB000011 | OK |
| MB000010 = (MW000020 >= 50) && MB000011 | OK |
| MB000010 = MW00001 \|\| MW00002 | NG |
| MB000010 = !MW00001 | NG |

## D.2.4 Substitution Operator

Real number and integer registers can be substituted with either real number or integer data, even if the data type differs on the right and left sides. When you substitute an integer with a real number, a round-off error will occur.
Bit registers can be substituted only with logical values, such as another bit register or a TRUE/FALSE. If you substitute a bit register with a non-logical value, that value will be compared against 0 or 0.0 and the TRUE or FALSE outcome will be converted to a code before it is substituted.
Bit data cannot be substituted into non-bit registers.

◀ EXAMPLE ▶

```
MW00001 = MW00002;                    OK
MF00000 = MW00002 / 345;              OK
MB000010 = MB000010;                  OK
MW00010 = MB0000101;                  NG
MW00001 = true;                       NG
```

## D.2.5 Functions

The arguments and return values for functions depend on the specifications of the functions in the Machine Controller. Therefore, if the input for the sin(), cos(), and atan() functions is an integer or integer register, the output value will be returned as an integer. If the input is a real number or a real number register, the output value will be returned as a real number.
The argument for the tan() function is a real number so an integer register input will be treated as a real number.

◀ EXAMPLE ▶

```
MW00001 = sin(MW00002);               OK
MF00002 = cos(MF00000×3.14);          OK
MW00001 = -atan(MF00002);             OK
```

## D.2.6 Others

### ■ Parentheses

You can group multiple expressions by enclosing them with parenthesis ( ).

◀ EXAMPLE ▶

```
MW00001 = -(MW00002 + 10) / (MW00003 – MW00005);    OK
```

### ■ Arrays

You can specify arrays by using square brackets [ ], just like with the C language.

◀ EXAMPLE ▶

```
MW00001 = MW00002[100];               OK
MW00001 = MW00002[MW00003];           OK
MB000010 = MB000020[0]:               OK
```

# Appendix E

## Precautions

This appendix provides precautions on ladder programs and motion parameters.

E.1 General Precautions ---------- E-2
E.2 Precautions on Motion Parameters ---------- E-2

# E.1 General Precautions

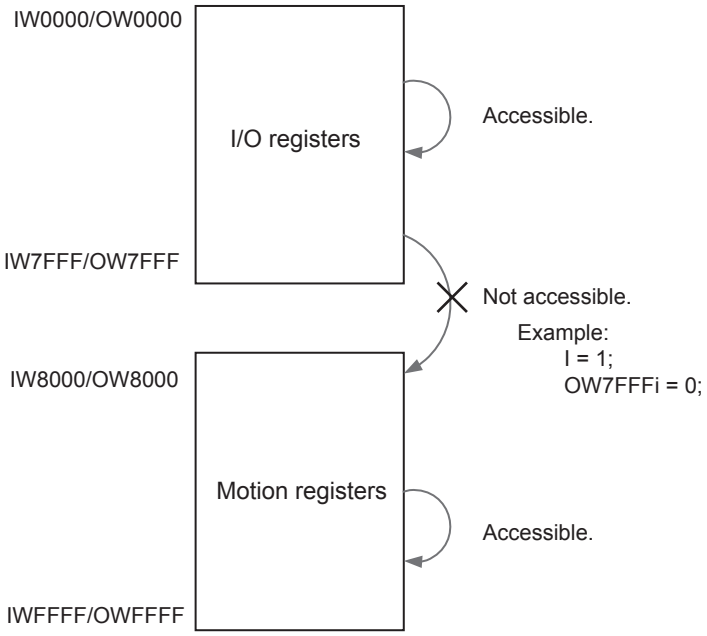
## ( 1 ) Do Not Forget to Save The Data to Flash Memory When You Change or Transfer a Program

Do not forget to save the data to flash memory when you change or transfer a ladder program or motion program. If you do not, any changes that were made to the program will be lost when the power supply to the Machine Controller is turned OFF.

# E.2 Precautions on Motion Parameters

## ( 1 ) Do Not Use a Subscript to Reference a Motion Register from an I/O Register

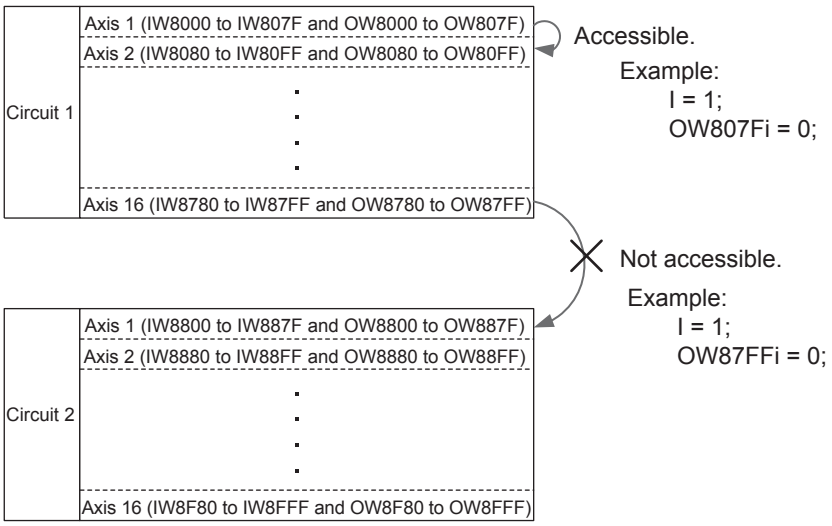
I/O registers and motion registers are not assigned to consecutive memory locations.
When using a subscript, make sure that you access registers within the range of I/O registers or within the range of motion registers.



## ( 2 ) Do Not Use a Subscript to Reference a Motion Register in a Different Circuit

Motion registers on different circuits are not assigned to continuous memory location, just as is true for I/O registers and motion registers.
When using a subscript, access registers within the range of motion registers for each circuit.
If the circuit numbers are the same, it is possible to access motion registers for different axes.

| Circuit No. | Axis 1 | Axis 2 | ... | Axis 16 |
|---|---|---|---|---|
| 1 | OW8000 to OW807F | OW8080 to OW80FF | ... | OW8780 to OW87FF |
| 2 | OW8800 to OW887F | OW8880 to OW88FF | ... | OW8F80 to OW8FFF |
| 3 | OW9000 to OW907F | OW9080 to OW90FF | ... | OW9780 to OW97FF |
| 4 | OW9800 to OW987F | OW9880 to OW98FF | ... | OW9F80 to OW9FFF |
| 5 | OWA000 to OWA07F | OWA080 to OWA0FF | ... | OWA780 to OWA7FF |
| 6 | OWA800 to OWA87F | OWA880 to OWA8FF | ... | OWAF80 to OWAFFF |
| 7 | OWB000 to OWB07F | OWB080 to OWB0FF | ... | OWB780 to OWB7FF |
| 8 | OWB800 to OWB87F | OWB880 to OWB8FF | ... | OWBF80 to OWBFFF |
| 9 | OWC000 to OWC07F | OWC080 to OWC0FF | ... | OWC780 to OWC7FF |
| 10 | OWC800 to OWC87F | OWC880 to OWC8FF | ... | OWCF80 to OWCFFF |
| 11 | OWD000 to OWD07F | OWD080 to OWD0FF | ... | OWD780 to OWD7FF |
| 12 | OWD800 to OWD87F | OWD880 to OWD8FF | ... | OWDF80 to OWDFFF |
| 13 | OWE000 to OWE07F | OWE080 to OWE0FF | ... | OWE780 to OWE7FF |
| 14 | OWE800 to OWE87F | OWE880 to OWE8FF | ... | OWEF80 to OWEFFF |
| 15 | OWF000 to OWF07F | OWF080 to OWF0FF | ... | OWF780 to OWF7FF |
| 16 | OWF800 to OWF87F | OWF880 to OWF8FF | ... | OWFF80 to OWFFFF |



# Index

## Symbols


# registers ---- 4-15


## Numerics


10-ms OFF-Delay Timer (TOFF[10ms]) ---- 5-9
10-ms ON-Delay Timer (TON[10ms]) ---- 5-7
1-s OFF-Delay Timer (TOFF[1s]) ---- 5-13
1-s ON-Delay Timer (TON[1s]) ---- 5-11


## A


Absolute Value (ABS) ---- 5-53
Add (ADD (+)) ---- 5-24
Add Time (TMADD) ---- 5-44
address ---- 4-17
alarms ---- 7-4
Arc Cosine (ACOS) ---- 5-107
Arc Sine (ASIN) ---- 5-106
Arc Tangent (ATAN) ---- 5-108
arithmetic operators ---- D-5
ASCII Conversion 1(ASCII) ---- 5-57
ASCII Conversion 2(BINASC) ---- 5-59
ASCII Conversion 3 (ASCBIN) ---- 5-61


## B


basic flow of troubleshooting ---- 7-2
BCD Conversion (BCD) ---- 5-55
Binary Conversion (BIN) ---- 5-54
Binary Search (BSRCH) ---- 5-128
Bit Rotate left (ROTL) ---- 5-112
Bit Rotate Right (ROTR) ---- 5-114
Bit Shift Left (SHFTL) ---- 5-132
Bit Shift Right (SHFTR) ---- 5-134
bits ---- 4-17
Byte Swap (BSWAP) ---- 5-138
Byte-to-word Expansion (BEXTD) ---- 5-124


## C


Call C-language Function (C-FUNC) ---- 5-281
Call Extended Program (XCALL) ---- 5-87
Call Motion Program (MSEE) ---- 5-78
Call Sequence Program (SEE) ---- 5-77
Call User Function (FUNC) ---- 5-80
calling a user function ---- 4-12
checking for multiple coils ---- 6-5
child drawings ---- 4-3
c-language programs ---- 4-26
C-language Task Control (TSK-CTRL) ---- 5-283
Clear Queue Table Pointers (QTBLCL) ---- 5-223
Clear Table Block (TBLCL) ---- 5-209
Coil (COIL) ---- 5-19
Common Logarithm (LOG) ---- 5-111
comparison operators ---- D-5
compiling for MPE720 version 5 ---- 6-9
constant registers ---- 4-14
controlling execution of drawings ---- 4-5
Copy Word (COPYW) ---- 5-136
Cosine (COS) ---- 5-103
Counter (COUNTER) ---- 5-225
cross references ---- 6-4


## D


D registers ---- 4-15
data registers ---- 4-14
data tracing ---- 4-28
data types ---- 4-17
Dead Zone A(DZA) ---- 5-139
Dead Zone B (DZB) ---- 5-141
Decrement (DEC) ---- 5-42
Direct Input String (INS) ---- 5-81
Direct Output String (OUTS) ---- 5-84
Divide (DIV (÷)) ---- 5-34
double-length integer ---- 4-17
drawing A ---- 4-3
drawing H ---- 4-3
drawing I ---- 4-3
drawing L ---- 4-3
DWG.A ---- 4-3
DWG.H ---- 4-3
DWG.I ---- 4-3
DWG.L ---- 4-3


## E


enabling and disabling ladder programs ---- 6-8
Equal (=) ---- 5-71
errors ---- 7-4
Exchange (XCHG) ---- 5-120
Exclusive OR (XOR) ---- 5-67
execution processing of drawings ---- 4-6
Exponential (EXP) ---- 5-109
Expression (EXPRESSION) ---- 5-97
Extended Add (ADDX (++)) ---- 5-26
Extended Subtract (SUBX (– –) ---- 5-30


## F


Falling-edge Pulses (OFF-PLS) ---- 5-17
First-in First-out (FINFOUT) ---- 5-228
First-order Lag (LAG) ---- 5-161
FOR Construct (FOR, END_FOR) ---- 5-91
forcing coils ON and OFF ---- 6-5
Function Generator (FGN) ---- 5-167
functional external registers ---- 4-15
functional input registers ---- 4-15
functional internal registers ---- 4-15
functional output registers ---- 4-15


## G


global registers ---- 4-13
grandchild drawings ---- 4-3
Greater Than (>) ---- 5-74
Greater Than or Equal (≥) ---- 5-73


## H


hierarchical configuration of drawings ---- 4-4


## I


I/O errors ---- 7-9
IF Construct (IF, END_IF) ---- 5-93
IF_ELSE Construct (IF, ELSE, END_IF) ---- 5-95
Inclusive AND (AND) ---- 5-63
Inclusive OR (OR) ---- 5-65
Increment (INC) ---- 5-40
index registers (i, j) ---- 4-19
indicator status ---- 7-3
input registers ---- 4-14
integer ---- 4-17

Integer Remainder (MOD) - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-36
Inverse Function Generator (IFGN) - - - - - - - - - - - - - - - - - - - - 5-172
Invert Sign (INV) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-51

## L

ladder drawings - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-3
ladder program - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1-2
Less Than (<) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-69
Less Than or Equal (≤) - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-70
Linear Accelerator/Decelerator 1 (LAU) - - - - - - - - - - - - - - - - 5-177
Linear Accelerator/Decelerator 2 (SLAU) - - - - - - - - - - - - - - - 5-184
local registers - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-13
local registers within a user function - - - - - - - - - - - - - - - - - - - 4-15
logic operators - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - D-5

## M

module synchronization errors - - - - - - - - - - - - - - - - - - - - - - - 7-10
motion programs - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-25
Move Bit (MOVB) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-116
Move Table Block (TBLMV) - - - - - - - - - - - - - - - - - - - - - - - - 5-212
Move Word (MOVW) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-118
MPE720 Version 6 Engineering Tool specifications - - - - - - - - - - - 2-4
Multiply (MUL (x)) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-32

## N

Natural Logarithm (LN) - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-110
NC Contact (NCC) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-6
NO contact (NOC) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-5
Not Equal (≠) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-72

## O

One's Complement (COM) - - - - - - - - - - - - - - - - - - - - - - - - - - 5-52
operation error drawings - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-3
operation errors - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 7-6
output registers - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-14

## P

parent drawings - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-3
Parity Conversion (PARITY) - - - - - - - - - - - - - - - - - - - - - - - 5-56
PD Control (PD) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-150
Phase Lead Lag (LLAG) - - - - - - - - - - - - - - - - - - - - - - - - - - 5-164
PI Control (PI) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-145
PID Control (PID) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-156
procedure to convert CP ladder programs - - - - - - - - - - - - - - - - - B-3
Pulse Width Modulation (PWM) - - - - - - - - - - - - - - - - - - - - - - 5-194

## R

Range Check (RCHK) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-75
Read Data Trace (DTRC-RD) - - - - - - - - - - - - - - - - - - - - - - - 5-234
Read Inverter Parameter (ICNS-RD) - - - - - - - - - - - - - - - - - - - 5-266
Read Inverter Trace (ITRC-RD) - - - - - - - - - - - - - - - - - - - - - 5-238
Read Motion Register (MOTREG-R) - - - - - - - - - - - - - - - - - - - 5-278
Read Queue Table (QTBLR and QTBLRI) - - - - - - - - - - - - - - - 5-215
Read Table Block (TBLBR) - - - - - - - - - - - - - - - - - - - - - - - - 5-197
reading data from and writing data to projects - - - - - - - - - - - - - 4-23
reading data from the Machine Controller - - - - - - - - - - - - - - - - 4-23
Real Remainder (REM) - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-38
realtime tracing - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-28
Receive Message (MSG-RCV) - - - - - - - - - - - - - - - - - - - - - - - 5-253
register lists - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 6-6
registers (variables) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-13
Reset Coil (R-COIL) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-21
Rising-edge Pulses (ON-PLS) - - - - - - - - - - - - - - - - - - - - - - - 5-15

## S

saving data to flash memory - - - - - - - - - - - - - - - - - - - - - - - - - 4-23
scheduling execution of scan process drawings - - - - - - - - - - - - - 4-5
Search for Table Column (TBLSRC) - - - - - - - - - - - - - - - - - - - 5-206
Search for Table Row (TBLSRL) - - - - - - - - - - - - - - - - - - - - - 5-203
searching and replacing in programs - - - - - - - - - - - - - - - - - - - 6-3
searching and replacing in project files - - - - - - - - - - - - - - - - - - 6-3
security - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-27
Send Message (MSG-SND) - - - - - - - - - - - - - - - - - - - - - - - - 5-241
Set Coil (S-COIL) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-20
setting high-speed/low-speed scan times - - - - - - - - - - - - - - - - - 4-24
Sine (SIN) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-101
single-precision real number - - - - - - - - - - - - - - - - - - - - - - - 4-17
Sort (SORT) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-130
Spend Time (SPEND) - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-48
Square Root (SQRT) - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-99
Store (STORE) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-22
substitute operators - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - D-6
Subtract (–) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-28
Subtract Time (TMSUB) - - - - - - - - - - - - - - - - - - - - - - - - - - 5-46
system error status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-7
system errors - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 7-11
system registers - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-14
system service registers - - - - - - - - - - - - - - - - - - - - - - - - - - - A-2
system status - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - A-6

## T

table data - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-21
Table Initialization (SETW) - - - - - - - - - - - - - - - - - - - - - - - 5-122
Tangent (TAN) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-105
Trace (TRACE) - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 5-232
transferring data - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-23
Tuning Panel - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 6-7

## U

Upper/Lower Limit (LIMIT) - - - - - - - - - - - - - - - - - - - - - - - 5-143
user functions - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-7
user operation error code -1 - - - - - - - - - - - - - - - - - - - - - - - - A-9
user operation error code -2 - - - - - - - - - - - - - - - - - - - - - - - - A-10
user operation error status - - - - - - - - - - - - - - - - - - - - - - - - - A-9

## W

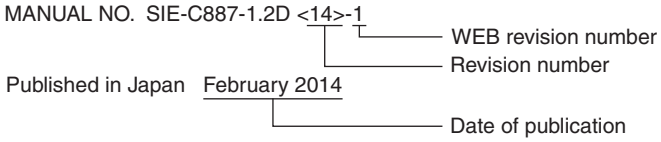
watchdog timer errors - - - - - - - - - - - - - - - - - - - - - - - - - - - - 7-10
WHILE Construct (WHILE, END_WHILE) - - - - - - - - - - - - - - - - 5-88
Word-to-byte Compression (BPRESS) - - - - - - - - - - - - - - - - - 5-126
Write Inverter Parameter (ICNS-WR) - - - - - - - - - - - - - - - - - 5-261
Write Motion Register (MOTREG-W) - - - - - - - - - - - - - - - - - 5-275
Write Queue Table (QTBLW and QTBLWI) - - - - - - - - - - - - - - 5-219
Write SERVOPACK Parameter (MLNK-SVW) - - - - - - - - - - - 5-270
Write Table Block (TBLBW) - - - - - - - - - - - - - - - - - - - - - - - 5-200
writing data to a Machine Controller - - - - - - - - - - - - - - - - - - - 4-23

## X

XY tracing - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 4-28

# Revision History

The revision dates and numbers of the revised manuals are given on the bottom of the back cover.




| Date of Publication | Rev. No. | WEB Rev. No. | Section | Revised Contents |
|---|---|---|---|---|
| June 2022 | <15> | 7 | 5.10.5, 5.10.8, 5.10.9 | Deletion: MP930, SVB-01 |
| | | | D.1 (3) | Partly revised. |
| | | | Back cover | Revision: Address |
| July 2021 | | 6 | 5.2.4, 5.2.6 | Partly revised. |
| | | | Back cover | Revision: Address |
| October 2019 | | 5 | 5.10.7 (3) | Revision: Programming Example |
| | | | Back cover | Revision: Address |
| April 2017 | | 4 | 5.8.2 | Revision: Figure of Dead Zone B |
| | | | Back cover | Revision: Address |
| October 2016 | | 3 | 5.8 | Addition: Information on the scan time set value |
| April 2016 | | 2 | 5.9.2 (1) | Revision: Table data for Write Table Block (TBLBW) |
| October 2015 | | 1 | 5.8.4 (2) (b), 5.8.5 (2) (b), 5.8.6 (2) (b) | Revision: Specifications of P gain, I gain and D gain |
| | | | 5.8.4 (3), 5.8.5 (3), 5.8.6 (3) | Revision: Programming Example |
| | | | Back cover | Revision: Address |
| May 2015 | | 0 | Front cover, spine | Revision: Format |
| | | | – | Printed version of the user's manual that is available on the web (web version: SIE-C887-1.2D <14>-1) |
| | | | Back cover | Revision: Address and format |
| February 2014 | <14> | 1 | 5.8.2 (1) | Revision: Formula for input value and output value related to dead zone |
| August 2013 | | 0 | All chapters | Completely revised. |
| | | | Back cover | Revision: Address |
| February 2013 | <13> | 0 | Back cover | Revision: Address |
| February 2012 | <12> | 0 | Back cover | Revision: Address |
| June 2011 | <11> | 0 | Front cover | Revision: Format |
| | | | Back cover | Revision: Address and format |
| December 2009 | <10> | 0 | – | Based on Japanese user's manual, SI-C887-1.2E <17> published in October 2009. |
| | | | Preface | Revision: General precautions<br>Addition: PL on fumigation and warranty |
| | | | 1.3 | Addition: Characteristics of registers in user functions |
| | | | 2.3.2 | Revision: Integer input for function input registers<br>Addition: Notes on the use of registers (X, Y, Z, and D) in functions |
| | | | 5.2 | Revision: Definition of TRACE function |
| | | | Back cover | Revision: Address |
| October 2008 | <9> | 0 | Back cover | Revision: Address |
| March 2007 | <8> | 0 | – | Based on Japanese user's manual, SI-C887-1.2D <14> published in July 2006. |
| | | | Back cover | Revision: Address |
| April 2006 | <7> | 0 | – | Based on Japanese user's manual, SI-C887-1.2D <13> published in February 2006. |
| | | | 3.3.1 | Revision: RSSEL, MDSEL, and STS designations |
| August 2005 | <6> | 0 | Back cover | Revision: Address |

| Date of Publication | Rev. No. | WEB Rev. No. | Section | Revised Contents |
|---|---|---|---|---|
| March 2005 | <5> | 0 | All chapters | Completely revised. |
| | | | Back cover | Revision: Address |
| June 2003 | <4> | 0 | Back cover | Revision: Address |
| December 2002 | <3> | 0 | Back cover | Revision: Address |
| February 2001 | <2> | 0 | All chapters | Completely revised. |
| October 1998 | <1> | 0 | All chapters | Partly revised. |
| July 1998 | – | – | – | First edition |

Machine Controller MP2000 Series

# USER'S MANUAL
# LADDER PROGRAMMING


**IRUMA BUSINESS CENTER (SOLUTION CENTER)**
480, Kamifujisawa, Iruma, Saitama, 358-8555, Japan
Phone: +81-4-2962-5151 Fax: +81-4-2962-6138
www.yaskawa.co.jp

**YASKAWA AMERICA, INC.**
2121, Norman Drive South, Waukegan, IL 60085, U.S.A.
Phone: +1-800-YASKAWA (927-5292) or +1-847-887-7000 Fax: +1-847-887-7310
www.yaskawa.com

**YASKAWA ELÉTRICO DO BRASIL LTDA.**
777, Avenida Piraporinha, Diadema, São Paulo, 09950-000, Brasil
Phone: +55-11-3585-1100 Fax: +55-11-3585-1187
www.yaskawa.com.br

**YASKAWA EUROPE GmbH**
Hauptstraße 185, 65760 Eschborn, Germany
Phone: +49-6196-569-300 Fax: +49-6196-569-398
www.yaskawa.eu.com E-mail: info@yaskawa.eu.com

**YASKAWA ELECTRIC KOREA CORPORATION**
18F, Hi Investment & Securities Building, 66 Yeoui-daero, Yeongdeungpo-gu, Seoul, 07325, Korea
Phone: +82-2-784-7844 Fax: +82-2-784-8495
www.yaskawa.co.kr

**YASKAWA ASIA PACIFIC PTE. LTD.**
30A, Kallang Place, #06-01, 339213, Singapore
Phone: +65-6282-3003 Fax: +65-6289-3003
www.yaskawa.com.sg

**YASKAWA ELECTRIC (THAILAND) CO., LTD.**
59, 1F-5F, Flourish Building, Soi Ratchadapisek 18, Ratchadapisek Road, Huaykwang, Bangkok, 10310, Thailand
Phone: +66-2-017-0099 Fax: +66-2-017-0799
www.yaskawa.co.th

**YASKAWA ELECTRIC (CHINA) CO., LTD.**
22F, Link Square 1, No.222, Hubin Road, Shanghai, 200021, China
Phone: +86-21-5385-2200 Fax: +86-21-5385-3299
www.yaskawa.com.cn

**YASKAWA ELECTRIC (CHINA) CO., LTD. BEIJING OFFICE**
Room 1011, Tower W3 Oriental Plaza, No.1, East Chang An Avenue,
Dong Cheng District, Beijing, 100738, China
Phone: +86-10-8518-4086 Fax: +86-10-8518-4082

**YASKAWA ELECTRIC TAIWAN CORPORATION**
12F, No. 207, Section 3, Beishin Road, Shindian District, New Taipei City 23143, Taiwan
Phone: +886-2-8913-1333 Fax: +886-2-8913-1513 or +886-2-8913-1519
www.yaskawa.com.tw


YASKAWA


**YASKAWA ELECTRIC CORPORATION**



In the event that the end user of this product is to be the military and said product is to be employed in any weapons systems or the manufacture thereof, the export will fall under the relevant regulations as stipulated in the Foreign Exchange and Foreign Trade Regulations. Therefore, be sure to follow all procedures and submit all relevant documentation according to any and all rules, regulations and laws that may apply.

Specifications are subject to change without notice for ongoing product modifications and improvements.

© 1998 YASKAWA ELECTRIC CORPORATION



MANUAL NO. SIE-C887-1.2E <15>-7

Published in Japan June 2022
22-4-18
Original instructions