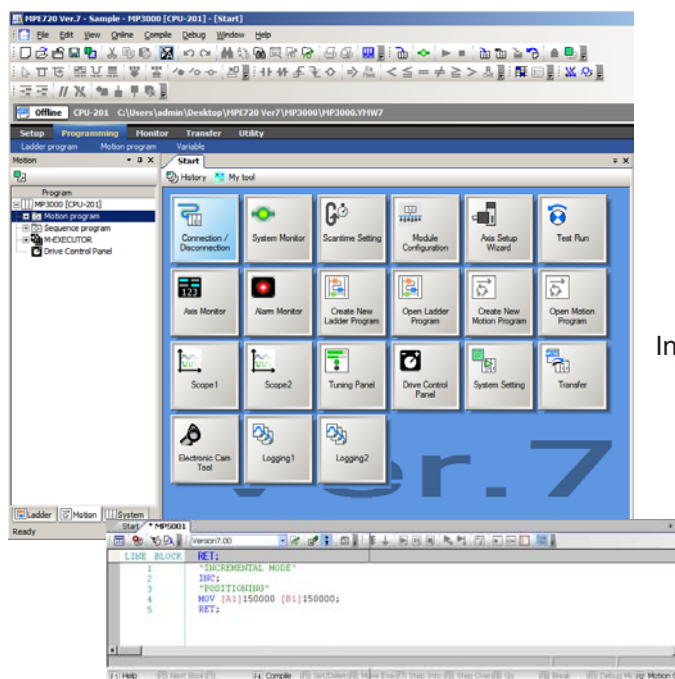


Machine Controller MP3000 Series

Motion Program

PROGRAMMING MANUAL



Introduction to Motion Programs **1**

Introduction to Sequence Programs **2**

Program Development Flow **3**

Registers **4**

Programming Rules **5**

Motion Language Instructions **6**

Features of the MPE720
Engineering Tool **7**

Specifications **AppA**

Sample Programs **AppB**

Differences between MP2000-series
and MP3000-series Machine Controllers **AppC**

Precautions **AppD**

Copyright © 2012 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 information on motion programming for MP3000-series Machine Controllers.

Read this manual carefully to ensure the correct usage of the Machine Controller and apply the Machine Controller to control your manufacturing system.

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 personnel.

- Designers for systems that use MP3000-series Machine Controllers
- Programmers of the motion programs and sequence programs for MP3000-series Machine Controllers

◆ Basic Terms

Unless otherwise specified, the following definitions are used:

- Fixed parameters: The motion fixed parameters.
- Setting parameters: The motion setting parameters.
- Monitor parameters: The motion monitor parameters.
- Machine Controller: MP3000-series Machine Controller
- MP3200: A generic name for the Power Supply Unit, CPU Unit, Base Unit, and Rack Expansion Interface Unit.
- MP3300: A generic name for the CPU Module and Base Unit.
- MPE720: The Engineering Tool or a personal computer running the Engineering Tool
- Motion Control Function Modules: The Function Modules in the Motion Modules and the Function Modules in the SVR, SVC, SVC 32, or SVR 32 built into the CPU Units/CPU Modules.

◆ MPE720 Engineering Tool Version Number

In this manual, the operation of MPE720 is described using screen captures of MPE720 version 7.

◆ The Meaning of “Torque” in This Manual

Although the term “torque” is commonly used when describing rotary Servomotors and “force” is used when describing linear Servomotors, this manual uses “torque” when describing either one (excluding parameter names).

◆ Visual Aids

The following aids are used to indicate certain types of information for easier reference.



Important

Indicates precautions or restrictions that must be observed.

Indicates alarm displays and other precautions that will not result in machine damage.



Note

Indicates items for which caution is required or precautions to prevent operating mistakes.

Example

Indicates operating or setting examples.

Information

Indicates supplemental information to deepen understanding or useful information.



Terms

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

Related Manuals

The following table lists the related manuals. Refer to these manuals as required.

Be aware of all product specifications and restrictions to product application before you attempt to use any product.

Category	Manual Name	Manual Number	Contents
Basic functionality	Machine Controller MP2000/MP3000 Series Machine Controller System Setup Manual	SIEP C880725 00	Describes the functions of the MP2000/MP3000-series Machine Controllers and the procedures that are required to use the Machine Controller, from installation and connections to settings, programming, trial operation, and debugging.
	Machine Controller MP3000 Series MP3200/MP3300 Troubleshooting Manual	SIEP C880725 01	Describes troubleshooting an MP3000-series Machine Controller.
	Machine Controller MP3000 Series MP3200 User's Manual	SIEP C880725 10	Describes the specifications and system configuration of the Basic Units in an MP3000-series Machine Controller and the functions of the CPU Unit.
	Machine Controller MP3000 Series MP3300 Product Manual	SIEP C880725 21	Describes the specifications and system configuration of an MP3000-series MP3300 Machine Controller and the functions of the CPU Module.
Communications functionality	Machine Controller MP3000 Series Communications User's Manual	SIEP C880725 12	Describes the specifications, system configuration, and communications connection methods for the Ethernet communications that are used with an MP3000-series Machine Controller.
Motion control functionality	Machine Controller MP3000 Series Motion Control User's Manual	SIEP C880725 11	Describes the specifications, system configuration, and operating methods for the SVC, SVC32, SVR, and SVR32 Motion Function Modules that are used in an MP3000-series Machine Controller.
	Machine Controller MP2000 Series Pulse Output Motion Module PO-01 User's Manual	SIEP C880700 28	Describes the functions, specifications, and operating methods of the MP2000-series PO-01 Motion Module.
	Machine Controller MP2000 Series SVA-01 Motion Module User's Manual	SIEP C880700 32	Describes the functions, specifications, and operating methods of the MP2000-series SVA-01 Motion Module.
	Machine Controller MP2000 Series Built-in SVB/SVB-01 Motion Module User's Manual	SIEP C880700 33	Describes the functions, specifications, and operating methods of the MP2000-series Motion Module (built-in Function Modules: SVB, SVB-01, and SVR).
	Machine Controller MP2000 Series SVC-01 Motion Module User's Manual	SIEP C880700 41	Describes the functions, specifications, and operating methods of the MP2000-series SVC-01 Motion Module.
Programming	Machine Controller MP3000 Series Ladder Programming Manual	SIEP C880725 13	Describes the ladder programming specifications and instructions of MP3000-series Machine Controller.
Engineering Tool	Machine Controller MP2000/MP3000 Series Engineering Tool MPE720 Version 7 User's Manual	SIEP C880761 03	Describes how to operate MPE720 version 7.

Safety Precautions

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.



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




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 () can lead to serious results depending on circumstances.



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



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

The following precautions are for storage, transportation, installation, wiring, operation, maintenance, inspection, and disposal. These precautions are important and must be observed.

◆ General Precautions

WARNING

- The installation must be suitable and it must be performed only by an experienced technician.
There is a risk of electrical shock or injury.
- Before connecting the machine and starting operation, make sure that an emergency stop procedure has been provided and is working correctly.
There is a risk of injury.
- Do not approach the machine after a momentary interruption to the power supply. When power is restored, the Machine Controller and the device connected to it may start operation suddenly. Provide safety measures in advance to ensure human safety when operation restarts.
There is a risk of injury.
- Do not touch anything inside the Machine Controller.
There is a risk of electrical shock.
- Do not remove the front cover, cables, connector, or options while power is being supplied.
There is a risk of electrical shock, malfunction, or damage.
- Do not damage, pull on, apply excessive force to, place heavy objects on, or pinch the cables.
There is a risk of electrical shock, operational failure of the Machine Controller, or burning.
- Do not attempt to modify the Machine Controller in any way.
There is a risk of injury or device damage.

◆ Storage and Transportation

CAUTION

- Do not store the Machine Controller in any of the following locations.
 - Locations that are subject to direct sunlight
 - Locations that are subject to ambient temperatures that exceed the storage conditions
 - Locations that are subject to ambient humidity that exceeds the storage conditions
 - Locations that are subject to rapid temperature changes and condensation
 - Locations that are subject to corrosive or inflammable gas
 - Locations that are subject to excessive dust, dirt, salt, or metallic powder
 - Locations that are subject to water, oil, or chemicals
 - Locations that are subject to vibration or shockThere is a risk of fire, electrical shock, or device damage.
- Hold onto the main body of the Machine Controller when transporting it.
Holding the cables or connectors may damage them or result in injury.
- Do not overload the Machine Controller during transportation. (Follow all instructions.)
There is a risk of injury or an accident.
- Never subject the Machine Controller to an atmosphere containing halogen (fluorine, chlorine, bromine, or iodine) during transportation.
There is a risk of malfunction or damage.
- 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.
Example: Heat treatment, where materials are kiln-dried to a core temperature of 56°C for 30 minutes or more.
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.

◆ Installation

CAUTION

- Do not install the Machine Controller in any of the following locations.
 - Locations that are subject to direct sunlight
 - Locations that are subject to ambient temperatures that exceed the operating conditions
 - Locations that are subject to ambient humidity that exceeds the operating conditions
 - Locations that are subject to rapid temperature changes and condensation
 - Locations that are subject to corrosive or inflammable gas
 - Locations that are subject to excessive dust, dirt, salt, or metallic powder
 - Locations that are subject to water, oil, or chemicals
 - Locations that are subject to vibration or shockThere is a risk of fire, electrical shock, or device damage.
- Never install the Machine Controller in an atmosphere containing halogen (fluorine, chlorine, bromine, or iodine).
There is a risk of malfunction or damage.
- Do not step on the Machine Controller or place heavy objects on the Machine Controller.
There is a risk of injury or an accident.
- Do not block the air exhaust ports on the Machine Controller. Do not allow foreign objects to enter the Machine Controller.
There is a risk of internal element deterioration, malfunction, or fire.
- Always mount the Machine Controller in the specified orientation.
There is a risk of malfunction.
- Leave the specified amount of space between the Machine Controller, and the interior surface of the control panel and other devices.
There is a risk of fire or malfunction.
- Do not subject the Machine Controller to strong shock.
There is a risk of malfunction.
- Suitable battery installation must be performed and it must be performed only by an experienced technician.
There is a risk of electrical shock, injury, or device damage.
- Do not touch the electrodes when installing the Battery.
Static electricity may damage the electrodes.

◆ Wiring

CAUTION

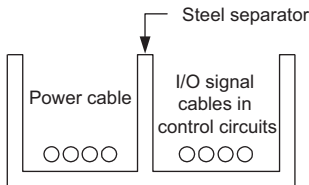
- Check the wiring to be sure it has been performed correctly.
There is a risk of motor run-away, injury, or accidents.
- Always use a power supply of the specified voltage.
There is a risk of fire or accident.
- In places with poor power supply conditions, ensure that the input power is supplied within the specified voltage range.
There is a risk of device damage.
- Install breakers and other safety measures to provide protection against shorts in external wiring.
There is a risk of fire.
- Provide sufficient shielding when using the Machine Controller in the following locations.
 - Locations that are subject to noise, such as from static electricity
 - Locations that are subject to strong electromagnetic or magnetic fields
 - Locations that are subject to radiation
 - Locations that are near power linesThere is a risk of device damage.
- Configure the circuits to turn ON the power supply to the CPU Unit/CPU Module before the 24-V I/O power supply. Refer to the following manual for details on circuits.
 - 📖 *MP3000 Series CPU Unit Instructions* (Manual No.: TOBP C880725 16)
 - 📖 *MP3000 Series MP3300 CPU Module Instructions* (Manual No.: SIEP C880725 23)If the power supply to the CPU Unit/CPU Module is turned ON after the external power supply, e.g., the 24-V I/O power supply, the outputs from the CPU Unit/CPU Module may momentarily turn ON when the power supply to the CPU Unit/CPU Module turns ON. This can result in unexpected operation that may cause injury or device damage.
- Provide emergency stop circuits, interlock circuits, limit circuits, and any other required safety measures in control circuits outside of the Machine Controller.
There is a risk of injury or device damage.
- If you use MECHATROLINK I/O Modules, use the establishment of MECHATROLINK communications as an interlock output condition.
There is a risk of device damage.
- Connect the Battery with the correct polarity.
There is a risk of battery damage or explosion.
- Suitable battery replacement must be performed and it must be performed only by an experienced technician.
There is a risk of electrical shock, injury, or device damage.
- Do not touch the electrodes when replacing the Battery.
Static electricity may damage the electrodes.
- Select the I/O signal wires for external wiring to connect the Machine Controller to external devices based on the following criteria:
 - Mechanical strength
 - Noise interference
 - Wiring distance
 - Signal voltage

⚠ CAUTION

- Separate the I/O signal cables for control circuits from the power cables both inside and outside the control panel to reduce the influence of noise from the power cables.

If the I/O signal lines and power lines are not separated properly, malfunction may occur.

Example of Separated Cables



◆ Operation

⚠ CAUTION

- Follow the procedures and instructions in the user's manuals for the relevant Machine Controllers to perform normal operation and trial operation.
Operating mistakes while the Servomotor and machine are connected may damage the machine or even cause accidents resulting in injury or death.
- Implement interlock signals and other safety circuits external to the Machine Controller to ensure safety in the overall system even if the following conditions occur.
 - Machine Controller failure or errors caused by external factors
 - Shutdown of operation due to Machine Controller detection of an error in self-diagnosis and the subsequent turning OFF or holding of output signals
 - Holding of the ON or OFF status of outputs from the Machine Controller due to fusing or burning of output relays or damage to output transistors
 - Voltage drops from overloads or short-circuits in the 24-V output from the Machine Controller and the subsequent inability to output signals
 - Unexpected outputs due to errors in the power supply, I/O, or memory that cannot be detected by the Machine Controller through self-diagnosis.

There is a risk of injury, device damage, or burning.

- Observe the setting methods that are given in the manual for the following parameters.
 - Parameters for absolute position detection when the axis type is set to a finite-length axis
 - Parameters for simple absolute infinite-length position control when the axis type is set to an infinite-length axis

📖 *MP3000 Series Motion Control User's Manual* (Manual No. SIEP C880725 11)

If any other methods are used, offset in the current position when the power supply is turned OFF and ON again may result in device damage.

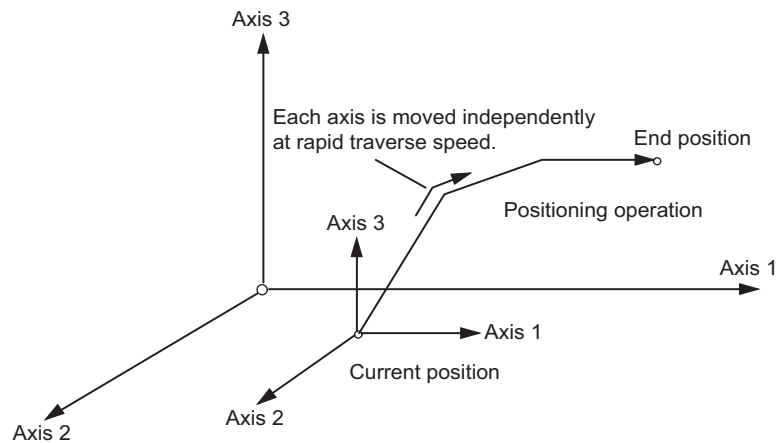
- OL□□□48 (Zero Point Position Offset in Machine Coordinate System) is always valid when the axis type is set to a finite-length axis. Do not change the setting of OL□□□48 while the Machine Controller is operating.

There is a risk of machine damage or an accident.

⚠ CAUTION

- Always check to confirm the paths of axes when any of the following axis movement instructions are used in programs to ensure that the system operates safely.
 - Positioning (MOV)
 - Linear Interpolation (MVS)
 - Circular Interpolation (MCC or MCW)
 - Helical Interpolation (MCC or MCW)
 - Set-time Positioning (MVT)
 - Linear Interpolation with Skip Function (SKP)
 - Zero Point Return (ZRN)
 - External Positioning (EXM)

Example



Example of Basic Path for Positioning (MOV) Instruction

There is a risk of injury or device damage.

- The same coordinate word will create a completely different travel operation in Absolute Mode and in Incremental Mode. Make sure that the ABS and INC instructions are used correctly before you start operation.

There is a risk of injury or device damage.

- The travel path for the Positioning (MOV) instructions will not necessarily be a straight line. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely.

There is a risk of injury or device damage.

- The Linear Interpolation (MVS) instruction can be used on both linear axes and rotary axes. However, if a rotary axis is included, the linear interpolation path will not necessarily be a straight line. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely.

There is a risk of injury or device damage.

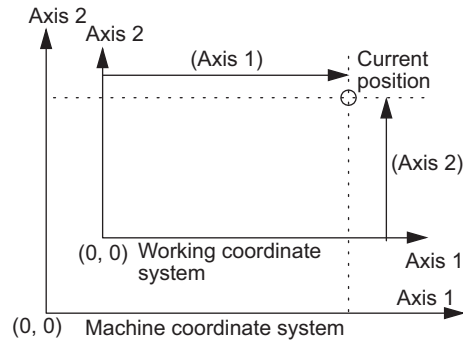
- The linear interpolation for the Helical Interpolation (MCW and MCC) instructions can be used for both linear axes and rotary axes. However, depending on how the linear axis is taken, the path of helical interpolation will not be a helix. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely.

There is a risk of injury or device damage.

⚠ CAUTION

- Unexpected operation may occur if the following coordinate instructions are specified incorrectly: Always confirm that the following instructions are specified correctly before you begin operation.
 - Absolute Mode (ABS)
 - Incremental Mode (INC)
 - Current Position Set (POS)

Example



Example of Working Coordinate System Created with the Set Current Position (POS) Instruction

There is a risk of injury or device damage.

- The Set Current Position (POS) Instruction creates a new working coordinate system. Therefore, unexpected operation may occur if the POS instruction is specified incorrectly. When you use the POS instruction, always confirm that the working coordinate system is in the correct position before you begin operation.

There is a risk of injury or device damage.

- The Move on Machine Coordinates (MVM) instruction temporarily performs positioning to a coordinate position in the machine coordinate system. Therefore, unexpected operation may occur if the instruction is executed without confirming the zero point position in the machine coordinate system first. When you use the MVM instruction, always confirm that the machine zero point is in the correct position before you begin operation.

There is a risk of injury or device damage.

◆ Maintenance and Inspection

CAUTION

- Do not attempt to disassemble or repair the Machine Controller.
There is a risk of electrical shock, injury, or device damage.
- Do not change any wiring while power is being supplied.
There is a risk of electrical shock, injury, or device damage.
- Suitable battery replacement must be performed and it must be performed only by an experienced technician.
There is a risk of electrical shock, injury, or device damage.
- Replace the Battery only while power is supplied to the Machine Controller.
Replacing the Battery while the power supply to the Machine Controller is turned OFF may result in loss of the data stored in memory in the Machine Controller.
- Do not touch the electrodes when you replace the Battery.
Static electricity may damage the electrodes.
- Do not forget to perform the following tasks when you replace the CPU Unit/CPU Module:
 - Back up all programs and parameters from the CPU Unit/CPU Module that is being replaced.
 - Transfer all saved programs and parameters to the new CPU Unit/CPU Module.If you operate the CPU Unit/CPU Module without transferring this data, unexpected operation may occur. There is a risk of injury or device damage.
- Do not touch the heat sink on the CPU Unit/CPU Module while the power supply is turned ON or for a sufficient period of time after the power supply is turned OFF.
The heat sink may be very hot, and there is a risk of burn injury.

◆ Disposal

CAUTION

- Dispose of the Machine Controller as general industrial waste.
- Observe all local laws and ordinances when you dispose of used Batteries.

◆ Other General Precautions

Observe the following general precautions to ensure safe application.

- The products shown in the 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.
- The illustrations that are presented in this manual are typical examples and may not match the product you received.
- 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.

Warranty

◆ 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.

- 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
- Causes not attributable to the delivered product itself
- Modifications or repairs not performed by Yaskawa
- Abuse of the delivered product in a manner in which it was not originally intended
- Causes that were not foreseeable with the scientific and technological understanding at the time of shipment from Yaskawa
- Events for which Yaskawa is not responsible, such as natural or human-made disasters

◆ Limitations of Liability

- 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.
- 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.
- 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.
- 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.

◆ Suitability for Use

- 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.
- The customer must confirm that the Yaskawa product is suitable for the systems, machines, and equipment used by the customer.
- 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
- 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.
- 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.
- Read and understand all use prohibitions and precautions, and operate the Yaskawa product correctly to prevent accidental harm to third parties.

◆ 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
Related Manuals	v
Safety Precautions	vi
Warranty	xiv

1

Introduction to Motion Programs

1.1	What Is a Motion Program?	1-3
1.2	Features of Motion Programs	1-4
	Motion Program Execution Methods	1-4
	Full Synchronization of Sequence Control and Motion Control	1-4
	Advanced Motion Control	1-5
	Easy-to-understand Motion Language Instructions	1-5
	Numerical Calculations in Motion Programs	1-5
	Data Transfer to and from Ladder Programs	1-6
	Memory Usage Reduced by Use of Subprograms	1-6
	Parallel Execution of Programs	1-7
	Axis Alarm Checks	1-10
	Online Editing of Programs	1-12
	Easy Programming Functions (MPE720 Version 7.0 or Later)	1-13
1.3	Motion Program System Configuration	1-14
1.4	Types of Motion Programs	1-15
1.5	Motion Program Groups	1-16
1.6	Motion Program Execution Timing	1-17
1.7	Executing Motion Programs	1-19
	Execution Processing Method	1-19
	Program Execution Registration Methods	1-22
	Work Registers	1-23
1.8	Advanced Programming	1-31
	Indirect Designation of a Program Number Using a Register	1-31
	Controlling Motion Programs Directly from an External Device	1-32
	Monitoring Motion Program Execution Information	1-33
1.9	Application Examples	1-43
	Conveyance Device	1-43
	Part Inserter	1-43
	Panel Processing Machine	1-44
	Metal Sheet Pressing Equipment	1-44

2

Introduction to Sequence Programs

2.1	What Is a Sequence Program?	2-2
2.2	Features of a Sequence Program.	2-3
	Sequence Program Execution Methods.	2-3
	Same Language as Motion Programs	2-3
	Data Transfer to and from Motion Programs	2-3
	Memory Usage Reduced by Use of Subprograms.	2-4
	Easy Programming Functions	2-4
2.3	Types of Sequence Programs	2-5
2.4	Executing Sequence Programs	2-6
	Execution Processing Method	2-6
	Registering Program Execution	2-8
	Work Registers.	2-9

3

Program Development Flow

3.1	Program Development Flow	3-2
3.2	Program Development Procedures	3-3
	Preparation for Devices to be Connected	3-3
	Creating a Project.	3-4
	Self Configuration.	3-6
	Going Online	3-6
	Group Definition Settings	3-6
	Creating Programs	3-8
	Registering Program Execution	3-10
	Transferring the Programs	3-13
	Debugging Programs	3-16
	Saving the Programs to Flash Memory	3-17
	Executing the Programs.	3-18

4

Registers

4.1	Registers.	4-2
	Types of Registers	4-2
	Global Registers.	4-5
	Local Registers.	4-6
	Data Types	4-8
4.2	Using Registers	4-11
	System Registers (S Registers)	4-11
	Data Registers (M Registers).	4-12
	Data Registers (G Registers).	4-13
	Input Registers (I Registers).	4-14
	Output Registers (O Registers)	4-15
	C Registers.	4-16
	D Registers.	4-17

4.3	Using Indices i and j	4-18
4.4	Using Array Registers	4-20

5

Programming Rules

5.1	Entering Programs	5-2
	Motion Program Structure	5-2
	Block Format	5-2
	Notation for Constants and Registers	5-8
5.2	Group Definition Details	5-9
5.3	Operation Priority Levels.	5-11
5.4	Instruction Types and Execution Scans	5-13
	Instruction Types	5-13
	Instruction Type Table	5-15
5.5	Programming with Variables	5-17
	Declaring Variables	5-17
	Variable Format	5-18
	Strings That Cannot Be Used in Variable Names	5-20
	Programming Examples.	5-21

6

Motion Language Instructions

6.1	Axis Setting Instructions	6-4
	Absolute Mode (ABS)	6-7
	Incremental Mode (INC).	6-11
	Change Acceleration Time (ACC)	6-15
	Change Deceleration Time (DCC).	6-21
	Change S-curve Time Constant (SCC)	6-27
	Set Speed (VEL)	6-33
	Set Maximum Interpolation Feed Speed (FMX).	6-39
	Set Maximum Individual Axis Speeds for Interpolation (IFMX)	6-42
	Change Interpolation Feed Speed Unit (FUT)	6-45
	Set Interpolation Feed Speed Ratio (IFP)	6-47
	Change Interpolation Acceleration Time (IAC)	6-50
	Change Interpolation Deceleration Time (IDC)	6-52
	Change Interpolation Deceleration Time for Temporary Stop (IDH)	6-54
	Change Interpolation Acceleration/Deceleration Unit (IUT).	6-58
	Set Interpolation Feed Speed Axes (+ and -).	6-60
	Set Interpolation Acceleration/Deceleration Mode (ACCMODE).	6-63

6.2	Axis Movement Instructions	6-77
	Positioning (MOV)	6-81
	Linear Interpolation (MVS)	6-85
	Circular Interpolation with Specified Center Point (MCW and MCC)	6-90
	Circular Interpolation with Specified Radius (MCW and MCC)	6-95
	Helical Interpolation with Specified Center Point (MCW and MCC)	6-99
	Helical Interpolation with Specified Radius (MCW and MCC)	6-102
	Zero Point Return (ZRN)	6-104
	Position after Distribution (DEN)	6-107
	Linear Interpolation with Skip Function (SKP)	6-109
	Set-time Positioning (MVT)	6-111
	External Positioning (EXM)	6-113
6.3	Axis Control Instructions	6-115
	Current Position Set (POS)	6-117
	Move on Machine Coordinates (MVM)	6-119
	Update Program Current Position (PLD)	6-120
	In-position Check (PFN)	6-122
	In-Position Range (INP)	6-124
	Positioning Completed Check (PFP)	6-126
	Coordinate Plane Setting (PLN)	6-128
6.4	Program Control Instructions	6-129
	Branching Instructions (IF, ELSE, and IEND)	6-131
	Repetition Instructions (WHILE, WEND)	6-134
	Repetition with One Scan Wait (WHILE and WENDX)	6-137
	Parallel Execution Instructions (PFORK, JOINTO, and PJOINT)	6-140
	Selective Execution Instructions (SFORK, JOINTO, SJOINT)	6-143
	Call Motion Subprogram (MSEE)	6-148
	Call Sequence Subprogram (SSEE)	6-149
	Call User Function from Motion Program (UFC)	6-150
	Call User Function from Sequence Program (FUNC)	6-158
	Program End (END)	6-159
	Subprogram Return (RET)	6-160
	Dwell Time (TIM)	6-161
	Dwell Time (TIM1MS)	6-162
	I/O Variable Wait (IOW)	6-163
	One Scan Wait (EOX)	6-166
	Disable Single-block Signal (SNGD) and Enable Single-block Signal (SNGE)	6-167
6.5	Numeric Operation Instructions	6-168
	Substitute (=)	6-169
	Add (+)	6-170
	Subtract (-)	6-171
	Extended Add (++)	6-172
	Extended Subtract (--)	6-174
	Multiply (*)	6-176
	Divide (/)	6-177
	Modulo (MOD)	6-178
6.6	Logic Operation Instructions	6-179
	Inclusive OR ()	6-180
	AND (&)	6-181
	Exclusive OR (^)	6-182
	NOT (!)	6-183

6.7	Numeric Comparison Instructions	6-184
	Numeric Comparison Instructions (==, <>, >, <, >=, <=)	6-186
6.8	Data Manipulations	6-189
	Bit Shift Right (SFR)	6-189
	Bit Shift Left (SFL)	6-191
	Move Block (BLK)	6-192
	Clear (CLR)	6-193
	Table Initialization (SETW)	6-194
	ASCII Conversion 1 (ASCII)	6-196
6.9	Basic Functions	6-198
	Sine (SIN)	6-200
	Cosine (COS)	6-201
	Tangent (TAN)	6-202
	Arc Sine (ASN)	6-203
	Arc Cosine (ACS)	6-204
	Arc Tangent (ATN)	6-205
	Square Root (SQT)	6-206
	BCD to Binary (BIN)	6-208
	Binary to BCD (BCD)	6-209
	Set Bit (S{ })	6-210
	Reset Bit (R{ })	6-211
	Rising-edge Pulse (PON)	6-212
	Falling-edge Pulse (NON)	6-214
	On-delay Timer: Measurement unit = 10 ms (TON)	6-216
	1-ms ON-Delay Timer (TON1MS)	6-217
	Off-delay Timer: Measurement unit = 10 ms (TOF)	6-218
	1-ms OFF-Delay Timer (TOF1MS)	6-219
6.10	Vision Instructions	6-220

7

Features of the MPE720 Engineering Tool

7.1	Motion Editor	7-2
7.2	Motion Instruction Entry Assistance	7-5
7.3	Task Assignments	7-9
7.4	Debug Operation	7-11
7.5	Drive Control Panel	7-18
7.6	Test Runs	7-20
7.7	Axis Monitor and Alarm Monitor	7-23
7.8	Cross References	7-27

Appendix A Specifications

A.1	Applicable Units and Modules	A-2
A.2	Machine Controller Specifications	A-3

Appendix B Sample Programs

B.1	Motion Program Control Program.	B-2
B.2	Parallel Processing.	B-3
B.3	Performing Speed Control with a Motion Program	B-4
B.4	Simple Synchronized Operation with a Virtual Axis	B-5
B.5	Sequence Programs.	B-7

Appendix C Differences between MP2000-series and MP3000-series Machine Controllers

Appendix D Precautions

D.1	General Precautions	D-2
	Saving Data to Flash Memory when Changing Applications	D-2
	Debugging a System in Operation	D-2
D.2	Precautions on Motion Parameters	D-3
	Performing Axis Movement Instructions on the Same Axis in Motion Programs	D-3
	Using a Subscript to Reference a Motion Register from an I/O Register	D-3
	Referencing the Motion Register of a Different Circuit	D-4
	OL□□□1C (Position Reference Setting) Setting Parameter	D-5
	Axis Operation for Software Limit Alarms.	D-5

Index

Revision History

Introduction to Motion Programs

1

This chapter introduces motion programs, their features, and how to use them for first-time users of motion programs.

1.1	What Is a Motion Program?	1-3
1.2	Features of Motion Programs	1-4
	Motion Program Execution Methods	1-4
	Full Synchronization of Sequence Control and Motion Control . .	1-4
	Advanced Motion Control	1-5
	Easy-to-understand Motion Language Instructions	1-5
	Numerical Calculations in Motion Programs	1-5
	Data Transfer to and from Ladder Programs	1-6
	Memory Usage Reduced by Use of Subprograms	1-6
	Parallel Execution of Programs	1-7
	Axis Alarm Checks	1-10
	Online Editing of Programs	1-12
	Easy Programming Functions (MPE720 Version 7.0 or Later) . .	1-13
1.3	Motion Program System Configuration	1-14
1.4	Types of Motion Programs	1-15
1.5	Motion Program Groups	1-16
1.6	Motion Program Execution Timing	1-17
1.7	Executing Motion Programs	1-19
	Execution Processing Method	1-19
	Program Execution Registration Methods	1-22
	Work Registers	1-23

1.8 **Advanced Programming** 1-31

Indirect Designation of a Program Number
Using a Register 1-31
Controlling Motion Programs Directly
from an External Device 1-32
Monitoring Motion Program Execution Information 1-33

1.9 **Application Examples** 1-43

Conveyance Device 1-43
Part Inserter 1-43
Panel Processing Machine 1-44
Metal Sheet Pressing Equipment 1-44

1.1

What Is a Motion Program?

Motion programs are programs that are written in Yaskawa's motion language, which is a textual programming language.

In comparison with ladder programs, motion programs allow you to execute various operations with one line of motion language code. As opposed to ladder programs, motion programs allow you to set the target position, acceleration/deceleration times, or interpolation feed speeds for interpolation instructions to automatically calculate the travel distance each scan based on parameters that are set in the system.

You can execute motion programs either by placing an MSEE instruction in a ladder program or by calling the motion programs from the M-EXECUTOR program execution definitions.

You can create up to 512 motion programs. These are in addition to any ladder programs.

The following is an example of a motion program.

LINE	BLOCK	Code	Description
1		"MPM001";	
2	0	OW803C=3;	"X Axis zero point return method(3: phase-C)"
3	1	OW80BC=3;	"Y Axis zero point return method(3: phase-C)"
4	2	VEL [X]1000 [Y]1000;	"Travel speed for positioning command"
5	3	ACC[X]100[Y]100;	"Acceleration time"
6	4	DCC[X]100[Y]100;	"Deceleration time"
7	5	OW803E=100;	"X Axis approach speed(mm/min)"
8	6	OW8040=50;	"X Axis creep speed(mm/min)"
9	7	OL8042=10000;	"X Axis final travel distance(0.001mm)"
10	8	OW80BE=100;	"Y Axis approach speed(mm/min)"
11	9	OW80C0=50;	"Y Axis creep speed(mm/min)"
12	10	OL80C2=10000;	"Y Axis final travel distance(0.001mm)"
13	11	ZRN[X]00[Y]00;	"Zero point return command"
14	12	END;	
15			

1.2 Features of Motion Programs

This section describes the features of motion programs.

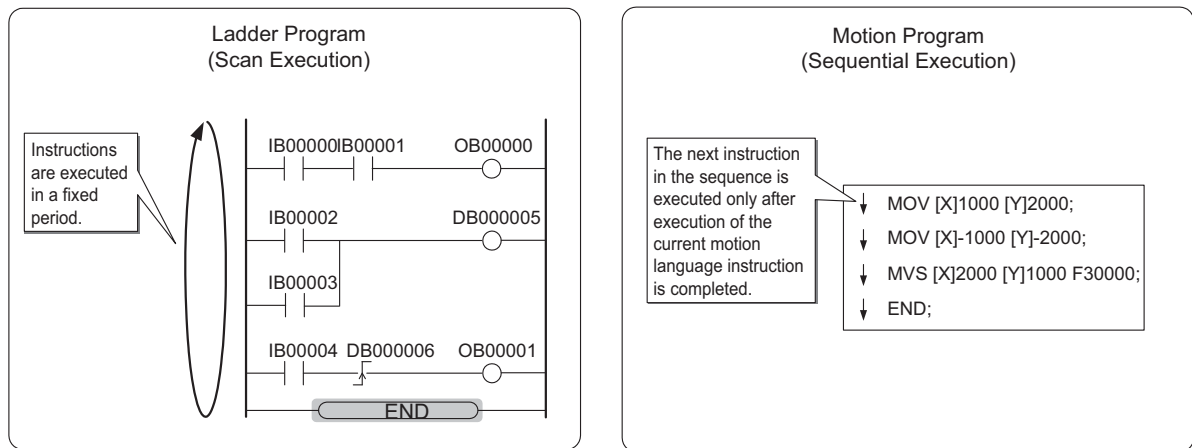
Motion Program Execution Methods

Motion programs are executed in a different way from ladder programs.

With a ladder program, processing from the start of the program to the END command is completed in one scan.

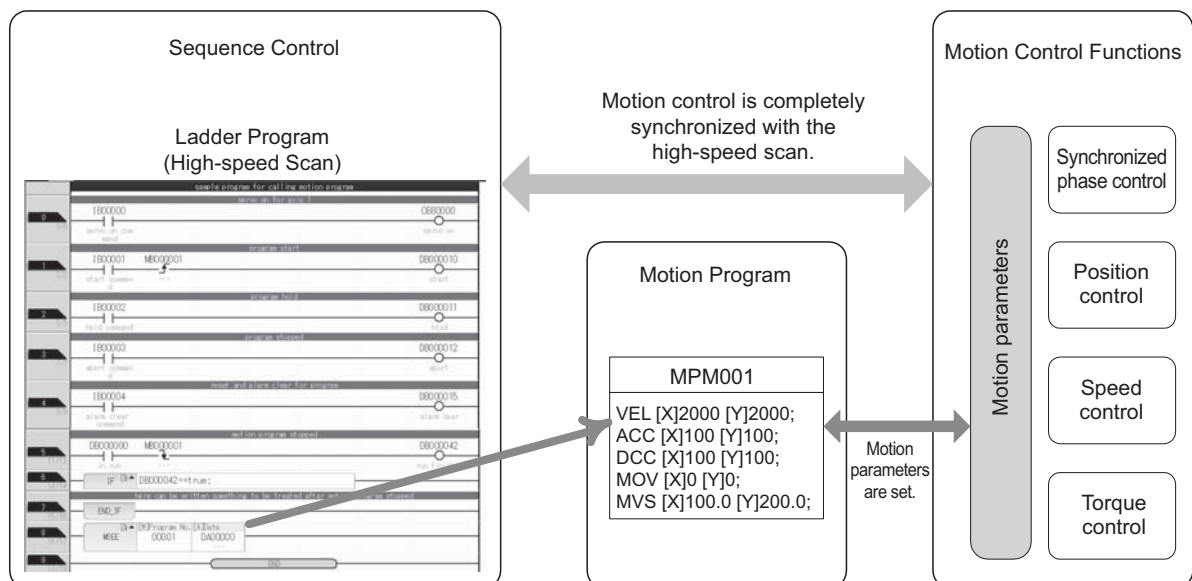
With a motion program, the processing requested for even one instruction normally requires more than one scan. Also, the instructions are executed sequentially in the order that they are programmed.

In this manual, the execution method for ladder programs is called scan execution, and the execution method for motion programs is called sequential execution.



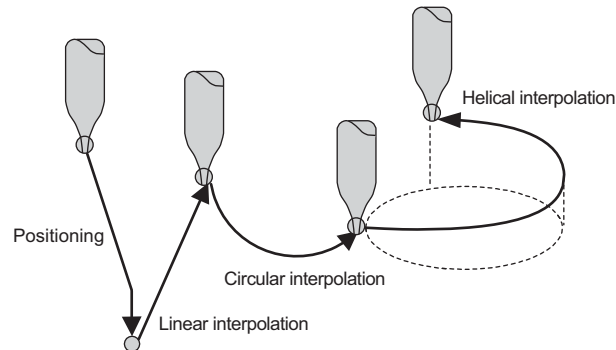
Full Synchronization of Sequence Control and Motion Control

Execution of the processing that is programmed in a motion program is completely synchronized with the high-speed scan of a MP3000-series Machine Controller. Execution of the motion program occurs within one scan from when a start request is executed in a ladder program. There is no time delay.



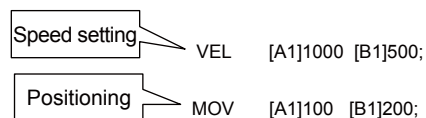
Advanced Motion Control

In addition to basic motion control, motion programs can also be used to easily achieve motion control for complex movements.



Easy-to-understand Motion Language Instructions

A motion program uses intuitive motion language commands such as VEL to set a velocity and MOV for positioning.



Numerical Calculations in Motion Programs

The motion language includes commands for arithmetic operations and logic operations. These commands allow you to include various calculations, such as calculations of target positions, in motion programs.

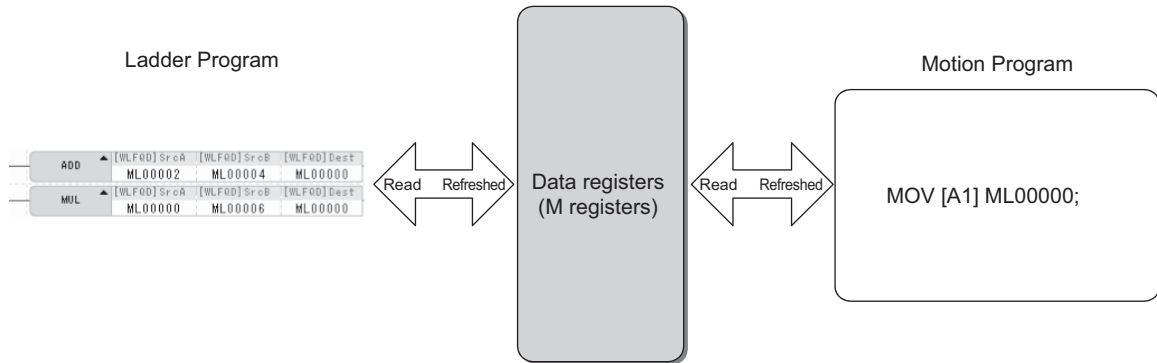
```
DL00000 = DL00002 + DW00004;
DL00000 = DW00002 * DL00004;
MW00000 = MW00000 & 00FFH;
MF00000 = SIN(30.0);
```

Data Transfer to and from Ladder Programs

You can pass data between ladder programs and motion programs.

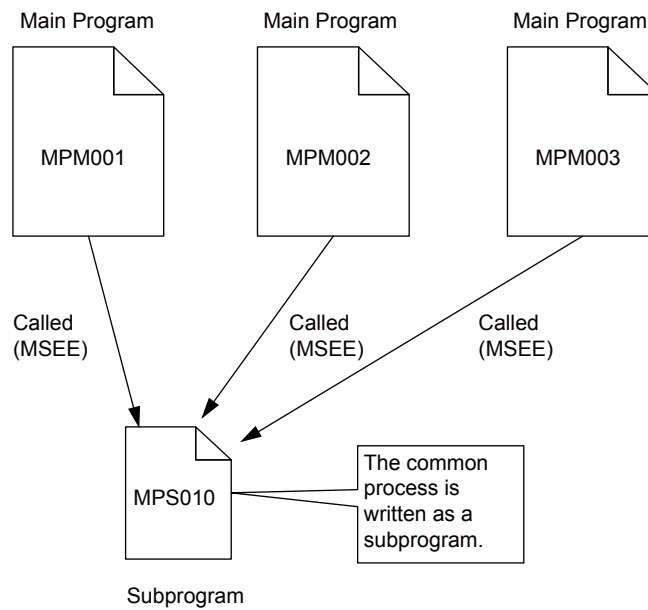
Data registers (M registers) are used to transfer data.

For example, this allows a value that is updated in a ladder program to be used in a motion program, and vice-versa.



Memory Usage Reduced by Use of Subprograms

Subprograms can be created within a motion program. Subprograms are created to perform common operations. They help minimizing the number of program steps and allow the efficient use of memory.



Parallel Execution of Programs

Up to 32 tasks can be executed simultaneously with a single MP3000-series Machine Controller using motion programs. This type of parallel execution can be used to control many different motion operations simultaneously.

Use the PFORK instruction in a main program or subprogram to perform operations in parallel. Up to 8 forks can be performed in parallel for each task.

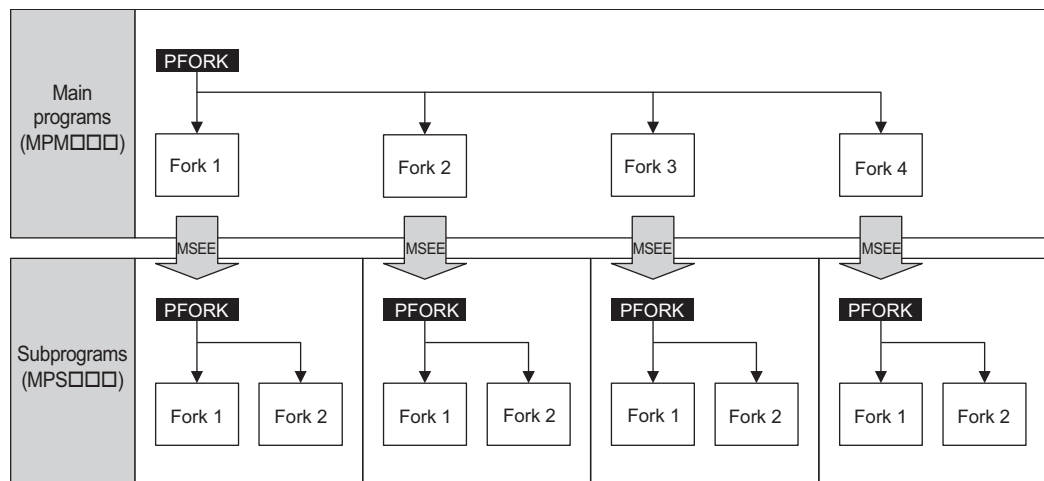
The parallel execution mode is set in the Program Properties Dialog Box.

There are four parallel execution modes for the PFORK instruction. The following sections describe these modes individually.

Main 4 × Sub 2 (MP2000-compatible Mode)

In this mode, up to four forks can be executed in parallel in a main program, and up to two forks can be executed in parallel in a subprogram.

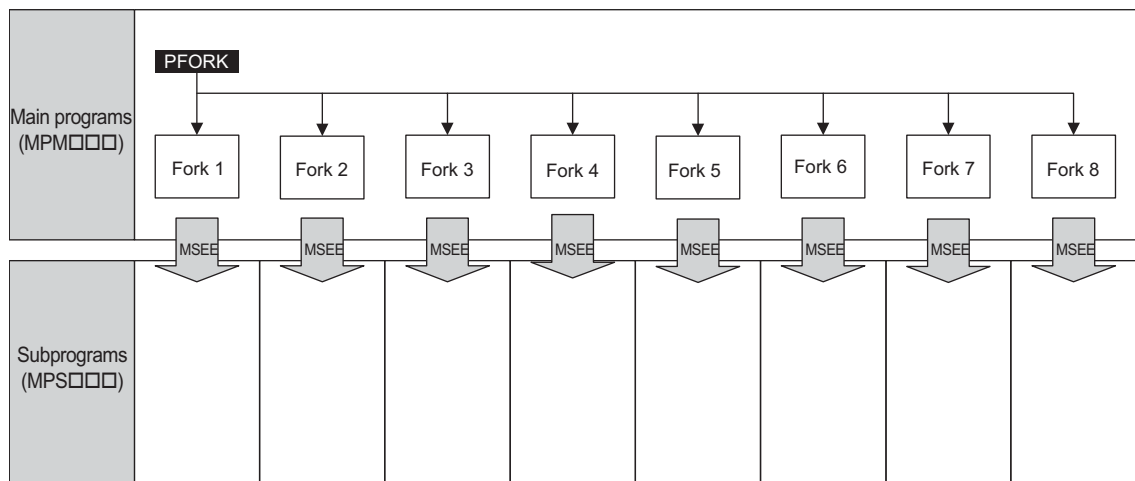
This is the default mode.



Main 8 × Sub 1

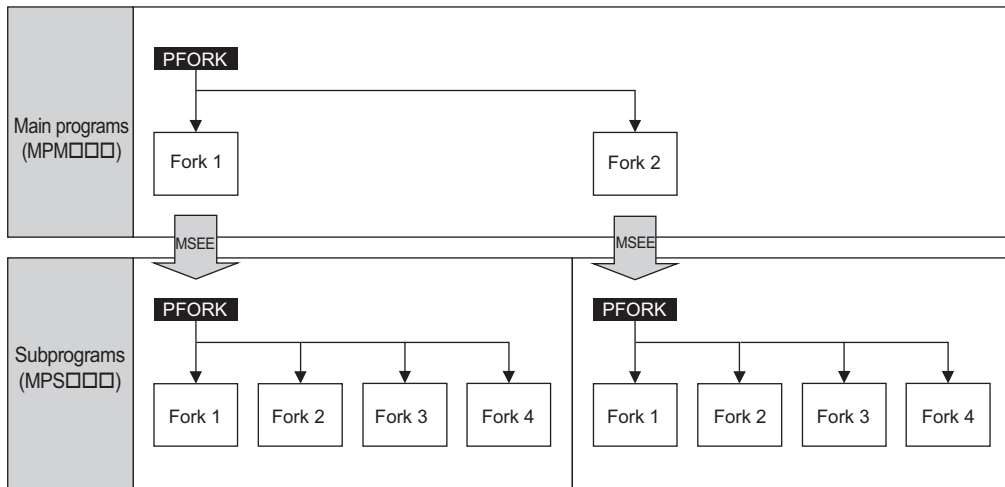
This mode allows the parallel execution of up to 8 forks in a main program.

Parallel execution is not possible for subprograms in this mode.



Main 2 × Sub 4

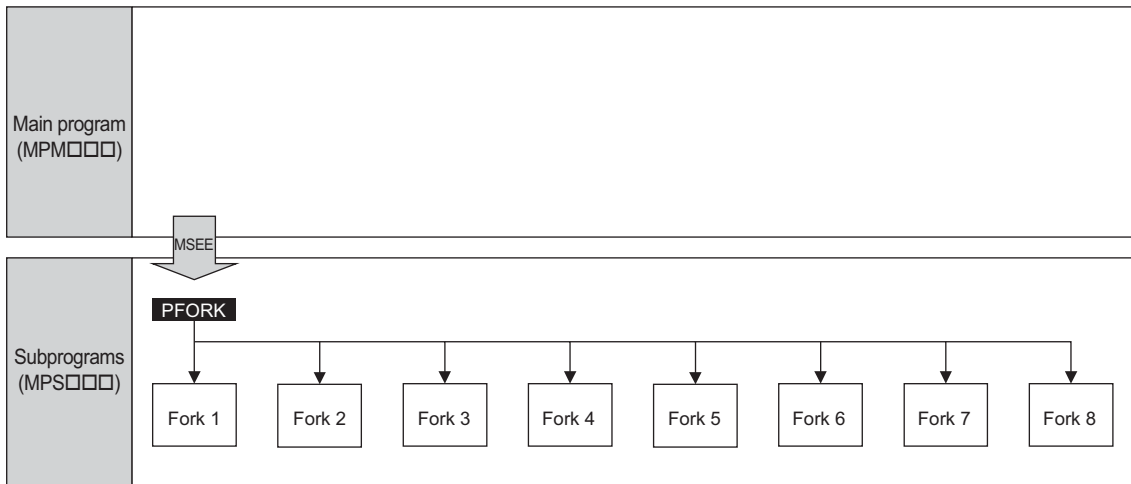
In this mode, up to two forks can be executed in parallel in a main program, and up to four forks can be executed in parallel in a subprogram.



Main 1 × Sub 8

This mode allows the parallel execution of up to 8 forks in a subprogram.

Parallel execution is not possible in main programs in this mode.

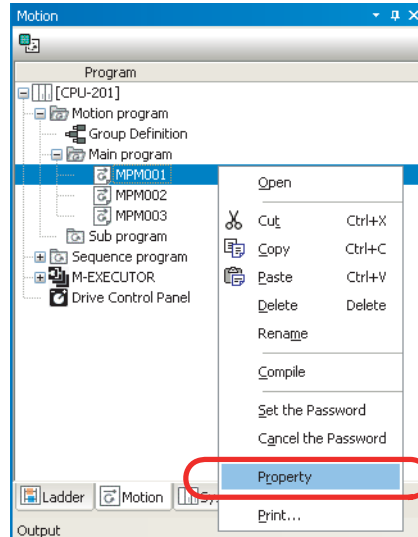


Setting the PFORK Parallel Execution Mode

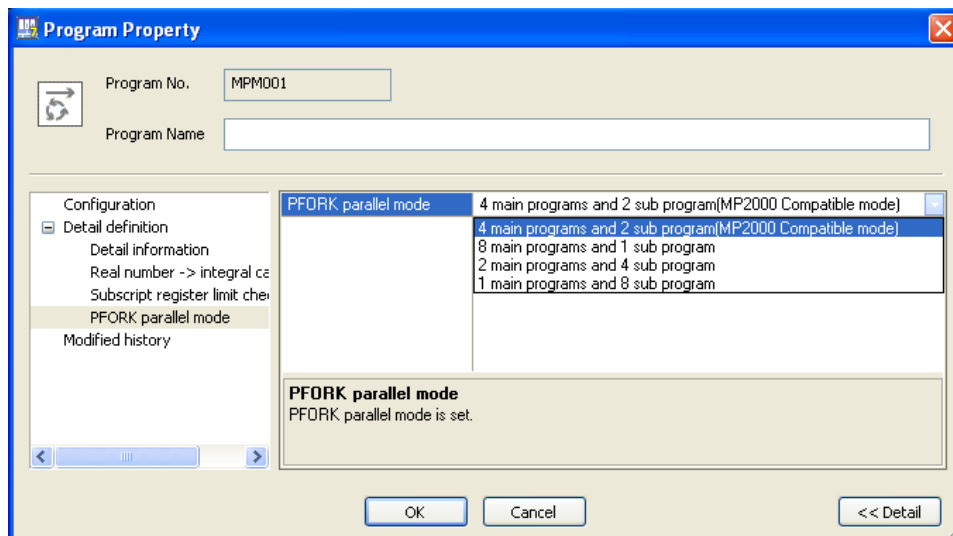
This section describes how to set the PFORK parallel execution mode.

The parallel execution mode is set in the Program Properties Dialog Box of the main program. By default, the parallel execution mode is set to Main 4 × Sub 2.

1. Right-click **MPM001** under **Motion Program - Main Program** in the Motion Pane and select **Properties** from the menu.



2. Select the parallel execution mode under **PFORK Parallel Execution Mode** in the Program Properties Dialog Box.



■ Timing at Which the Parallel Execution Mode Setting Becomes Valid


The parallel execution mode setting becomes valid as soon as the **OK** Button is clicked in the Program Properties Dialog Box.

Axis Alarm Checks


With an MP3000-series Machine Controller, you can check for alarms (IL□□□04) that can occur in axes specified in axis movement instructions in a motion program.

You can enable or disable these checks in the environment settings of MPE720 version 7.

Refer to the following manual for information on the environment settings of MPE720 version 7.

 *MP2000/MP3000 Series Engineering Tool MPE720 Version 7 User's Manual* (Manual No.: SIEP C880761 03)

Refer to the following appendix for details on checking for axis alarms.

 *Appendix C Differences between MP2000-series and MP3000-series Machine Controllers*

Checking for Axis Alarms (MP3000-series Standard Feature)

If an alarm occurs (IL□□□04 ≠ 0) for an axis specified in an axis movement instruction, a motion program alarm will occur, all axes will stop (OW□□□09 Bit 1 = ON), and NOP (OW□□□08 = 0) motion commands will be issued.

This is the default operation for the MP3000-series Machine Controllers.

Not Checking for Axis Alarms (MP2000-series Compatible)

Even if an alarm occurs (IL□□□04 ≠ 0) for an axis specified in an axis movement instruction, references continue for axes for which no alarm has occurred.

If you use this mode, implement interlocks externally.

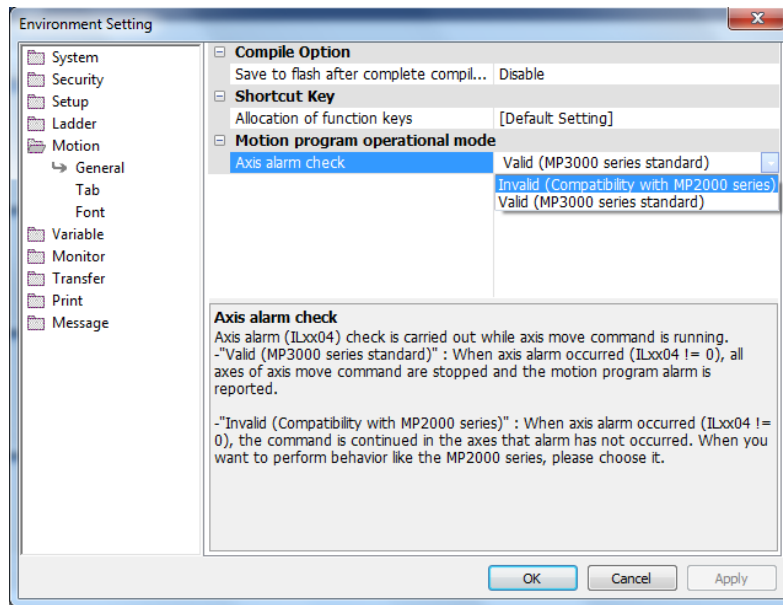
Information

This mode produces the same operation as the MP2000-series Machine Controllers. Select this mode if you are replacing an MP2000-series Machine Controller with an MP3000-series Machine Controller or want to use the same operation as the MP2000-series Machine Controller.

Procedure to Check for Axis Alarms

This section describes how to set the mode to check for axis alarms.

1. Select **Environment Setting** from the File Menu of the MPE720 Version 7 Window.
2. Select **Motion - General** in the Environment Setting Dialog Box.
3. Set **Axis Alarm Check** under **Motion Program Operation Mode** to **Check (MP3000-series Standard)**.



Combinations of MP3000-series Machine Controllers and MPE720 Version 7 Revisions

The following table shows the combinations of MP3000-series Machine Controller and MPE720 versions.

MP3000-series Software Version	MPE720 Version 7.21.0100 or Later		MPE720 Version 7.20.0100 or Earlier	
	Axis Alarm Check Selection	Axis Alarm Checks (Performed/Not Performed)	Axis Alarm Check Selection	Axis Alarm Checks (Performed/Not Performed)
MP3000-series Machine Controller Version 1.05 or Earlier	Cannot be selected.	Always performed.	Cannot be selected.	Always performed.
MP3000-series Machine Controller Version 1.06 or Later	Can be selected.	Performed by default.	Cannot be selected.	Always performed.

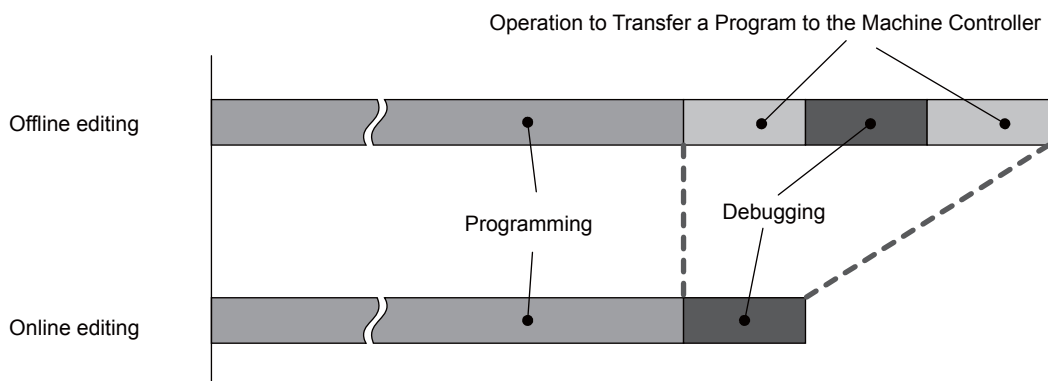
Information

If you enable axis alarm checks with an MP3000-series Machine Controller version 1.06 or later and MPE720 version 7.21.0100 or later and then save the setting to flash memory, the setting will be used in the future as well.

Online Editing of Programs

Motion programs can be edited online in the same way as ladder programs.

Online editing allows you to edit programs while you are logged onto the Machine Controller. In online editing mode, the edited program is automatically transferred to the Machine Controller when the program is saved. This helps save time through the elimination of any operations to manually transfer the program to the Machine Controller.



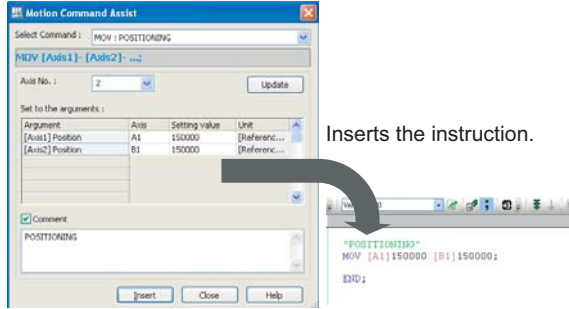
Information Online editing is not possible while execution of the motion program is in process.

Easy Programming Functions (MPE720 Version 7.0 or Later)

MPE720 Engineering Tool version 7.0 for MP3000-series Machine Controllers includes the following easy programming functions.

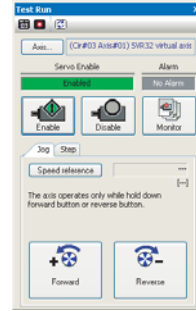
- Instruction Entry Assistance

You can simply select an instruction and set the data in the dialog box shown below to insert the instruction into the editor.



- Test Runs

You can control the axes from the following dialog box.



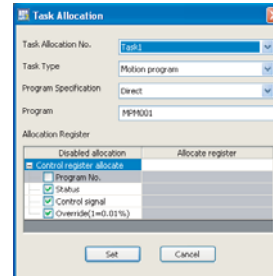
- Axis Monitor

You can view the operating status of each axis.

	Axis#01	SVR132	Axis#02	SVR132	virtual	Axis#03	SVR132	virtual	Axis#04	SVR132	virtual
Ready / Servo Enable	Ready	Disabled	Ready	Disabled	Not Ready	Disabled	Not Ready	Disabled	Not Ready	Disabled	Not Ready
Alarm / Warning	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm	No Alarm
Prof. Comp / In Position	Prof. Comp	In Position	Prof. Comp	In Position	Prof. Comp	In Position	Prof. Comp	In Position	Prof. Comp	In Position	In Position
Motion Command	STOP	STOP	STOP	STOP	STOP	STOP	STOP	STOP	STOP	STOP	STOP
Machine coordinate feedback	0	0	0	0	0	0	0	0	0	0	0
Position error (PERR)	0	0	0	0	0	0	0	0	0	0	0
Feedback speed	0	0	0	0	0	0	0	0	0	0	0
Feedback torque / thrust	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

- Task Assignments

You can easily register the programs to execute in the system.



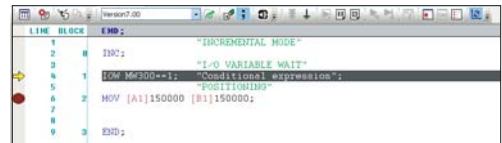
- Operation Control Panel

You can execute motion programs from the Motion Editor.

Task	Task1	Task2	Task3
Main program	MFM001	No allocate	No allocate
Motion Program Control Signals	SW0301 H0000	SW0302 H0000	SW0303 H0000
RE 0 : Start request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 1 : Pause request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 2 : Stop request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 3 : Single block mode selection	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 4 : Single block start request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 5 : Alarm reset request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 6 : Program continuous operation start request	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 8 : Signl information	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 9 : Signl information	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 0 : System work number setting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RE 1 : Interpolation override setting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Status	SW0300 H0000	SW0302 H0000	SW0303 H0000
RE 0 : Running	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

- Debugging

You can debug motion programs. Common debugging operations, such as step-by-step execution and setting breakpoints, are provided.

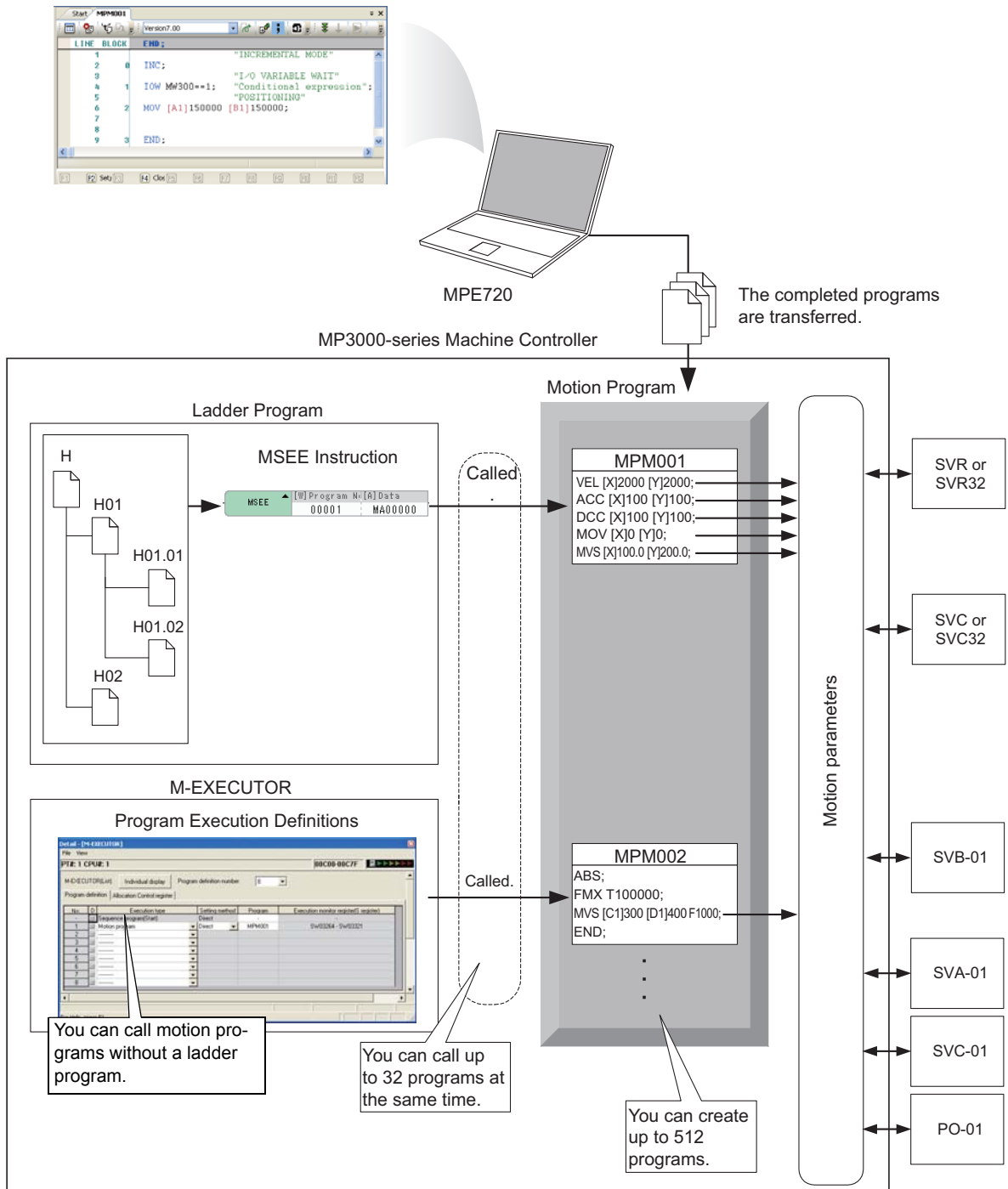


1.3 Motion Program System Configuration

Motion programs are transferred to the MP3000-series Machine Controller after they are created in the MPE720 Motion Editor. The transferred motion programs can be called with MSEE instructions in a ladder program, or from the M-EXECUTOR execution definitions. Motion language instructions are sent to the Motion Control Function Module through the motion parameters to operate the axes.

The following figure shows the system configuration of a motion program.

Programming Dialog Box



1.4

Types of Motion Programs

There are two types of motion programs. These are given in the following table.

Type	Designation Method	Features	Number of Programs
Main programs	MPM□□□ (□□□ = 1 to 512)	<ul style="list-style-type: none"> • Main programs can be called from the M-EXECUTOR program execution definitions. • Main programs can be called from a DWG.H drawing. 	You can create up to 512 motion programs, including the following programs: <ul style="list-style-type: none"> • Motion main programs • Motion subprograms • Sequence main programs • Sequence subprograms
Subprograms	MPS□□□ (□□□ = 1 to 512)	<ul style="list-style-type: none"> • Subprograms are called from a main program. 	



Important

1. Use a unique program number for each motion program and sequence program. If the same number is used more than once, an error will be displayed on the MPE720.
2. MP3000-series Machine Controllers can execute up to 32 motion programs simultaneously. If 33 or more programs are executed simultaneously, a motion program alarm occurs (No System Work Available Error).
 - The No System Work Available Error is indicated by bit E in the Status Flags of the motion program.


Information

In this manual, the high-speed process drawing for a ladder program is called DWG.H (high-speed drawing).

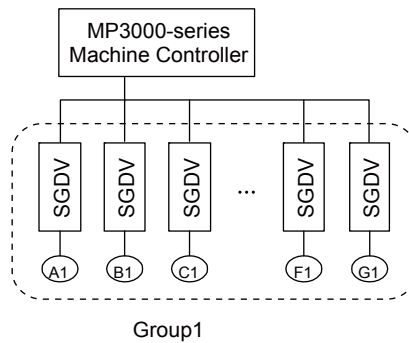
1.5 Motion Program Groups

With motion programs, the axes that have related operations are organized into individual groups. You can create programs for each group. Motion program groups allow a single Machine Controller to control multiple machines independently. A group operation can be an operation as a single group or an operation with multiple groups.

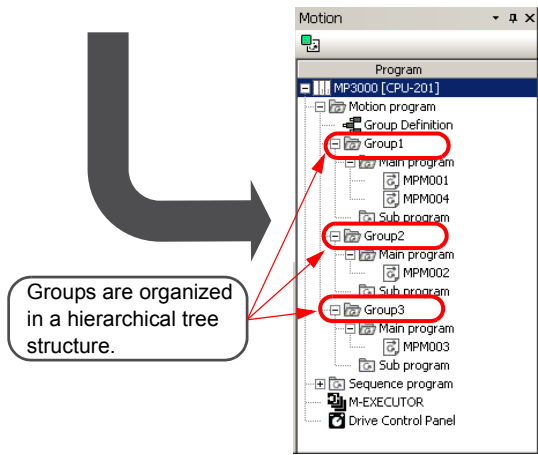
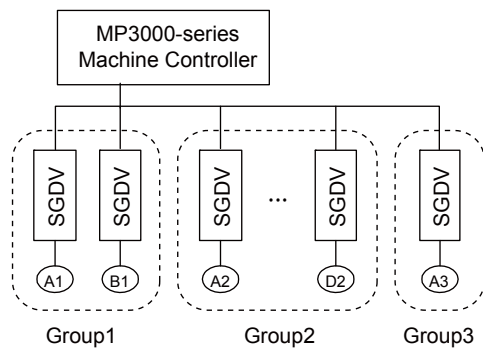
The definitions for axes to be grouped are made in the Group Definitions. Refer to the following section for the procedure to set group definitions.

 5.2 Group Definition Details (page 5-9)

■ Operation with One Group



■ Operation with Multiple Groups



1.6

Motion Program Execution Timing

The processing in a motion program is executed in full synchronization with the high-speed scan of the MP3000-series Machine Controller.

In the high-speed scan cycle, I/O services are performed first, and then the motion programs that are registered in the M-EXECUTOR are executed. Next, the motion programs that are called with the MSEE instructions that are programmed in DWG.H are executed when the individual MSEE instructions are executed.

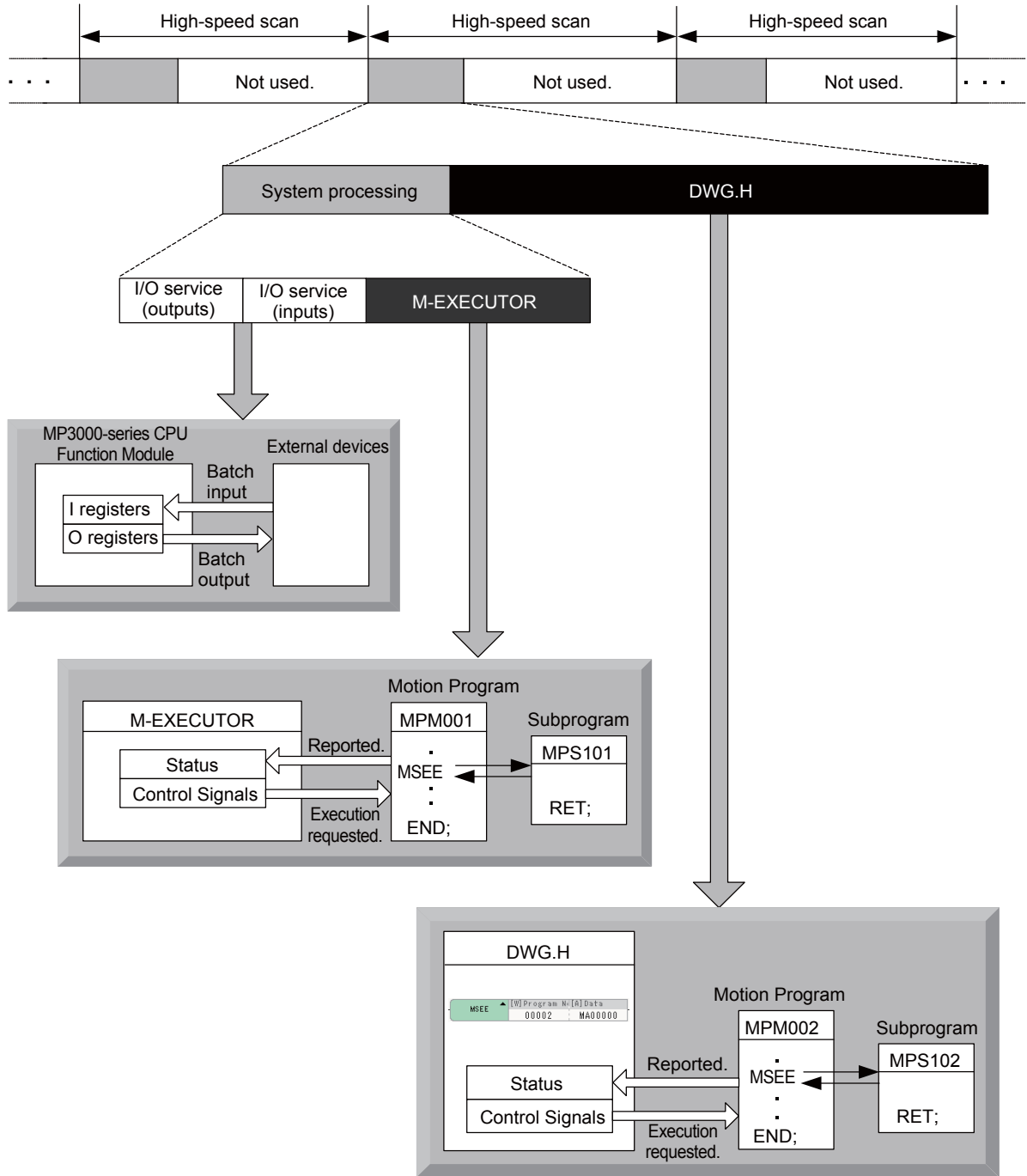


Terms

I/O Service

I/O services process the execution of data I/O between the MP3000-series CPU Unit/CPU Module and external devices (i.e., the Optional Modules).

The following figure shows the execution timing of a motion program.



1.7

Executing Motion Programs

This section describes how to execute motion programs.

Execution Processing Method

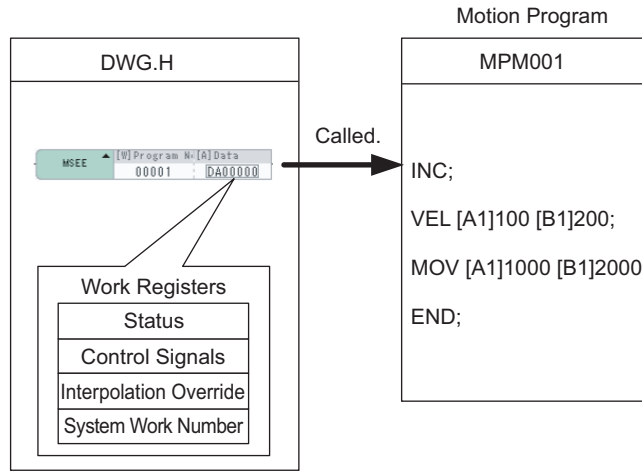
You must register the motion programs that you create in the system to execute them. The motion programs that are registered in the system are called in the high-speed scan cycle.

There are two execution methods that you can use when a motion program is registered in the system for execution.

- Calling the motion program from a ladder program with an MSEE instruction
- Calling the motion program using the M-EXECUTOR program execution definitions

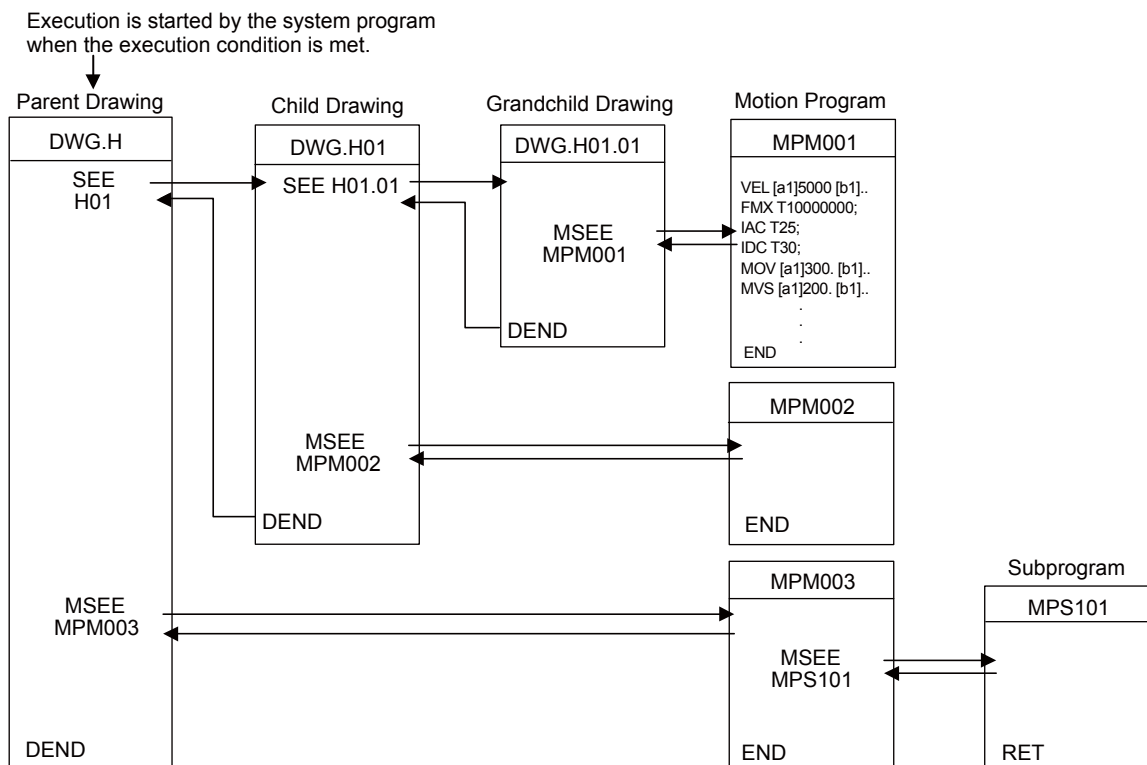
Calling the Motion Programs from a Ladder Program with an MSEE Instruction

After you create the motion program, insert an MSEE (Call Motion Program) instruction in the high-speed drawing.



Motion programs can be called from any H drawing, regardless of whether it is a parent, child, or grandchild drawing.

The following figure shows an execution example.



The ladder instructions in the high-speed drawing are executed every high-speed scan cycle according to the hierarchical organization of parent-child-grandchild drawings.

The above programming only prepares for execution of the motion program. The motion program is not executed at the location where the MSEE instruction is inserted. To execute the motion program, use a control signal to turn ON the request for start of program operation after the MSEE instruction has been inserted.

The motion program is executed in the scan cycle, but unlike ladder programs, the entire program is not executed in a single scan. Execution of motion programs is controlled by the system.



Note

Observe the following precautions when executing motion programs:

- Motion programs that are registered in the M-EXECUTOR cannot be executed with MSEE instructions.
- More than one instance of the same motion program (i.e., the same program number) cannot be executed with MSEE instructions.
- Subprograms (MPS□□□) cannot be executed with MSEE instructions in ladder programs. You can call subprograms from other motion programs (MPM□□□ and MPS□□□) only.
- Sequence programs (SPM□□□ or SPS□□□) cannot be called with MSEE instructions from ladder programs.

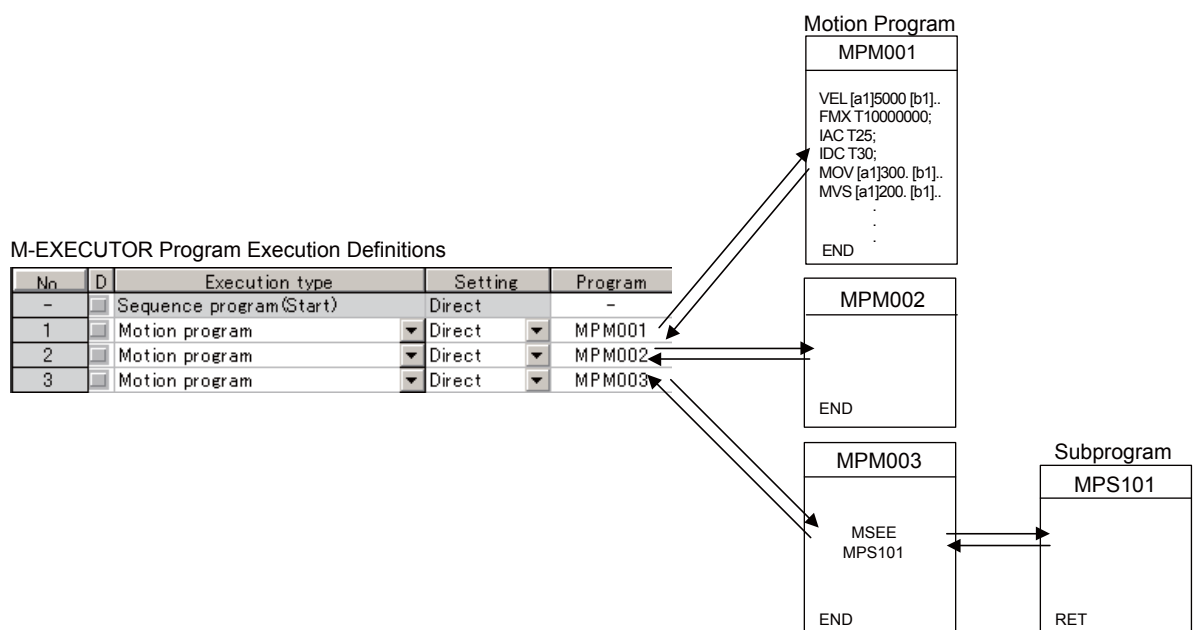
Calling the Motion Programs Using the M-EXECUTOR Program Execution Definitions

After creating a motion program, register it in the M-EXECUTOR program execution definitions.

Control registers (I/O registers) are used to start and stop the registered motion programs.

Programs registered in the M-EXECUTOR program execution definitions are executed in ascending numeric order.

The following figure shows an execution example.



To execute a motion program, first register the program in the M-EXECUTOR program execution definitions, then use a control signal to turn ON the request for start of program operation.

A motion program that is registered in the M-EXECUTOR program execution definitions is executed in the scan cycle, but unlike a ladder program, the entire program is not executed in a single scan. Execution of motion programs is controlled by the system.



Note

Observe the following precautions when registering motion programs in the M-EXECUTOR program execution definitions:

- Each motion program must be registered with a unique program number.
- More than one motion program with the same number cannot be called using indirect designation.

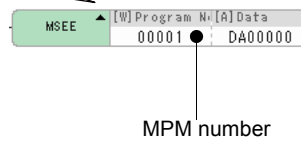
Program Execution Registration Methods

There are two methods to register a program for execution.

The following examples demonstrate how to register motion program MPM001 for execution.

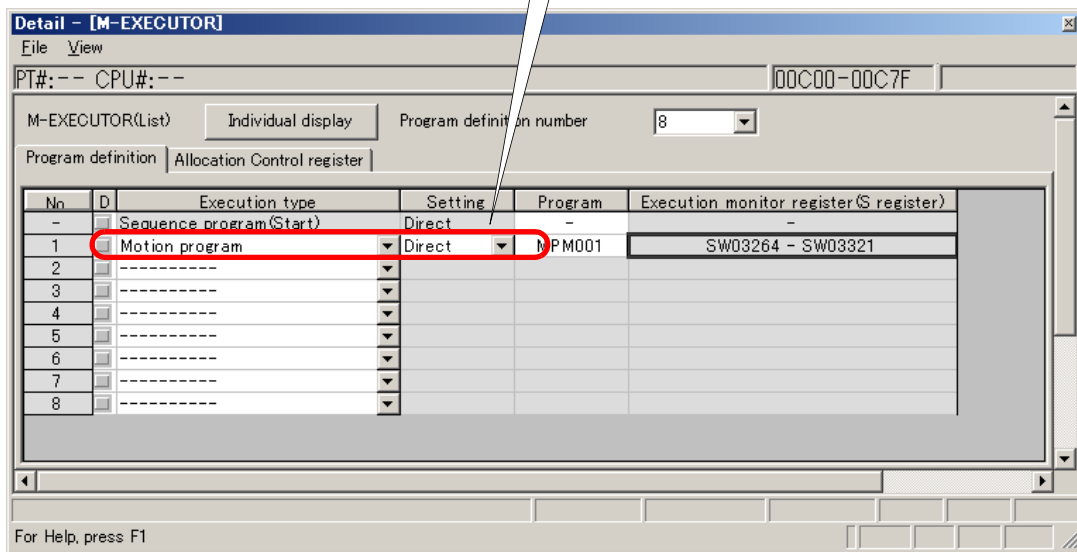
Inserting an MSEE Instruction into a Ladder Program

Program an MSEE instruction in the high-speed drawing.
Program the MSEE instruction so that it is executed every scan.



Registering Motion Programs in the M-EXECUTOR Program Execution Definitions

Register the program to execute.




Work Registers

When a program is registered for execution, that program is assigned work registers to control and monitor the execution of the program. The work registers are used to send commands to the motion programs from the motion program control program, and to get the motion program status.

■ When a Motion Program Is Called from a Ladder Program with an MSEE Instruction

Four registers (words) starting from the register that is specified with the Data parameter of the MSEE instruction (MA□□□□□□ or DA□□□□□□) are used as work registers.



Work Register	Register Address in the Example	Contents	I/O
1st register	DW00000	Status Flags	OUT
2nd register	DW00001	Control Signals	IN
3rd register	DW00002	Interpolation Override	IN
4th register	DW00003	System Work Number	IN

■ When the Motion Program Is Registered in the M-EXECUTOR Program Execution Definitions

The M-EXECUTOR control registers are used as the work registers.

The M-EXECUTOR control registers are automatically assigned by the system.

Program definition		Allocation Control register
No.	Item	M-EXECUTOR Control register
1	Program number	MPM001
	Status	IW00C00
	Control signal	OW00C01
	Override	OW00C02

Work Register (M-EXECUTOR Control Register)	Register Address in the Example	Contents	I/O
Status	IW00C00	Status Flags	OUT
Control Signals	OW00C01	Control Signals	IN
Override	OW00C02	Interpolation Override	IN

Status Flags

The Motion Program Status Flags give the execution condition of the motion program.

The following table describes the meanings of the Status Flags.

Bit No	Name	Contents
Bit 0	Program Executing	This bit is set to 1 when a motion program is running. 0: Motion program is stopped. 1: Motion program is running.
Bit 1	Program Paused	This bit is set to 1 when execution of a motion program is paused by a Request for Pause of Program. After a Request for Pause of Program control signal is input, it is confirmed that the axis decelerated to a stop and then the status flag is turned ON. 0: Program is not stopped by a pause request. 1: Program is stopped by a pause request.
Bit 2	Program Stopped for Request for Stop Request	This bit is set to 1 when execution of a motion program is stopped by a Request for Stop of Program. 0: Program is not stopped by a stop request. 1: Program is stopped by a stop request.
Bit 3	(Reserved for system.)	–
Bit 4	Program Single-block Execution Stopped	This bit is set to 1 when execution of a single block is stopped in Debug Operation Mode. 0: Single block execution is not stopped. 1: Single block execution is stopped.
Bit 5	(Reserved for system.)	–
Bit 6	(Reserved for system.)	–
Bit 7	(Reserved for system.)	–
Bit 8	Program Alarm	This bit is set to 1 when a program alarm occurs. When this bit is set to 1, details on the error will be displayed in the Error Information Dialog Box and are given in the S registers. 0: There is no program alarm. 1: A program alarm occurred.
Bit 9	Program Stopped at Breakpoint	This bit is set to 1 when execution of a program stops at a breakpoint in Debug Operation Mode. 0: Not stopped at a breakpoint. 1: Stopped at a breakpoint.
Bit A	(Reserved for system.)	–
Bit B	Debug Operation Mode	This bit is set to 1 when a program is running in Debug Operation Mode. 0: Not in Debug Operation Mode (Normal Execution Mode). 1: In Debug Operation Mode.
Bit C	Program Type	This bit reports whether the program that is being executed is a motion program or a sequence program. 0: Motion program 1: Sequence program
Bit D	Start Request History	This bit is set to 1 when the Request for Start of Program Operation is ON. 0: Turn OFF the request to start the program. 1: Turn ON the request to start the program.
Bit E	No System Work Available Error or Execution Scan Error	This bit is set to 1 when a system work number that was needed to execute a motion program could not be obtained, or when an MSEE instruction is programmed in a drawing other than a DWG.H. 0: There is no system work available error or execution scan error. 1: A no system work available error or execution scan error occurred.

Continued on next page.


Continued from previous page.

Bit No	Name	Contents
Bit F	Main Program Number Limit Exceeded Error	This bit is set to 1 when the specified motion program number is out of range. Motion program number range: 1 to 512 0: There is no motion program number error. 1: A motion program number error occurred.













Note: If a motion program alarm occurs, program error information is provided in the Error Information Dialog Box and in the S registers.

Control Signals

To control the execution of a motion program, you must input program control signals (Request for Start of Program Operation, or Request for Stop of Program, etc.). The following table describes the control signals for motion programs.

: This symbol indicates that the signal must be kept ON until the system acknowledges it.

: This symbol indicates that the signal needs to be turned ON only for one high-speed scan.

Bit No	Name	Description
Bit 0  	Request for Start of Program Operation	This bit makes a request to start execution of a motion program. The motion program starts when this bit changes from 0 to 1. This bit is ignored when there is a program alarm. 0: Turn OFF the request to start the program. 1: Turn ON the request to start the program.
Bit 1 	Request for Pause of Program	This bit makes a request to pause execution of a motion program. Execution of the program that was paused will resume when the pause request is turned OFF. 0: Turn OFF the request to pause the program (i.e., cancels the pause). 1: Turn ON the request to pause the program.
Bit 2 	Request for Stop of Program	This bit makes a request to stop execution of a motion program. A motion program alarm occurs if this bit is set to 1 while the axis is in motion. 0: Turn OFF the request to stop the program. 1: Turn ON the request to stop the program.
Bit 3 	Program Single-block Mode Selection	This bit makes a request to select Program Single-block Execution Mode. This mode can be used in place of Debug Operation Mode. 0: Deselect single block mode. 1: Select single block mode.
Bit 4  	Program Single-block Start Request	When this bit is changed from 0 to 1, program execution changes to single-block execution (step execution). This bit is only valid when bit 3 (Program Single-block Mode Selection) in the control signal word is set to 1. 0: Turn OFF the request to start a single program block. 1: Turn ON the request to start a single program block.
Bit 5 	Program Reset and Alarm Reset Request	This bit resets motion programs and alarms. 0: Turn OFF the request to reset the program and alarms. 1: Turn ON the request to reset the program and alarms.
Bit 6  	Request for Start of Continuous Program Operation	This bit makes a request to resume execution of a program that was stopped by a Request for Stop of Program. 0: Turn OFF the request to resume the program. 1: Turn ON the request to resume the program.
Bit 7	(Reserved for system.)	—
Bit 8 	Skip 1 Information	If this bit changes to 1 while an axis is in motion due to a SKP instruction (when the skip input signal selection is set to SS1), the axis will decelerate to a stop, and the reference in the remaining travel distance will be canceled. 0: Turn OFF the skip 1 signal. 1: Turn ON the skip 1 signal.
Bit 9 	Skip 2 Information	If this bit changes to 1 while an axis is in motion due to a SKP instruction (when the skip input signal selection is set to SS2), the axis will decelerate to a stop, and the reference in the remaining travel distance will be canceled. 0: Turn OFF the skip 2 signal. 1: Turn ON the skip 2 signal.

Continued on next page.

Continued from previous page.

Bit No	Name	Description
Bit A, B	(Reserved for system.)	–
Bit C	(Reserved for system.)	–
Bit D — —	System Work Number Setting*1	To specify a system work number, set this bit to 1. 0: Do not specify a system work number. 1: Specify a system work number.
Bit E — —	Interpolation Override Setting*2	To specify an interpolation override, set this bit to 1. 0: Do not specify an interpolation override. 1: Specify an interpolation override.
Bit F	(Reserved for system.)	–

*1. System Work Number Setting

- When the Motion Program Is Registered in M-EXECUTOR:
The system work number cannot be specified. The system will use the definition number as the system work number.
- When a Motion Program Is Called from a Ladder Program with an MSEE Instruction:
OFF: The system will use an automatically acquired system work number. The system work number will be different each time.
ON: The work number that is specified by the system will be used.
However, if the work number is assigned to the M-EXECUTOR, a No System Work Available Error (Status Flag Bit E) is reported.

*2. Interpolation Override Setting

- OFF: The interpolation override is always 100%.
- ON: The interpolation override in the parameter setting is used.

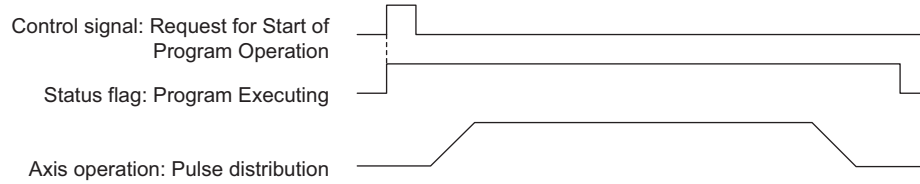
Note: 1. Use the specified signal types for the ladder program inputs.

- At startup, the motion programs for which the Request for Start of Program Operation control signals are ON will be executed.

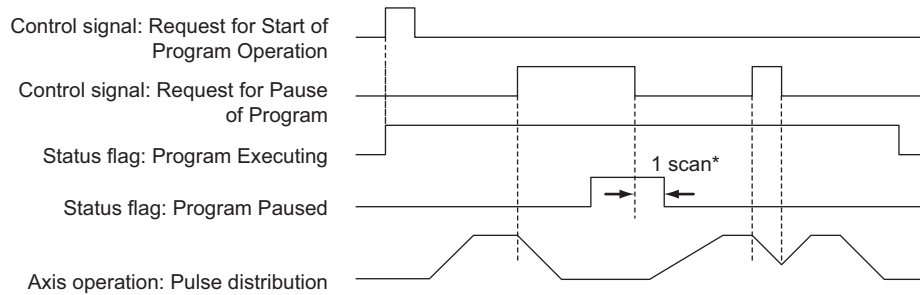
■ Motion Program Control Signals Timing Chart

Timing chart examples for axis operations and status flags after a control signal is input are provided below.

• Request for Start of Program Operation

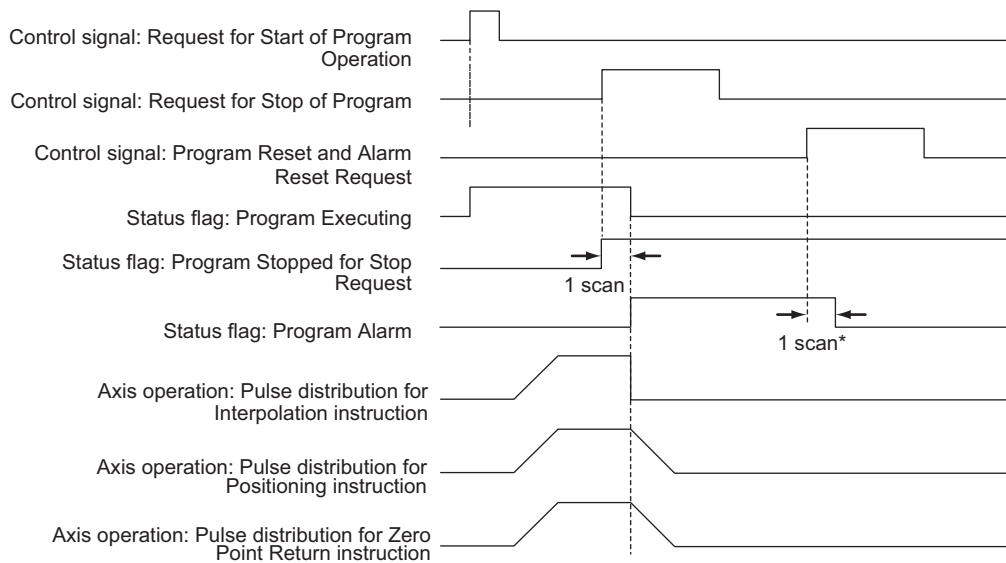


• Request for Pause



* Status flags related to control signal input are updated after one scan.

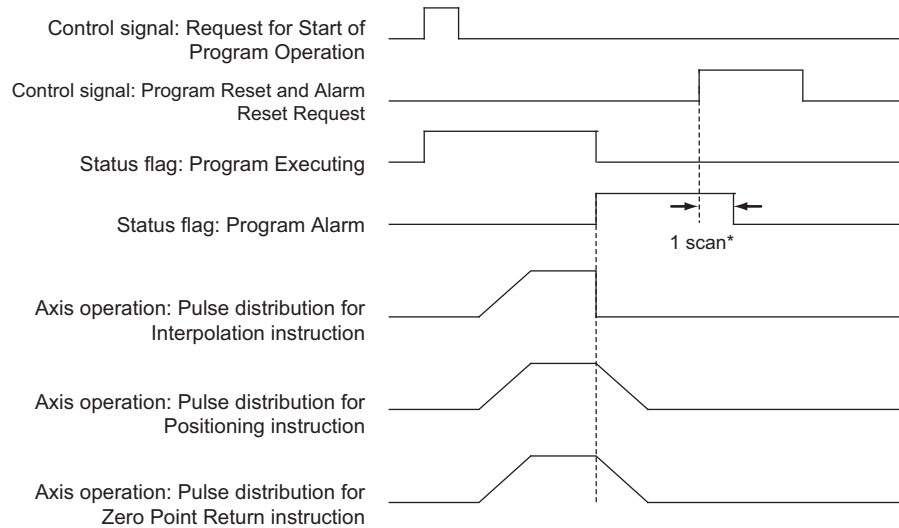
• Request for Stop



* Status flags related to control signal input are updated after one scan.

When restarting operation of a program that has been stopped by the Request for Stop of Program control signal, execute the Request for Start of Continuous Program Operation control signal instead of executing the Program Reset and Alarm Reset Request control signal.

• If a Motion Program Alarm Occurs




* Status flags related to control signal input are updated after one scan.



Important

1. If the Request for Stop of Program control signal is turned ON while the axis is being controlled for a motion language instruction, an alarm will occur.
2. If the Request for Stop of Program control signal is turned ON while the axis is being controlled for an interpolation motion language instruction, the axes will stop immediately. To perform a deceleration stop, use the Request for Pause of Operation control signal.
3. The Request for Pause of Program control signal is not acknowledged while a Zero Point Return (ZRN) instruction is being executed. To stop the operation, use the Request for Stop of Program control signal.
4. If a motion program alarm occurs while an axis is in motion, the axis stops immediately.

Refer to the following section for programming examples for motion program control.

 [B.1 Motion Program Control Program \(page B-2\)](#)

Interpolation Override

An interpolation override allows you to change the output ratio of the axis movement speed reference for interpolation motion language instructions.

Set the override value to use when executing interpolation instructions (MVS, MCW, MCC, or SKP).

The interpolation override is valid only when bit E (Interpolation Override Setting) in the control signals is ON.

The setting range of the interpolation override is 0 to 32,767.

Unit: 1 = 0.01%

System Work Number

When you call a motion program from a ladder program with the MSEE instruction, set the system work number to use to call the motion program. This system work number is valid only when bit D (System Work Number Setting) of the control signals is ON.

Setting range: 1 to 32



Important

When using MSEE instructions in ladder programs along with the M-EXECUTOR, do not specify the system work numbers that are for the M-EXECUTOR in the MSEE instructions in the ladder programs. If you specify one, a No System Work Available Error will occur.

System work numbers for the M-EXECUTOR: 0 to the set value of the number of program definitions

Information

You cannot set the system work numbers when you use the M-EXECUTOR. The system will use system work numbers that are the same as the definition numbers.

1.8 Advanced Programming

This section describes practical methods of executing motion programs.

Indirect Designation of a Program Number Using a Register

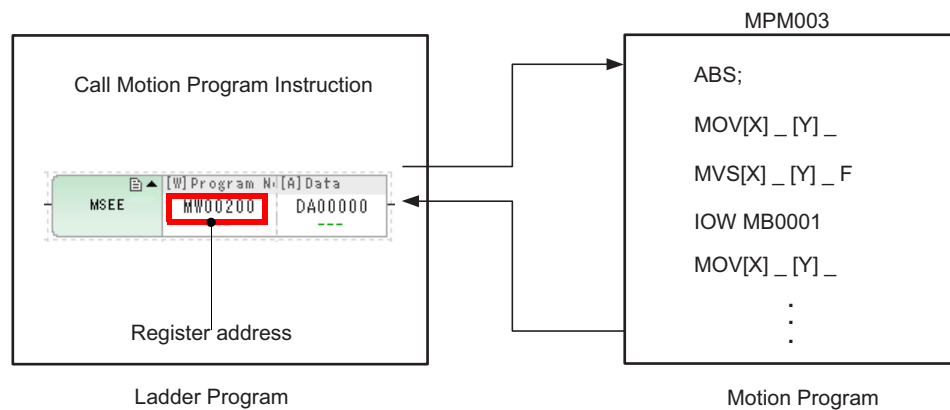
You can use a register to call a motion program with value stored in that register.

There are two methods of calling a motion program in this way.

When a Motion Program Is Called from a Ladder Program with an MSEE Instruction

Specify the register (M, G, or D register) to use for the indirect designation in the Program No. parameter of the MSEE instruction.

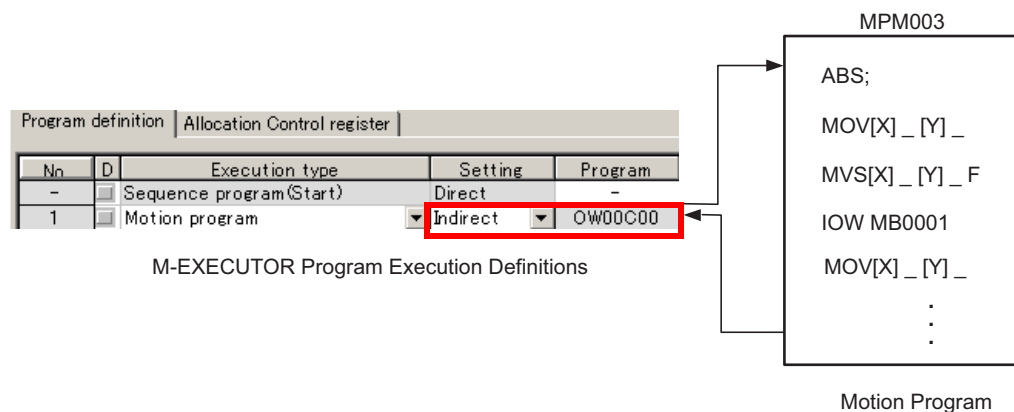
If a value of 3 is stored in MW00200, the MPM003 program is called.



Calling Motion Program with the M-EXECUTOR Program Execution Definitions

Select the indirect designation method in the **Setting** Column. The register used for indirect designation is assigned automatically by the system.

If a value of 3 is stored in OW00C00, the MPM003 program is called.



Controlling Motion Programs Directly from an External Device

The M-EXECUTOR allows you to assign M-EXECUTOR control registers.

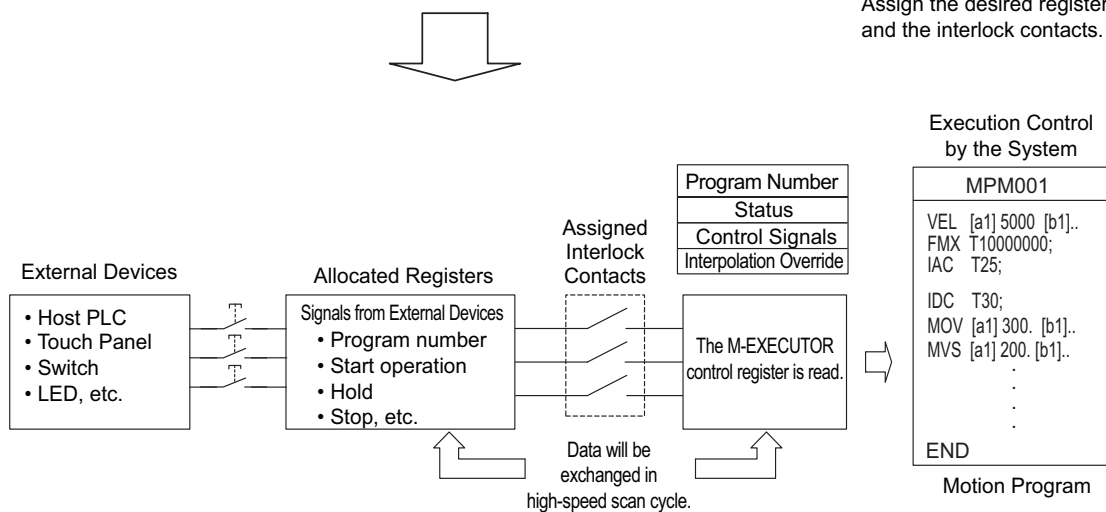
This can be used to automatically exchange data between an M-EXECUTOR control register and the I/O register connected to an external device.

The following are sample settings to directly control a motion program from an external device.

M-EXECUTOR Register Assignment Dialog Box

No.	Item	M-EXECUTOR Control register	Allocation Disable	Direction	Allocation register	Allocation Contact interlock
1	Program number	MPM001				
	Status	IW00C00	<input type="checkbox"/>	->	OW00000	IB000020
	Control signal	OW00C01	<input type="checkbox"/>	<-	IW00001	IB000020
	Override	OW00C02	<input type="checkbox"/>	<-	IW00002	IB000020

Assign the desired registers and the interlock contacts.



The assigned interlock contacts are used to interlock motion program operation. If you assign registers, always assign interlock contacts.

The following processes are performed according to the ON/OFF status of the assigned interlock contacts.

- When the assigned interlock contact is ON, the assigned register exchanges data with the M-EXECUTOR control register in the high-speed scan cycle. Motion program execution is enabled during this data exchange.
- When the assigned interlock contact is OFF, the assigned register does not exchange data with the M-EXECUTOR control register, and motion program execution is disabled.
- If the assigned interlock contact changes from ON to OFF while the motion program is running, the motion program stops and the axes stop moving. At this moment, the 1B hex motion program alarm (Emergency Stop in Progress) occurs, and bit 8 (Program Alarm) in the status flags turns ON.

Use the following procedure to restart the motion program.

1. Turn ON the assigned interlock contact.
2. Turn ON bit 5 (Program Reset and Alarm Reset Request) in the control signals.
3. Confirm that bit 8 (Program Alarm) in the status flags turns OFF.
4. Turn OFF bit 5 (Program Reset and Alarm Reset Request) in the control signals.
5. Turn ON bit 0 (Request for Start of Program Operation) in the control signals.

Monitoring Motion Program Execution Information

The execution information for motion programs can be monitored using the S registers (SW03200 to SW05119 and SL08192 to SL09214).

The execution information is monitored differently, depending on whether the motion program is called from a ladder program with an MSEE instruction, or the motion program is registered in the M-EXECUTOR program execution definitions.

This section describes these two monitoring methods.

When a Motion Program Is Called from a Ladder Program with an MSEE Instruction

When a motion program is called from a ladder program with an MSEE instruction, the effects of the setting of bit D (System Work Number Setting) in the motion program control signal are as follows:

- When bit D (System Work Number Setting) in the Motion Program Control Signal is ON, the execution information is reported in the Work n Program Information registers (SW03264 to SW05119 and SL08192 to SL09214).

For example, if the system work number is 1, you can monitor the execution information of the motion program with the Work 1 Program Information registers (SW03264 to SW03321 and SL08192 to SL08222).

- When bit D (System Work Number Setting) in the Motion Program Control Signal is OFF, the system work number that is used is determined automatically by the system. You can check the work numbers that are in use in the Active Program Numbers registers (SW03200 to SW03231).

For example, if MPM001 is the motion program to be monitored and SW03202 contains a 1, the system work number is 3. You can therefore monitor the execution information of the motion program with the Work 3 Program Information registers (SW03380 to SW03437 and SL08256 to SL08286).

When Execution the Motion Program Is Registered in the M-EXECUTOR Program Execution Definitions

When the motion program is registered in the M-EXECUTOR program execution definitions, the system work number used will be the same as the program execution registration number in the M-EXECUTOR.

For example, if the motion program is registered for execution as number 3, system work number 3 is used. You can therefore monitor the execution information of the motion program with the Work 3 Program Information registers (SW03380 to SW03437 and SL08256 to SL08286).

◆ Register Ranges for Motion Program Execution Information

		Active Program Numbers		
SW03200	Numbers of Currently Executing Main Programs 32W	SW03200	Number of Program Using Work Number 1	
SW03201		Number of Program Using Work Number 2		
SW03202	Program Execution Bits Executing while the corresponding bit is ON. 32W	SW03202	Number of Program Using Work Number 3	
SW03203		Number of Program Using Work Number 4		
SW03204		Number of Program Using Work Number 5		
SW03205		Number of Program Using Work Number 6		
SW03206		Number of Program Using Work Number 7		
SW03207		Number of Program Using Work Number 8		
SW03208		Number of Program Using Work Number 9		
SW03209		Number of Program Using Work Number 10		
SW03210		Number of Program Using Work Number 11		
SW03211		Number of Program Using Work Number 12		
SW03212		Number of Program Using Work Number 13		
SW03213		Number of Program Using Work Number 14		
SW03214		Number of Program Using Work Number 15		
SW03215		Number of Program Using Work Number 16		
SW03216		Number of Program Using Work Number 17		
SW03217		Number of Program Using Work Number 18		
SW03218		Number of Program Using Work Number 19		
SW03219		Number of Program Using Work Number 20		
SW03220		Number of Program Using Work Number 21		
SW03221		Number of Program Using Work Number 22		
SW03222		Number of Program Using Work Number 23		
SW03223		Number of Program Using Work Number 24		
SW03224		Number of Program Using Work Number 25		
SW03225		Number of Program Using Work Number 26		
SW03226		Number of Program Using Work Number 27		
SW03227		Number of Program Using Work Number 28		
SW03228		Number of Program Using Work Number 29		
SW03229		Number of Program Using Work Number 30		
SW03230		Number of Program Using Work Number 31		
SW03231		Number of Program Using Work Number 32		
SW03264		Work 1 Program Information 58W	Program Execution Bits MP□016(Bit F) to MP□001(Bit 0)	
SW03322		Work 2 Program Information 58W		
SW03380	Work 3 Program Information 58W			
SW03438	Work 4 Program Information 58W			
SW03496	Work 5 Program Information 58W			
SW03554	Work 6 Program Information 58W			
SW03612	Work 7 Program Information 58W			
SW03670	Work 8 Program Information 58W			
SW03728	Work 9 Program Information 58W			
SW03786	Work 10 Program Information 58W			
SW03844	Work 11 Program Information 58W			
SW03902	Work 12 Program Information 58W			
SW03960	Work 13 Program Information 58W			
SW04018	Work 14 Program Information 58W			
SW04076	Work 15 Program Information 58W			
SW04134	Work 16 Program Information 58W			
SW04192	Work 17 Program Information 58W			
SW04250	Work 18 Program Information 58W			
SW04308	Work 19 Program Information 58W			
SW04366	Work 20 Program Information 58W			
SW04424	Work 21 Program Information 58W			
SW04482	Work 22 Program Information 58W			
SW04540	Work 23 Program Information 58W			
SW04598	Work 24 Program Information 58W			
SW04656	Work 25 Program Information 58W			
SW04714	Work 26 Program Information 58W			
SW04772	Work 27 Program Information 58W			
SW04830	Work 28 Program Information 58W			
SW04888	Work 29 Program Information 58W			
SW04946	Work 30 Program Information 58W			
SW05004	Work 31 Program Information 58W			
SW05062	Work 32 Program Information 58W			
SW03232	MP□016(Bit F) to MP□001(Bit 0)	SW03232	MP□016(Bit F) to MP□001(Bit 0)	
SW03233	MP□032(Bit F) to MP□017(Bit 0)	SW03233	MP□032(Bit F) to MP□017(Bit 0)	
SW03234	MP□048(Bit F) to MP□033(Bit 0)	SW03234	MP□048(Bit F) to MP□033(Bit 0)	
SW03235	MP□064(Bit F) to MP□049(Bit 0)	SW03235	MP□064(Bit F) to MP□049(Bit 0)	
SW03236	MP□080(Bit F) to MP□065(Bit 0)	SW03236	MP□080(Bit F) to MP□065(Bit 0)	
SW03237	MP□096(Bit F) to MP□081(Bit 0)	SW03237	MP□096(Bit F) to MP□081(Bit 0)	
SW03238	MP□112(Bit F) to MP□097(Bit 0)	SW03238	MP□112(Bit F) to MP□097(Bit 0)	
SW03239	MP□128(Bit F) to MP□113(Bit 0)	SW03239	MP□128(Bit F) to MP□113(Bit 0)	
SW03240	MP□144(Bit F) to MP□129(Bit 0)	SW03240	MP□144(Bit F) to MP□129(Bit 0)	
SW03241	MP□160(Bit F) to MP□145(Bit 0)	SW03241	MP□160(Bit F) to MP□145(Bit 0)	
SW03242	MP□176(Bit F) to MP□161(Bit 0)	SW03242	MP□176(Bit F) to MP□161(Bit 0)	
SW03243	MP□192(Bit F) to MP□177(Bit 0)	SW03243	MP□192(Bit F) to MP□177(Bit 0)	
SW03244	MP□208(Bit F) to MP□193(Bit 0)	SW03244	MP□208(Bit F) to MP□193(Bit 0)	
SW03245	MP□224(Bit F) to MP□209(Bit 0)	SW03245	MP□224(Bit F) to MP□209(Bit 0)	
SW03246	MP□240(Bit F) to MP□225(Bit 0)	SW03246	MP□240(Bit F) to MP□225(Bit 0)	
SW03247	MP□256(Bit F) to MP□241(Bit 0)	SW03247	MP□256(Bit F) to MP□241(Bit 0)	
SW03248	MP□272(Bit F) to MP□257(Bit 0)	SW03248	MP□272(Bit F) to MP□257(Bit 0)	
SW03249	MP□288(Bit F) to MP□273(Bit 0)	SW03249	MP□288(Bit F) to MP□273(Bit 0)	
SW03250	MP□304(Bit F) to MP□289(Bit 0)	SW03250	MP□304(Bit F) to MP□289(Bit 0)	
SW03251	MP□320(Bit F) to MP□305(Bit 0)	SW03251	MP□320(Bit F) to MP□305(Bit 0)	
SW03252	MP□336(Bit F) to MP□321(Bit 0)	SW03252	MP□336(Bit F) to MP□321(Bit 0)	
SW03253	MP□352(Bit F) to MP□337(Bit 0)	SW03253	MP□352(Bit F) to MP□337(Bit 0)	
SW03254	MP□368(Bit F) to MP□353(Bit 0)	SW03254	MP□368(Bit F) to MP□353(Bit 0)	
SW03255	MP□384(Bit F) to MP□369(Bit 0)	SW03255	MP□384(Bit F) to MP□369(Bit 0)	
SW03256	MP□400(Bit F) to MP□385(Bit 0)	SW03256	MP□400(Bit F) to MP□385(Bit 0)	
SW03257	MP□416(Bit F) to MP□401(Bit 0)	SW03257	MP□416(Bit F) to MP□401(Bit 0)	
SW03258	MP□432(Bit F) to MP□417(Bit 0)	SW03258	MP□432(Bit F) to MP□417(Bit 0)	
SW03259	MP□448(Bit F) to MP□433(Bit 0)	SW03259	MP□448(Bit F) to MP□433(Bit 0)	
SW03260	MP□464(Bit F) to MP□449(Bit 0)	SW03260	MP□464(Bit F) to MP□449(Bit 0)	
SW03261	MP□480(Bit F) to MP□465(Bit 0)	SW03261	MP□480(Bit F) to MP□465(Bit 0)	
SW03262	MP□496(Bit F) to MP□481(Bit 0)	SW03262	MP□496(Bit F) to MP□481(Bit 0)	
SW03263	MP□512(Bit F) to MP□497(Bit 0)	SW03263	MP□512(Bit F) to MP□497(Bit 0)	

Note: The □ in MP□ for registers SW03232 to SW03263 is either an M or an S.

◆ Registers Used for System Work Numbers 1 to 32

The registers that are used for system work numbers 1 to 32 are given in the following table. Two system registers are given in the register table for the alarm code, but we recommend that you use system registers SL26□□□. You can use the system registers that are given in parentheses to check for alarms in most cases, but they do not report all alarms.

Refer to the following manual for details on alarm codes.

📖 *MP3000 Series MP3200/MP3300 Troubleshooting Manual* (Manual No.: SIEP C880725 01)

• System Work Numbers 1 to 8

System Work Number	Work 1	Work 2	Work 3	Work 4	Work 5	Work 6	Work 7	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	SL26000 (SW03268)	SL26016 (SW03326)	SL26032 (SW03384)	SL26048 (SW03442)	SL26064 (SW03500)	SL26080 (SW03558)	SL26096 (SW03616)	SL26112 (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	SL26002 (SW03271)	SL26018 (SW03329)	SL26034 (SW03387)	SL26050 (SW03445)	SL26066 (SW03503)	SL26082 (SW03561)	SL26098 (SW03619)	SL26114 (SW03677)
Fork 2	Program Number	SW03272	SW03330	SW03388	SW03446	SW03504	SW03562	SW03620	SW03678
	Block Number	SW03273	SW03331	SW03388	SW03447	SW03505	SW03563	SW03621	SW03679
	Alarm Code	SL26004 (SW03274)	SL26020 (SW03332)	SL26036 (SW03390)	SL26052 (SW03448)	SL26068 (SW03506)	SL26084 (SW03564)	SL26100 (SW03622)	SL26116 (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	SL26006 (SW03277)	SL26022 (SW03335)	SL26038 (SW03393)	SL26054 (SW03451)	SL26070 (SW03509)	SL26086 (SW03567)	SL26102 (SW03625)	SL26118 (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	SL26008 (SW03280)	SL26024 (SW03338)	SL26040 (SW03396)	SL26056 (SW03454)	SL26072 (SW03512)	SL26088 (SW03570)	SL26104 (SW03628)	SL26120 (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	SL26010 (SW03283)	SL26026 (SW03341)	SL26042 (SW03399)	SL26058 (SW03457)	SL26074 (SW03515)	SL26090 (SW03573)	SL26106 (SW03631)	SL26122 (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	SL26012 (SW03286)	SL26028 (SW03344)	SL26044 (SW03402)	SL26060 (SW03460)	SL26076 (SW03518)	SL26092 (SW03576)	SL26108 (SW03634)	SL26124 (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	SL260014 (SW03289)	SL26030 (SW03347)	SL26046 (SW03405)	SL26062 (SW03463)	SL26078 (SW03521)	SL26094 (SW03579)	SL26110 (SW03637)	SL26126 (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	

Continued on next page.

Continued from previous page.

System Work Number	Work 1	Work 2	Work 3	Work 4	Work 5	Work 6	Work 7	Work 8
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
Logical Axis 17 Program Current Position	SL08192	SL08224	SL08256	SL08288	SL08320	SL08352	SL08384	SL08416
Logical Axis 18 Program Current Position	SL08194	SL08226	SL08258	SL08290	SL08322	SL08354	SL08386	SL08418
Logical Axis 19 Program Current Position	SL08196	SL08228	SL08260	SL08292	SL08324	SL08356	SL08388	SL08420
Logical Axis 20 Program Current Position	SL08198	SL08230	SL08262	SL08294	SL08326	SL08358	SL08390	SL08422
Logical Axis 21 Program Current Position	SL08200	SL08232	SL08264	SL08296	SL08328	SL08360	SL08392	SL08424
Logical Axis 22 Program Current Position	SL08202	SL08234	SL08266	SL08298	SL08330	SL08362	SL08394	SL08426
Logical Axis 23 Program Current Position	SL08204	SL08236	SL08268	SL08300	SL08332	SL08364	SL08396	SL08428
Logical Axis 24 Program Current Position	SL08206	SL08238	SL08270	SL08302	SL08334	SL08366	SL08398	SL08430
Logical Axis 25 Program Current Position	SL08208	SL08240	SL08272	SL08304	SL08336	SL08368	SL08400	SL08432
Logical Axis 26 Program Current Position	SL08210	SL08242	SL08274	SL08306	SL08338	SL08370	SL08402	SL08434
Logical Axis 27 Program Current Position	SL08212	SL08244	SL08276	SL08308	SL08340	SL08372	SL08404	SL08436
Logical Axis 28 Program Current Position	SL08214	SL08246	SL08278	SL08310	SL08342	SL08374	SL08406	SL08438
Logical Axis 29 Program Current Position	SL08216	SL08248	SL08280	SL08312	SL08344	SL08376	SL08408	SL08440
Logical Axis 30 Program Current Position	SL08218	SL08250	SL08282	SL08314	SL08346	SL08378	SL08410	SL08442
Logical Axis 31 Program Current Position	SL08220	SL08252	SL08284	SL08316	SL08348	SL08380	SL08412	SL08444
Logical Axis 32 Program Current Position	SL08222	SL08254	SL08286	SL08318	SL08350	SL08382	SL08414	SL08446

- System Work Numbers 9 to 16

System Work Number	Work 9	Work 10	Work 11	Work 12	Work 13	Work 14	Work 15	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	SL26128 (SW03732)	SL26144 (SW03790)	SL26160 (SW03848)	SL26176 (SW03906)	SL26192 (SW03964)	SL26208 (SW04022)	SL26224 (SW04080)	SL26240 (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	SL26130 (SW03735)	SL26146 (SW03793)	SL26162 (SW03851)	SL26178 (SW03909)	SL26194 (SW03967)	SL26210 (SW04025)	SL26226 (SW04083)	SL26242 (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	SL26132 (SW03738)	SL26148 (SW03796)	SL26164 (SW03854)	SL26180 (SW03912)	SL26196 (SW03970)	SL26212 (SW04028)	SL26228 (SW04086)	SL26244 (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	SL26134 (SW03741)	SL26150 (SW03799)	SL26166 (SW03857)	SL26182 (SW03915)	SL26198 (SW03973)	SL26214 (SW04031)	SL26230 (SW04089)	SL26246 (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	SL26136 (SW03744)	SL26152 (SW03802)	SL26168 (SW03860)	SL26184 (SW03918)	SL26200 (SW03976)	SL26216 (SW04034)	SL26232 (SW04092)	SL26248 (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	SL26138 (SW03747)	SL26154 (SW03805)	SL26170 (SW03863)	SL26186 (SW03921)	SL26202 (SW03979)	SL26218 (SW04037)	SL26234 (SW04095)	SL26250 (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	SL26140 (SW03750)	SL26156 (SW03808)	SL26172 (SW03866)	SL26188 (SW03924)	SL26204 (SW03982)	SL26220 (SW04040)	SL26236 (SW04098)	SL26252 (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	SL26142 (SW03753)	SL26158 (SW03811)	SL26174 (SW03869)	SL26190 (SW03927)	SL26206 (SW03985)	SL26222 (SW04043)	SL26238 (SW04101)	SL26254 (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	

Continued on next page.

Continued from previous page.

System Work Number	Work 9	Work 10	Work 11	Work 12	Work 13	Work 14	Work 15	Work 16
Logical Axis 16 Program Current Position	SL03784	SL03842	SL03900	SL03958	SL04016	SL04074	SL04132	SL04190
Logical Axis 17 Program Current Position	SL08448	SL08480	SL08512	SL08544	SL08576	SL08608	SL08640	SL08672
Logical Axis 18 Program Current Position	SL08450	SL08482	SL08514	SL08546	SL08578	SL08610	SL08642	SL08674
Logical Axis 19 Program Current Position	SL08452	SL08484	SL08516	SL08548	SL08580	SL08612	SL08644	SL08676
Logical Axis 20 Program Current Position	SL08454	SL08486	SL08518	SL08550	SL08582	SL08614	SL08646	SL08678
Logical Axis 21 Program Current Position	SL08456	SL08488	SL08520	SL08552	SL08584	SL08616	SL08648	SL08680
Logical Axis 22 Program Current Position	SL08458	SL08490	SL08522	SL08554	SL08586	SL08618	SL08650	SL08682
Logical Axis 23 Program Current Position	SL08460	SL08492	SL08524	SL08556	SL08588	SL08620	SL08652	SL08684
Logical Axis 24 Program Current Position	SL08462	SL08494	SL08526	SL08558	SL08590	SL08622	SL08654	SL08686
Logical Axis 25 Program Current Position	SL08464	SL08496	SL08528	SL08560	SL08592	SL08624	SL08656	SL08688
Logical Axis 26 Program Current Position	SL08466	SL08498	SL08530	SL08562	SL08594	SL08626	SL08658	SL08690
Logical Axis 27 Program Current Position	SL08468	SL08500	SL08532	SL08564	SL08596	SL08628	SL08660	SL08692
Logical Axis 28 Program Current Position	SL08470	SL08502	SL08534	SL08566	SL08598	SL08630	SL08662	SL08694
Logical Axis 29 Program Current Position	SL08472	SL08504	SL08536	SL08568	SL08600	SL08632	SL08664	SL08696
Logical Axis 30 Program Current Position	SL08474	SL08506	SL08538	SL08570	SL08602	SL08634	SL08666	SL08698
Logical Axis 31 Program Current Position	SL08476	SL08508	SL08540	SL08572	SL08604	SL08636	SL08668	SL08700
Logical Axis 32 Program Current Position	SL08478	SL08510	SL08542	SL08574	SL08606	SL08638	SL08670	SL08702

• System Work Numbers 17 to 24

System Work Number	Work 17	Work 18	Work 19	Work 20	Work 21	Work 22	Work 23	Work 24	
Executing Main Program No.	SW03216	SW03217	SW03218	SW03219	SW03220	SW03221	SW03222	SW03223	
Status	SW04192	SW04250	SW04308	SW04366	SW04424	SW04482	SW04540	SW04598	
Control Signals	SW04193	SW04251	SW04309	SW04367	SW04425	SW04483	SW04541	SW04599	
Fork 0	Program Number	SW04194	SW04252	SW04310	SW04368	SW04426	SW04484	SW04542	SW04600
	Block Number	SW04195	SW04253	SW04311	SW04369	SW04427	SW04485	SW04543	SW04601
	Alarm Code	SL26256 (SW04196)	SL26272 (SW04254)	SL26288 (SW04312)	SL26304 (SW04370)	SL26320 (SW04428)	SL26336 (SW04486)	SL26352 (SW04544)	SL26368 (SW04602)
Fork 1	Program Number	SW04197	SW04255	SW04313	SW04371	SW04429	SW04487	SW04545	SW04603
	Block Number	SW04198	SW04256	SW04314	SW04372	SW04430	SW04488	SW04546	SW04604
	Alarm Code	SL26258 (SW04199)	SL26274 (SW04257)	SL26290 (SW04315)	SL26306 (SW04373)	SL26322 (SW04431)	SL26338 (SW04489)	SL26354 (SW04547)	SL26370 (SW04605)
Fork 2	Program Number	SW04200	SW04258	SW04316	SW04374	SW04432	SW04490	SW04548	SW04606
	Block Number	SW04201	SW04259	SW04317	SW04375	SW04433	SW04491	SW04549	SW04607
	Alarm Code	SL26260 (SW04202)	SL26276 (SW04260)	SL26292 (SW04318)	SL26308 (SW04376)	SL26324 (SW04434)	SL26340 (SW04492)	SL26356 (SW04550)	SL26372 (SW04608)
Fork 3	Program Number	SW04203	SW04261	SW04319	SW04377	SW04435	SW04493	SW04551	SW04609
	Block Number	SW04204	SW04262	SW04320	SW04378	SW04436	SW04494	SW04552	SW04610
	Alarm Code	SL26262 (SW04205)	SL26278 (SW04263)	SL26294 (SW04321)	SL26310 (SW04379)	SL26326 (SW04437)	SL26342 (SW04495)	SL26358 (SW04553)	SL26374 (SW04611)
Fork 4	Program Number	SW04206	SW04264	SW04322	SW04380	SW04438	SW04496	SW04554	SW04612
	Block Number	SW04207	SW04265	SW04323	SW04381	SW04439	SW04497	SW04555	SW04613
	Alarm Code	SL26264 (SW04208)	SL26280 (SW04266)	SL26296 (SW04324)	SL26312 (SW04382)	SL26328 (SW04440)	SL26344 (SW04498)	SL26360 (SW04556)	SL26376 (SW04614)
Fork 5	Program Number	SW04209	SW04267	SW04325	SW04383	SW04441	SW04499	SW04557	SW04615
	Block Number	SW04210	SW04268	SW04326	SW04384	SW04442	SW04500	SW04558	SW04616
	Alarm Code	SL26266 (SW04211)	SL26282 (SW04269)	SL26298 (SW04327)	SL26314 (SW04385)	SL26330 (SW04443)	SL26346 (SW04501)	SL26362 (SW04559)	SL26378 (SW04617)
Fork 6	Program Number	SW04212	SW04270	SW04328	SW04386	SW04444	SW04502	SW04560	SW04618
	Block Number	SW04213	SW04271	SW04329	SW04387	SW04445	SW04503	SW04561	SW04619
	Alarm Code	SL26268 (SW04214)	SL26284 (SW04272)	SL26300 (SW04330)	SL26316 (SW04388)	SL26332 (SW04446)	SL26348 (SW04504)	SL26364 (SW04562)	SL26380 (SW04620)
Fork 7	Program Number	SW04215	SW04273	SW04331	SW04389	SW04447	SW04505	SW04563	SW04621
	Block Number	SW04216	SW04274	SW04332	SW04390	SW04448	SW04506	SW04564	SW04622
	Alarm Code	SL26270 (SW04217)	SL26286 (SW04275)	SL26302 (SW04333)	SL26318 (SW04391)	SL26334 (SW04449)	SL26350 (SW04507)	SL26366 (SW04565)	SL26382 (SW04623)
Logical Axis 1 Program Current Position	SL04218	SL04276	SL04334	SL04392	SL04450	SL04508	SL04566	SL04624	
Logical Axis 2 Program Current Position	SL04220	SL04278	SL04336	SL04394	SL04452	SL04510	SL04568	SL04626	
Logical Axis 3 Program Current Position	SL04222	SL04280	SL04338	SL04396	SL04454	SL04512	SL04570	SL04628	
Logical Axis 4 Program Current Position	SL04224	SL04282	SL04340	SL04398	SL04456	SL04514	SL04572	SL04630	
Logical Axis 5 Program Current Position	SL04226	SL04284	SL04342	SL04400	SL04458	SL04516	SL04574	SL04632	
Logical Axis 6 Program Current Position	SL04228	SL04286	SL04344	SL04402	SL04460	SL04518	SL04576	SL04634	
Logical Axis 7 Program Current Position	SL04230	SL04288	SL04346	SL04404	SL04462	SL04520	SL04578	SL04636	
Logical Axis 8 Program Current Position	SL04232	SL04290	SL04348	SL04406	SL04464	SL04522	SL04580	SL04638	
Logical Axis 9 Program Current Position	SL04234	SL04292	SL04350	SL04408	SL04466	SL04524	SL04582	SL04640	
Logical Axis 10 Program Current Position	SL04236	SL04294	SL04352	SL04410	SL04468	SL04526	SL04584	SL04642	
Logical Axis 11 Program Current Position	SL04238	SL04296	SL04354	SL04412	SL04470	SL04528	SL04586	SL04644	
Logical Axis 12 Program Current Position	SL04240	SL04298	SL04356	SL04414	SL04472	SL04530	SL04588	SL04646	
Logical Axis 13 Program Current Position	SL04242	SL04300	SL04358	SL04416	SL04474	SL04532	SL04590	SL04648	
Logical Axis 14 Program Current Position	SL04244	SL04302	SL04360	SL04418	SL04476	SL04534	SL04592	SL04650	
Logical Axis 15 Program Current Position	SL04246	SL04304	SL04362	SL04420	SL04478	SL04536	SL04594	SL04652	

Continued on next page.

1.8 Advanced Programming

Monitoring Motion Program Execution Information

Continued from previous page.

System Work Number	Work 17	Work 18	Work 19	Work 20	Work 21	Work 22	Work 23	Work 24
Logical Axis 16 Program Current Position	SL04248	SL04306	SL04364	SL04422	SL04480	SL04538	SL04596	SL04654
Logical Axis 17 Program Current Position	SL08704	SL08736	SL08768	SL08800	SL08832	SL08864	SL08896	SL08928
Logical Axis 18 Program Current Position	SL08706	SL08738	SL08770	SL08802	SL08834	SL08866	SL08898	SL08930
Logical Axis 19 Program Current Position	SL08708	SL08740	SL08772	SL08804	SL08836	SL08868	SL08900	SL08932
Logical Axis 20 Program Current Position	SL08710	SL08742	SL08774	SL08806	SL08838	SL08870	SL08902	SL08934
Logical Axis 21 Program Current Position	SL08712	SL08744	SL08776	SL08808	SL08840	SL08872	SL08904	SL08936
Logical Axis 22 Program Current Position	SL08714	SL08746	SL08778	SL08810	SL08842	SL08874	SL08906	SL08938
Logical Axis 23 Program Current Position	SL08716	SL08748	SL08780	SL08812	SL08844	SL08876	SL08908	SL08940
Logical Axis 24 Program Current Position	SL08718	SL08750	SL08782	SL08814	SL08846	SL08878	SL08910	SL08942
Logical Axis 25 Program Current Position	SL08720	SL08752	SL08784	SL08816	SL08848	SL08880	SL08912	SL08944
Logical Axis 26 Program Current Position	SL08722	SL08754	SL08786	SL08818	SL08850	SL08882	SL08914	SL08946
Logical Axis 27 Program Current Position	SL08724	SL08756	SL08788	SL08820	SL08852	SL08884	SL08916	SL08948
Logical Axis 28 Program Current Position	SL08726	SL08758	SL08790	SL08822	SL08854	SL08886	SL08918	SL08950
Logical Axis 29 Program Current Position	SL08728	SL08760	SL08792	SL08824	SL08856	SL08888	SL08920	SL08952
Logical Axis 30 Program Current Position	SL08730	SL08762	SL08794	SL08826	SL08858	SL08890	SL08922	SL08954
Logical Axis 31 Program Current Position	SL08732	SL08764	SL08796	SL08828	SL08860	SL08892	SL08924	SL08956
Logical Axis 32 Program Current Position	SL08734	SL08766	SL08798	SL08830	SL08862	SL08894	SL08926	SL08958

- System Work Numbers 25 to 32

System Work Number	Work 25	Work 26	Work 27	Work 28	Work 29	Work 30	Work 31	Work 32	
Executing Main Program No.	SW03224	SW03225	SW03226	SW03227	SW03228	SW03229	SW03230	SW03231	
Status	SW04656	SW04714	SW04772	SW04830	SW04888	SW04946	SW05004	SW05062	
Control Signals	SW04657	SW04715	SW04773	SW04831	SW04889	SW04947	SW05005	SW05063	
Fork 0	Program Number	SW04658	SW04716	SW04774	SW04832	SW04890	SW04948	SW05006	SW05064
	Block Number	SW04659	SW04717	SW04775	SW04833	SW04891	SW04949	SW05007	SW05065
	Alarm Code	SL26384 (SW04660)	SL26400 (SW04718)	SL26416 (SW04776)	SL26432 (SW04834)	SL26448 (SW04892)	SL26464 (SW04950)	SL26480 (SW05008)	SL26496 (SW05066)
Fork 1	Program Number	SW04661	SW04719	SW04777	SW04835	SW04893	SW04951	SW05009	SW05067
	Block Number	SW04662	SW04720	SW04778	SW04836	SW04894	SW04952	SW05010	SW05068
	Alarm Code	SL26386 (SW04663)	SL26402 (SW04721)	SL26418 (SW04779)	SL26434 (SW04837)	SL26450 (SW04895)	SL26466 (SW04953)	SL26482 (SW05011)	SL26498 (SW05069)
Fork 2	Program Number	SW04664	SW04722	SW04780	SW04838	SW04896	SW04954	SW05012	SW05070
	Block Number	SW04665	SW04723	SW04781	SW04839	SW04897	SW04955	SW05013	SW05071
	Alarm Code	SL26388 (SW04666)	SL26404 (SW04724)	SL26420 (SW04782)	SL26436 (SW04840)	SL26452 (SW04898)	SL26468 (SW04956)	SL26484 (SW05014)	SL26500 (SW05072)
Fork 3	Program Number	SW04667	SW04725	SW04783	SW04841	SW04899	SW04957	SW05015	SW05073
	Block Number	SW04668	SW04726	SW04784	SW04842	SW04900	SW04958	SW05016	SW05074
	Alarm Code	SL26390 (SW04669)	SL26406 (SW04727)	SL26422 (SW04785)	SL26438 (SW04843)	SL26454 (SW04901)	SL26470 (SW04959)	SL26486 (SW05017)	SL26502 (SW05075)
Fork 4	Program Number	SW04670	SW04728	SW04786	SW04844	SW04902	SW04960	SW05018	SW05076
	Block Number	SW04671	SW04729	SW04787	SW04845	SW04903	SW04961	SW05019	SW05077
	Alarm Code	SL26392 (SW04672)	SL26408 (SW04730)	SL26424 (SW04788)	SL26440 (SW04846)	SL26456 (SW04904)	SL26472 (SW04962)	SL26488 (SW05020)	SL26504 (SW05078)
Fork 5	Program Number	SW04673	SW04731	SW04789	SW04847	SW04905	SW04963	SW05021	SW05079
	Block Number	SW04674	SW04732	SW04790	SW04848	SW04906	SW04964	SW05022	SW05080
	Alarm Code	SL26394 (SW04675)	SL26410 (SW04733)	SL26426 (SW04791)	SL26442 (SW04849)	SL26458 (SW04907)	SL26474 (SW04965)	SL26490 (SW05023)	SL26506 (SW05081)
Fork 6	Program Number	SW04676	SW04734	SW04792	SW04850	SW04908	SW04966	SW05024	SW05082
	Block Number	SW04677	SW04735	SW04793	SW04851	SW04909	SW04967	SW05025	SW05083
	Alarm Code	SL26396 (SW04678)	SL26412 (SW04736)	SL26428 (SW04794)	SL26444 (SW04852)	SL26460 (SW04910)	SL26476 (SW04968)	SL26492 (SW05026)	SL26508 (SW05084)
Fork 7	Program Number	SW04679	SW04737	SW04795	SW04853	SW04911	SW04969	SW05027	SW05085
	Block Number	SW04680	SW04738	SW04796	SW04854	SW04912	SW04970	SW05028	SW05086
	Alarm Code	SL26398 (SW04681)	SL26414 (SW04739)	SL26430 (SW04797)	SL26446 (SW04855)	SL26462 (SW04913)	SL26478 (SW04971)	SL26494 (SW05029)	SL26510 (SW05087)
Logical Axis 1 Program Current Position	SL04682	SL04740	SL04798	SL04856	SL04914	SL04972	SL05030	SL05088	
Logical Axis 2 Program Current Position	SL04684	SL04742	SL04800	SL04858	SL04916	SL04974	SL05032	SL05090	
Logical Axis 3 Program Current Position	SL04686	SL04744	SL04802	SL04860	SL04918	SL04976	SL05034	SL05092	
Logical Axis 4 Program Current Position	SL04688	SL04746	SL04804	SL04862	SL04920	SL04978	SL05036	SL05094	
Logical Axis 5 Program Current Position	SL04690	SL04748	SL04806	SL04864	SL04922	SL04980	SL05038	SL05096	
Logical Axis 6 Program Current Position	SL04692	SL04750	SL04808	SL04866	SL04924	SL04982	SL05040	SL05098	
Logical Axis 7 Program Current Position	SL04694	SL04752	SL04810	SL04868	SL04926	SL04984	SL05042	SL05100	
Logical Axis 8 Program Current Position	SL04696	SL04754	SL04812	SL04870	SL04928	SL04986	SL05044	SL05102	
Logical Axis 9 Program Current Position	SL04698	SL04756	SL04814	SL04872	SL04930	SL04988	SL05046	SL05104	
Logical Axis 10 Program Current Position	SL04700	SL04758	SL04816	SL04874	SL04932	SL04990	SL05048	SL05106	
Logical Axis 11 Program Current Position	SL04702	SL04760	SL04818	SL04876	SL04934	SL04992	SL05050	SL05108	
Logical Axis 12 Program Current Position	SL04704	SL04762	SL04820	SL04878	SL04936	SL04994	SL05052	SL05110	
Logical Axis 13 Program Current Position	SL04706	SL04764	SL04822	SL04880	SL04938	SL04996	SL05054	SL05112	
Logical Axis 14 Program Current Position	SL04708	SL04766	SL04824	SL04882	SL04940	SL04998	SL05056	SL05114	
Logical Axis 15 Program Current Position	SL04710	SL04768	SL04826	SL04884	SL04942	SL05000	SL05058	SL05116	

Continued on next page.

1.8 Advanced Programming

Monitoring Motion Program Execution Information

Continued from previous page.

System Work Number	Work 25	Work 26	Work 27	Work 28	Work 29	Work 30	Work 31	Work 32
Logical Axis 16 Program Current Position	SL04712	SL04770	SL04828	SL04886	SL04944	SL05002	SL05060	SL05118
Logical Axis 17 Program Current Position	SL08960	SL08992	SL09024	SL09056	SL09088	SL09120	SL09152	SL09184
Logical Axis 18 Program Current Position	SL08962	SL08994	SL09026	SL09058	SL09090	SL09122	SL09154	SL09186
Logical Axis 19 Program Current Position	SL08964	SL08996	SL09028	SL09060	SL09092	SL09124	SL09156	SL09188
Logical Axis 20 Program Current Position	SL08966	SL08998	SL09030	SL09062	SL09094	SL09126	SL09158	SL09190
Logical Axis 21 Program Current Position	SL08968	SL09000	SL09032	SL09064	SL09096	SL09128	SL09160	SL09192
Logical Axis 22 Program Current Position	SL08970	SL09002	SL09034	SL09066	SL09098	SL09130	SL09162	SL09194
Logical Axis 23 Program Current Position	SL08972	SL09004	SL09036	SL09068	SL09100	SL09132	SL09164	SL09196
Logical Axis 24 Program Current Position	SL08974	SL09006	SL09038	SL09070	SL09102	SL09134	SL09166	SL09198
Logical Axis 25 Program Current Position	SL08976	SL09008	SL09040	SL09072	SL09104	SL09136	SL09168	SL09200
Logical Axis 26 Program Current Position	SL08978	SL09010	SL09042	SL09074	SL09106	SL09138	SL09170	SL09202
Logical Axis 27 Program Current Position	SL08980	SL09012	SL09044	SL09076	SL09108	SL09140	SL09172	SL09204
Logical Axis 28 Program Current Position	SL08982	SL09014	SL09046	SL09078	SL09110	SL09142	SL09174	SL09206
Logical Axis 29 Program Current Position	SL08984	SL09016	SL09048	SL09080	SL09112	SL09144	SL09176	SL09208
Logical Axis 30 Program Current Position	SL08986	SL09018	SL09050	SL09082	SL09114	SL09146	SL09178	SL09210
Logical Axis 31 Program Current Position	SL08988	SL09020	SL09052	SL09084	SL09116	SL09148	SL09180	SL09212
Logical Axis 32 Program Current Position	SL08990	SL09022	SL09054	SL09086	SL09118	SL09150	SL09182	SL09214

1.9

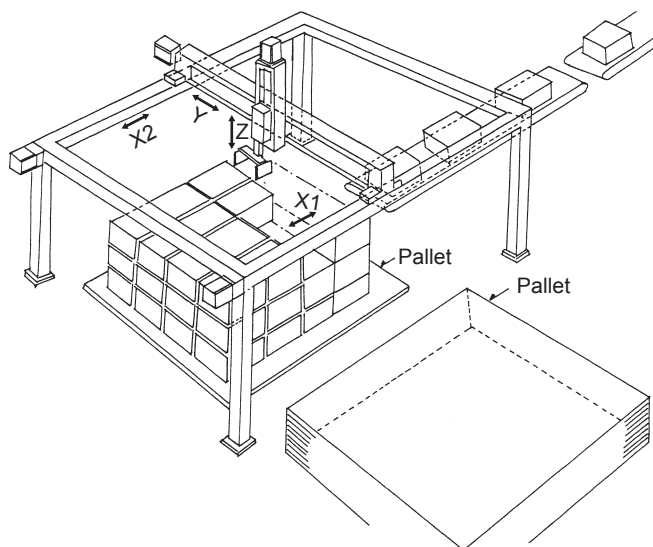
Application Examples

Motion programs can be used for a variety of different devices and systems. This section gives some application examples.

Conveyance Device

In this example, a device stacks a specified number of cardboard boxes on a pallet and transports them to the next process.

Three axes are controlled with motion control for the palletizing process and an automatic pallet feeding sequence is performed.



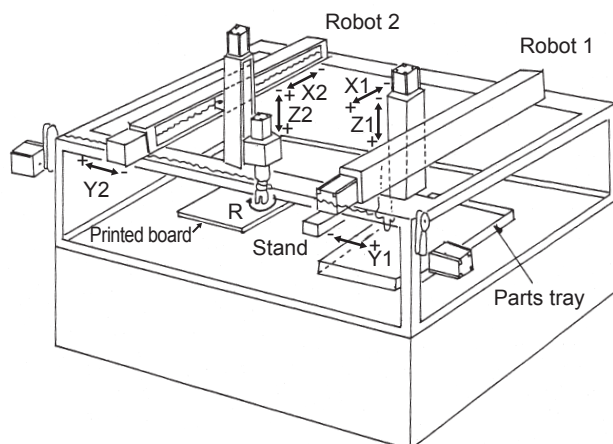
Control Points

- Axes X1 and X2 are moved in synchronized operation using a virtual axis.
- Interpolation is used to enable smooth movement.
- Palletizing is performed by calculating the position data with a motion program according to predefined conditions (box dimensions, the number of boxes in a horizontal row, the number of boxes in a vertical row, and the number of boxes in a stack).

Part Inserter

In this example, a device inserts parts, such as connectors, into a printed circuit board.

The transport robot takes out the parts and brings them to the stand. The inserting robot inserts the parts at the specified positions and angles on the circuit board.



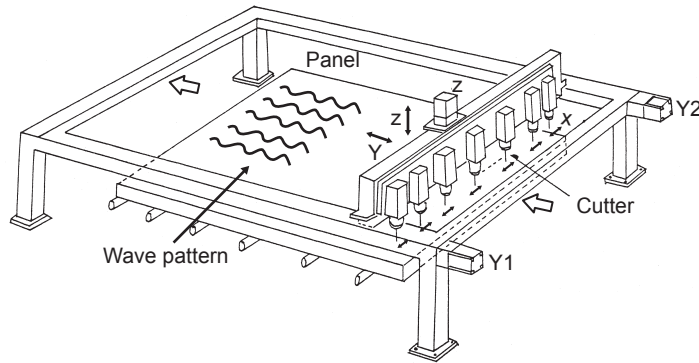
Control Points

- Two groups of axes are created and a program is created for each group so that each robot is independently controlled.
- The tact time is shortened by using two-axis or three-axis linear interpolation.

Panel Processing Machine

In this example, a device draws patterns on flat panels for construction materials.

More than ten cutters are mounted in series on the X axis so that the width of the pattern can be easily changed.



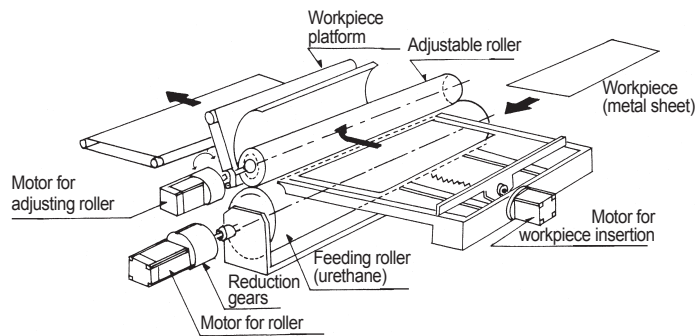
Control Points

- The X and Y axes are moved with circular interpolation to draw waveform patterns.
- Movement of the X1 and X2 axes is synchronized by using a virtual axis.

Metal Sheet Pressing Equipment

In this example, a device is used to bend metal sheets.

A metal sheet is bent into various shapes by changing the adjustable axis while feeding a sheet using the rolling axis.



Control Points

- Two axes, a linear axis and rotational axis, are controlled by using linear interpolation.
- The motion program to call is changed based on the processing that needs to be performed.

Introduction to Sequence Programs

2

This chapter introduces sequence programs, their features, and how to use them for first-time users of sequence programs.

2.1	What Is a Sequence Program?	2-2
2.2	Features of a Sequence Program	2-3
	Sequence Program Execution Methods	2-3
	Same Language as Motion Programs	2-3
	Data Transfer to and from Motion Programs	2-3
	Memory Usage Reduced by Use of Subprograms	2-4
	Easy Programming Functions	2-4
2.3	Types of Sequence Programs	2-5
2.4	Executing Sequence Programs	2-6
	Execution Processing Method	2-6
	Registering Program Execution	2-8
	Work Registers	2-9

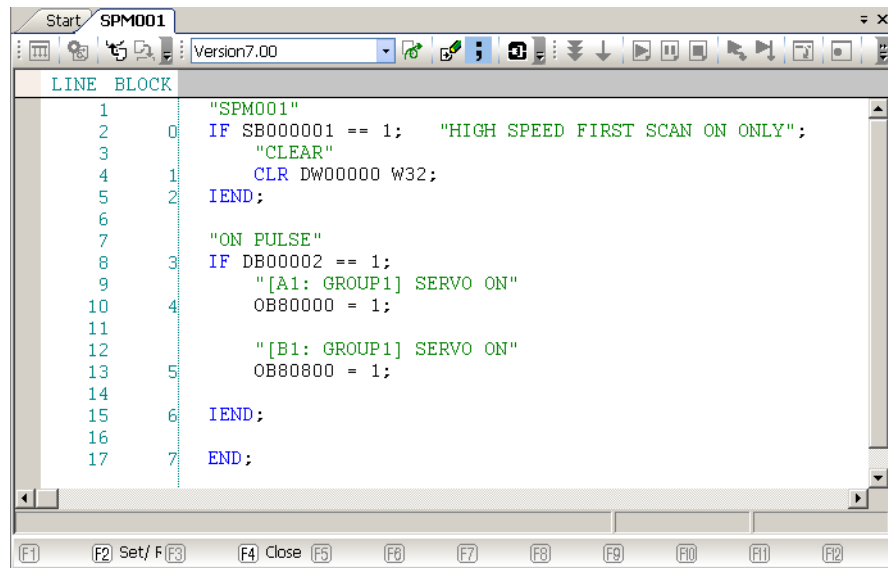
2.1 What Is a Sequence Program?

A sequence program is executed in a scan and it is written in the same language as a motion program. An application to cyclically check status, such as interlock status, can be created by using a sequence program.

Sequence programs can be executed by calling them from the M-EXECUTOR program execution definitions.

You can create up to 512 sequence programs. However, you must also include motion programs in this total.

An example of a sequence program is shown below.



The screenshot shows a software interface for editing a sequence program named SPM001. The window title is "Start SPM001" and the version is "Version7.00". The code is displayed in a table with columns for "LINE" and "BLOCK". The code includes comments and logic for checking status and controlling servos.

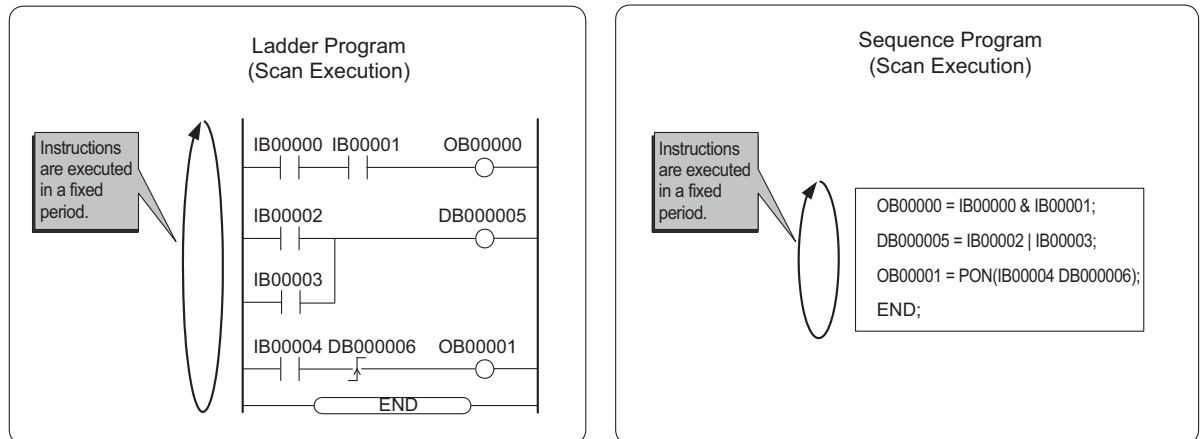
LINE	BLOCK	Code
1		"SPM001"
2	0	IF SB000001 == 1; "HIGH SPEED FIRST SCAN ON ONLY";
3		"CLEAR"
4	1	CLR DW00000 W32;
5	2	IEND;
6		
7		"ON PULSE"
8	3	IF DB00002 == 1;
9		"[A1: GROUP1] SERVO ON"
10	4	OB80000 = 1;
11		
12		"[B1: GROUP1] SERVO ON"
13	5	OB80800 = 1;
14		
15	6	IEND;
16		
17	7	END;

2.2 Features of a Sequence Program

Sequence Program Execution Methods

Sequence programs are executed in the same way as ladder programs.

A sequence program is executed cyclically in a fixed scan. Processing from the start of the program to the END instruction is completed in one scan. Sequence programs can be executed by calling them from the M-EXECUTOR program execution definitions.



Same Language as Motion Programs

Sequence programs use the motion language, just like motion programs.

The motion language instructions that can be used in sequence programs, however, are limited to sequence instructions, such as math instructions. Instructions for motion control, such as axis movement instructions, cannot be used.

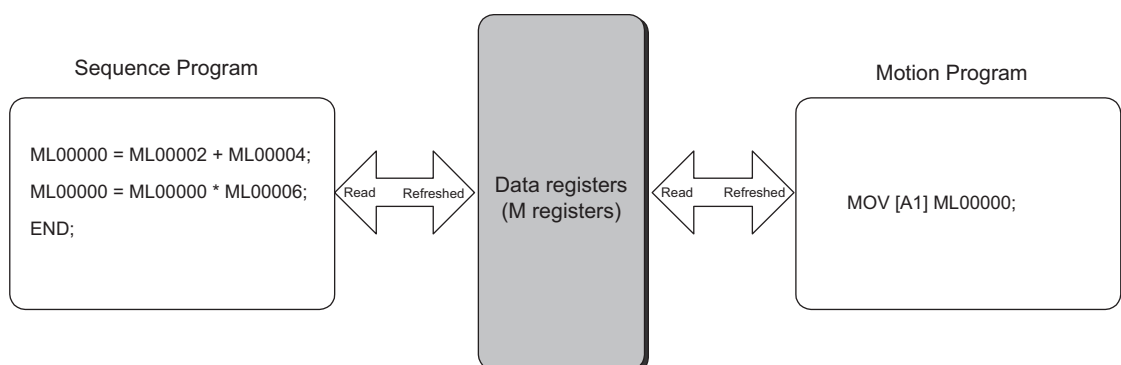
You can use sequence programs to create applications for sequence control without using ladder programs.

Data Transfer to and from Motion Programs

You can transfer data between sequence and motion programs.

Data registers (M registers) are used to transfer data.

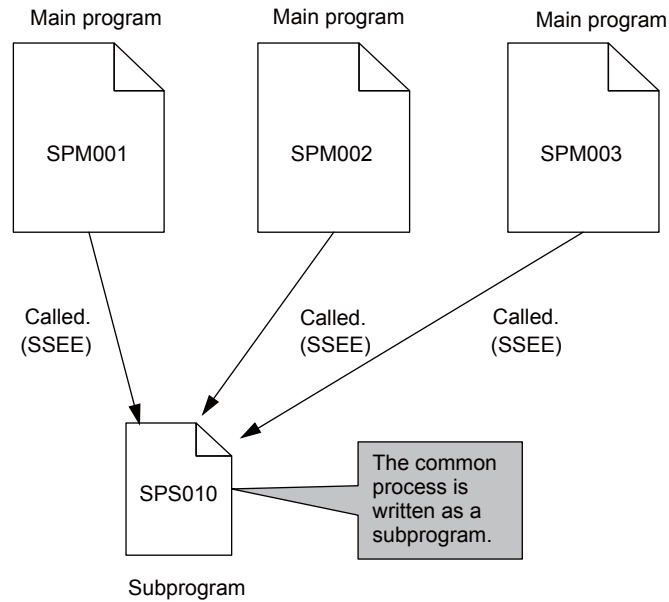
For example, this allows a value that is updated in a sequence program to be used in a motion program, and vice-versa.



Memory Usage Reduced by Use of Subprograms

You can create sequence programs as subprograms.

Subprograms are created to perform common operations. They help minimizing the number of program steps and allow the efficient use of memory.



Easy Programming Functions

The following easy programming functions can also be used for sequence programs.

- Instruction Entry Assistance

Simply select an instruction and set the data in the Instruction Input Assistance Dialog Box shown below to insert the instruction into the editor.



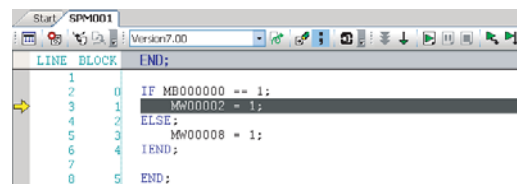
Inserts the instruction.

```

Version:
"BLOCK MOVE"
BLK M#00100 M#00500 W20;
END;
    
```

- Debugging

This mode allows you to debug sequence programs. Common debugging commands, such as step-by-step execution and setting breakpoints, are provided.



2.3

Types of Sequence Programs

There are two types of sequence programs.

Type	Designation Method	Features	Number of Programs
Main programs	SPM□□□ (□□□ = 1 to 512)	Main programs are called from the M-EXECUTOR program execution definitions.	You can create up to 512 motion programs, including the following programs: Motion main programs
Subprograms	SPS□□□ (□□□ = 1 to 512)	Subprograms are called from a main program.	Motion subprograms Sequence main programs Sequence subprograms



Important

The same numbers are used to manage the sequence programs and motion programs.

Use a unique number for each program.

- Motion program numbers are given in the form MPM□□□ or MPS□□□.
- Sequence program numbers are given in the form SPM□□□ or SPS□□□.

2.4 Executing Sequence Programs

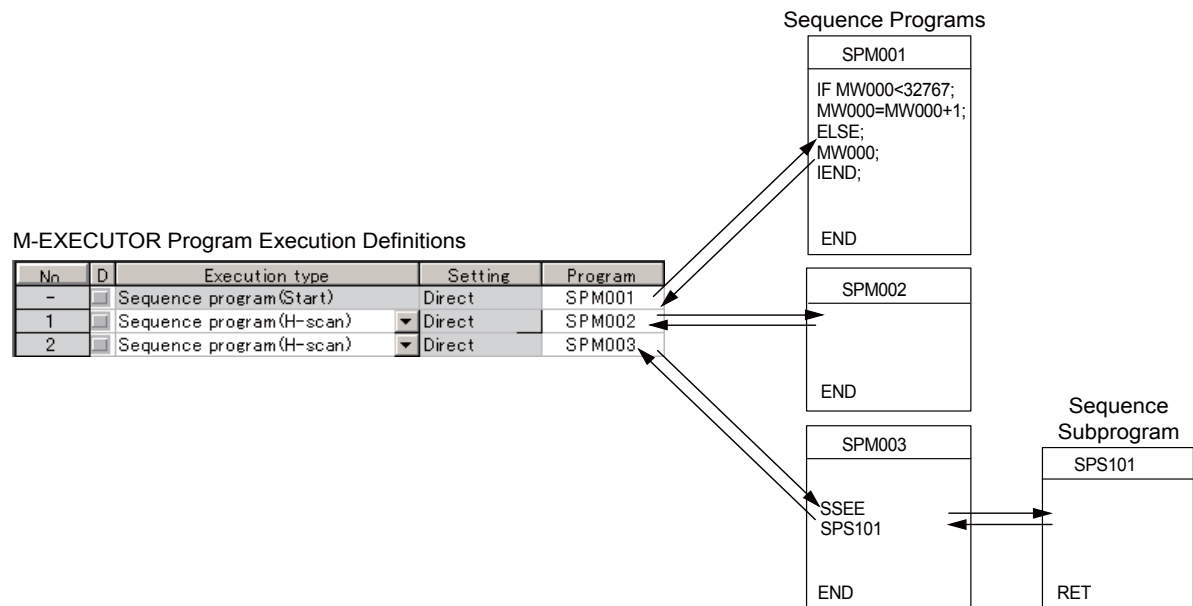
This section describes how to execute sequence programs.

Execution Processing Method

A sequence program is executed by calling it from the M-EXECUTOR execution definitions.

Sequence programs are executed in ascending order.

The following figure shows an execution example.



If the execution type is set to a high-speed scan sequence program or low-speed scan sequence program, then the program will be executed as soon as the definition is saved. If the execution type is set to a startup sequence program, then the program will be executed the next time when the power supply is turned ON.

M-EXECUTOR Program Execution Definitions

Example

Sequence Program Execution Example

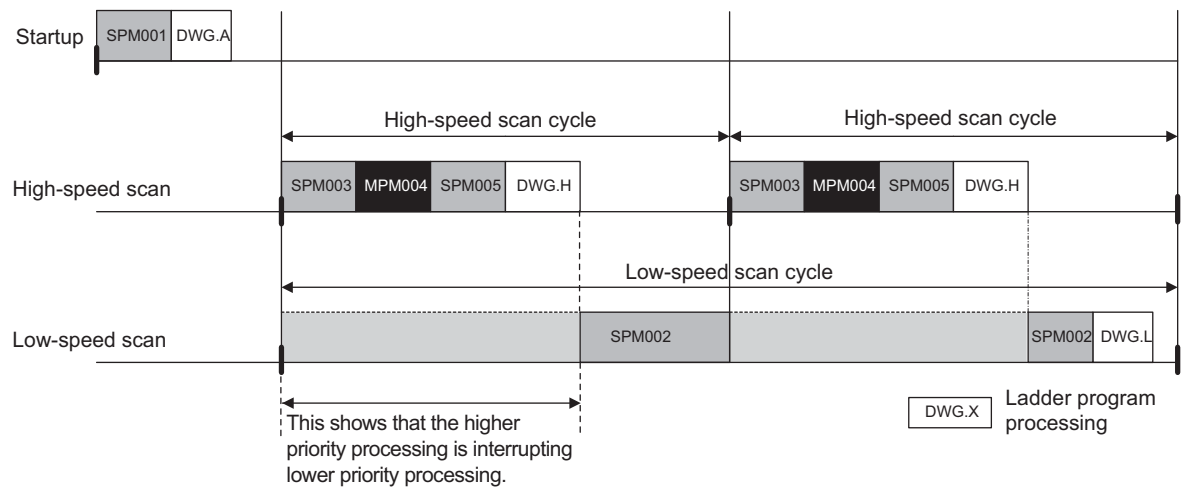
The following figure shows an example of the sequence programs registered in the M-EXECUTOR.

Nr	D	Execution type	Setting	Program	Execution monitor register(S register)
-		Sequence program(Start)	Direct	-	-
1		Motion program	Direct	MPM001	SW03264 - SW03321
2		Sequence program(L-scan)	Direct	SPM002	-
3		Sequence program(H-scan)	Direct	SPM003	-
4		Motion program	Direct	MPM004	SW03438 - SW03495
5		Sequence program(H-scan)	Direct	SPM005	-
6		-----			
7		-----			
8		-----			

Execution Timing

This section describes the execution timing of programs in the above example.

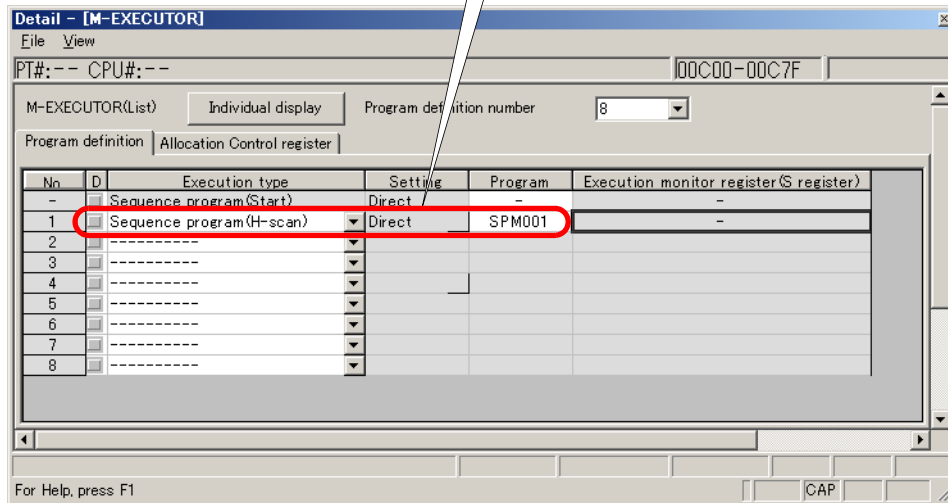
The following figure shows program and drawing execution that is based on the order of registration in the M-EXECUTOR program definitions.



Registering Program Execution

Register the programs to execute as shown below. The following screen capture shows an example of registering the SPM001 sequence program for execution in a high-speed scan cycle.

Register the program to execute.

**Information**

Sequence programs must be directly designated. Indirect designations cannot be used.

Work Registers

When a sequence program is registered for execution, that program is assigned status flags to monitor its status. The address of the status flags for a sequence program can be obtained with the following equation.

$$IW\boxed{\square}\boxed{\square}\boxed{\square}\boxed{\square} + 4 \times (\text{Program execution registration No. } 1)$$

└ First M-EXECUTOR I/O register address*

*You can check the first I/O register address on the Module Configuration Definition Tab Page.

Module	Function Module/Slave	Status	Circuit No./AxisAddress Start Duplied circ.	Motion Register	Disabled	Register (Input/Output) Start - End Size	Scan	Com
01 CPU-201	UNDEFINED							
PSA-12								
01 CPU	Driving							
02 218FD	Driving		Circuit No1 1			Input 0000 - 07FF[H] 2048 Output		
03 SVC32	Driving		Circuit No1 2	8000 - 8FFF[H]		Input 0800 - 0BFF[H] 1004 Output		
04 SVR32	Driving		Circuit No3 2	8000 - 9FFF[H]				
05 M-EXECUTOR	Driving					Input 0000 - 007F[H] 128		
06	UNDEFINED							
01	UNDEFINED							
02	UNDEFINED							
03	UNDEFINED							
04	UNDEFINED							
05	UNDEFINED							
02	UNDEFINED							
03	UNDEFINED							

Status Flags

The Sequence Program Status Flags give the execution conditions of the sequence program.

The following table describes the meanings of the Status Flags.

Bit No	Name	Description
0 to 3	Bit 0 Program Executing	This bit is set to 1 when the sequence program is running. 0: Sequence program is stopped. 1: Sequence program is running.
	Bit 1 (Reserved for system.)	—
	Bit 2 (Reserved for system.)	—
	Bit 3 (Reserved for system.)	—
4 to 7	Bit 4 (Reserved for system.)	—
	Bit 5 (Reserved for system.)	—
	Bit 6 (Reserved for system.)	—
	Bit 7 (Reserved for system.)	—

Continued on next page.

Continued from previous page.

Bit No	Name	Description
8 to B	Bit 8	Program Alarm This bit changes to 1 when any of the following errors occur after calling a sequence subprogram using an SSEE instruction. This bit changes back to 0 when the error is cleared. <ul style="list-style-type: none"> • The called program is not registered. • The called program is not a sequence program. • The called program is not a subprogram (a main program was called). • Called Program Number Limit Exceeded Error • Too Many Nesting Levels Error 0: There is no program alarm. 1: A program alarm occurred.
	Bit 9	Program Stopped at Breakpoint This bit is set to 1 when execution of a program stops at a breakpoint in Debug Operation Mode. 0: Not stopped at a breakpoint. 1: Stopped at a breakpoint.
	Bit A	(Reserved for system.) –
	Bit B	Debug Operation Mode This bit is set to 1 when a program is running in Debug Operation Mode. 0: Not in Debug Operation Mode (Normal Execution Mode). 1: In Debug Operation Mode.
C to F	Bit C	Program Type This bit reports whether the program that is being executed is a motion program or a sequence program. 0: Motion program 1: Sequence program
	Bit D	Start Request History This bit is set to 1 when the sequence program is running. 0: Sequence program is stopped. 1: Sequence program is running.
	Bit E	(Reserved for system.) –
	Bit F	(Reserved for system.) –



Note

Sequence Program Alarms

Bit 8 (Program Alarm) in the Status Flags changes to 1 if an error is detected after calling a sequence subprogram with an SSEE instruction. This bit changes back to 0 when the error is cleared.

The following errors can occur.

- The called program is not registered.
- The called program is not a sequence program.
- The called program is not a subprogram (a main program was called).
- Called Program Number Limit Exceeded Error
- Too Many Nesting Levels Error

Program Development Flow

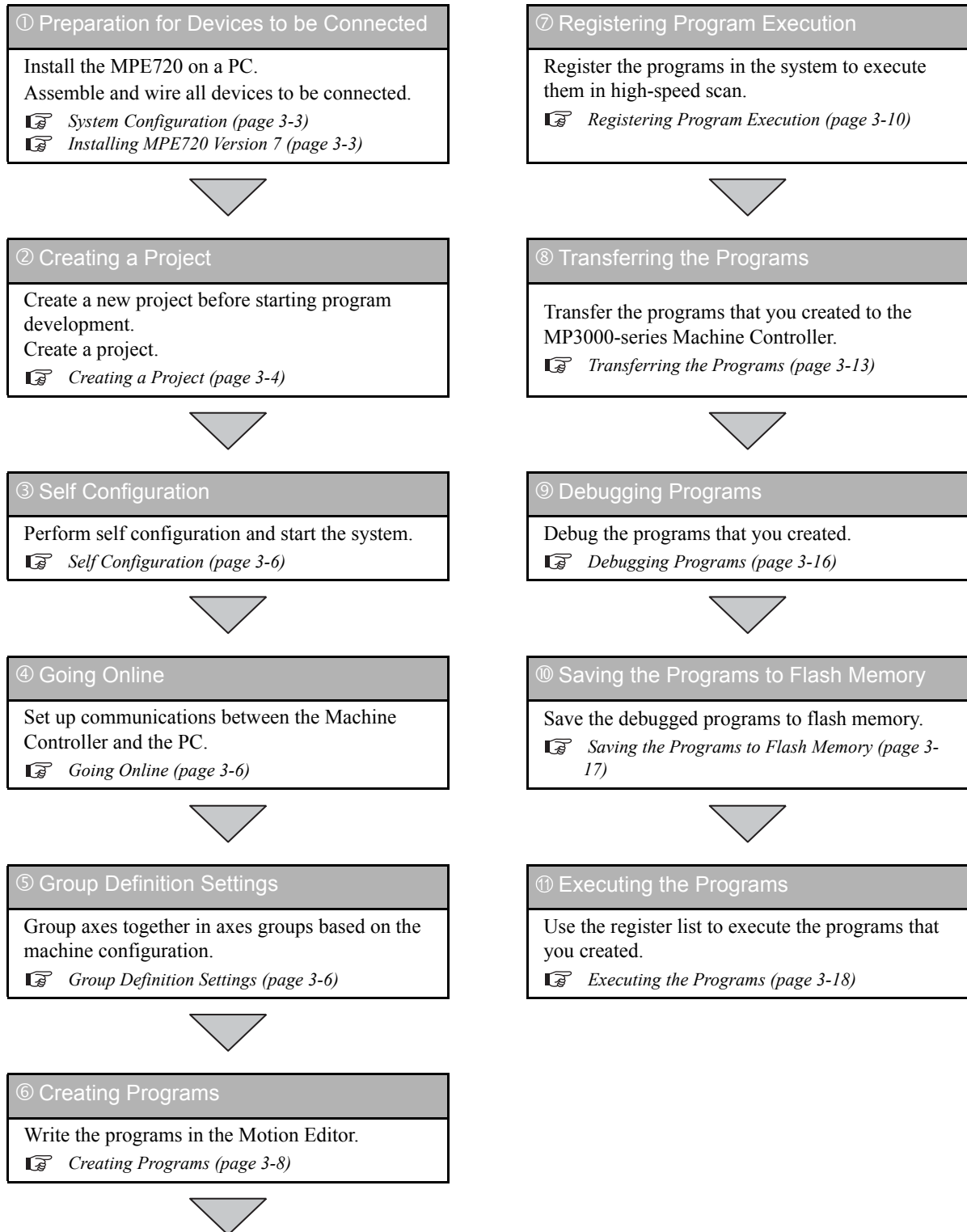
3

This chapter describes the procedures from system setup to actual operation using MPE720 Engineering Tool version 7.

3.1	Program Development Flow	3-2
3.2	Program Development Procedures	3-3
	Preparation for Devices to be Connected	3-3
	Creating a Project	3-4
	Self Configuration	3-6
	Going Online	3-6
	Group Definition Settings	3-6
	Creating Programs	3-8
	Registering Program Execution	3-10
	Transferring the Programs	3-13
	Debugging Programs	3-16
	Saving the Programs to Flash Memory	3-17
	Executing the Programs	3-18

3.1 Program Development Flow

In this chapter, motion program development procedures are described according to the following flowchart.



Note: 1. The development procedure for sequence programs is basically the same as that for motion programs.

This section describes the development flow for motion programs.

2. The above flowchart is an example of the program development process. External devices must be set up to use programs on the actual system.

3.2 Program Development Procedures

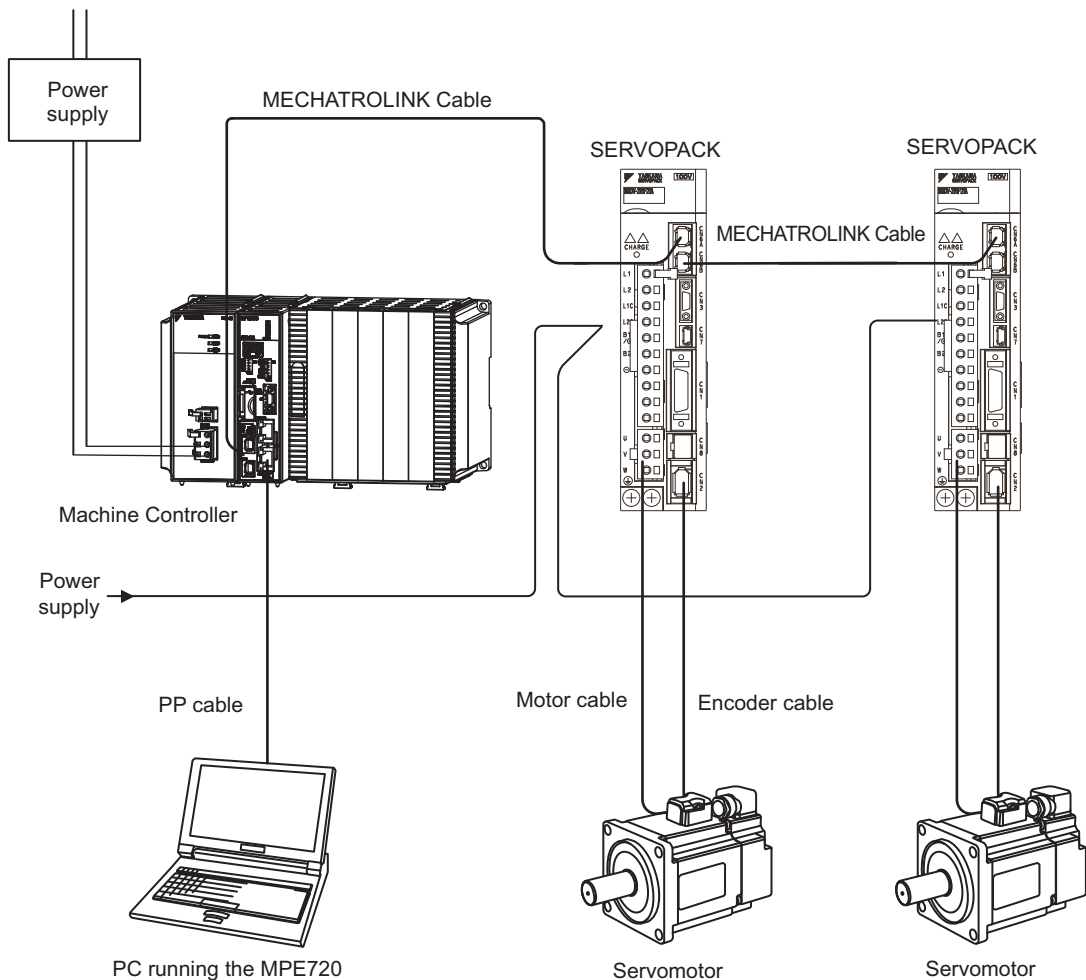
This section describes the procedures to develop programs based on an example system.

Preparation for Devices to be Connected

This section describes an example system configuration for the devices connected to the Machine Controller and the setup procedures that are required before starting the system.

System Configuration

The following figure shows a typical system configuration.



Note: In the system configuration example that is given above, the SERVOPACK station numbers are set to 1 and 2.

Installing MPE720 Version 7

Install MPE720 version 7 on a PC.

Refer to the following manual for the installation procedure.

MP2000/MP3000 Series Machine Controller System Setup Manual (Manual No.: SIEP C880725 00)

Creating a Project

A project file is the application file for MPE720 version 7. It includes the following information.

System Configuration	<ul style="list-style-type: none"> • System definitions • Scan time definitions • Module configuration definition • Data tracing information
Program	<ul style="list-style-type: none"> • Ladder programs (high-speed, low-speed, start, interrupt, and function programs) • Motion programs (main programs, subprograms, and group definitions) • Table data • Variables (axis, I/O, global, constant, and user-defined structure variables) • Comments (I/O, global, and constant comments)
Registers	<ul style="list-style-type: none"> • M (data registers) • D (internal registers) • C (constant registers) • S (system registers) • I (input registers) • O (output registers) • G (data registers)

The project file includes files for all of the above information but allows you to handle them as a single file in Windows. The project file extension is .YMW7.

Opening a project file enables editing all of these files.

Only one project file can be opened in a single window with MPE720 version 7. The same project file cannot be opened in more than one window with MPE720 version 7. If you try to open a project file that is already open, the window that contains the open project file will move to the front.



Important

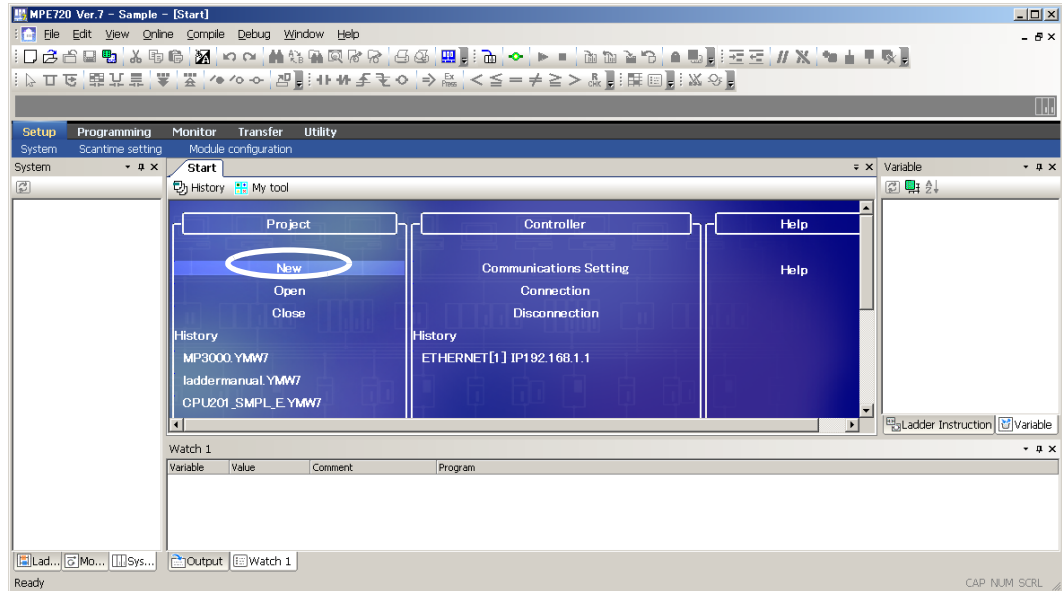
You can also use project files created in MPE720 version 6.0 (extension .YMV). In this case, the extended features of the MP3000-series Machine Controllers cannot be used.

Use the following procedure to create a new project.

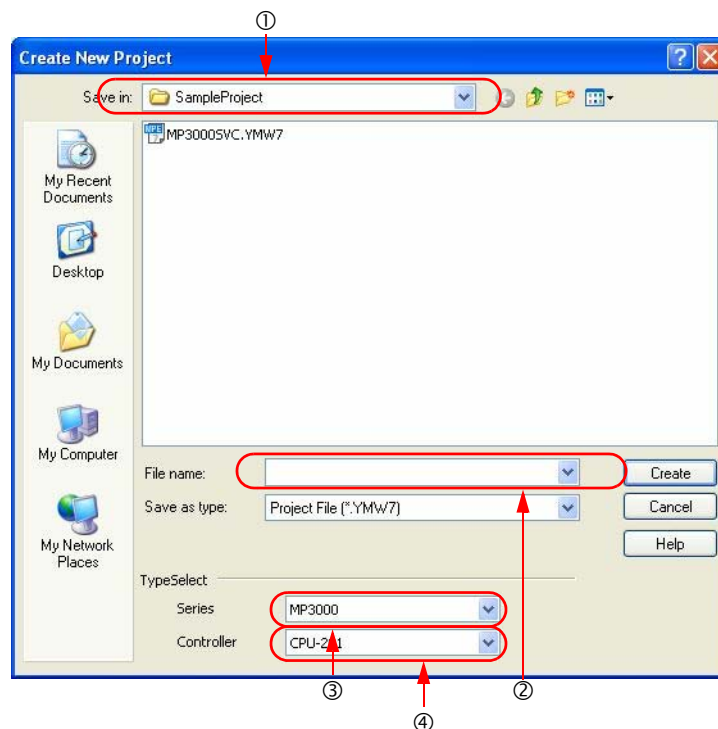
1. Double-click the icon shown below on the computer desktop to start MPE720 version 7.



2. Select **New** on the Start Tab Page.



3. Specify the file name, file storage location, Machine Controller series, and model.



- ① Specify the destination location in the **Save in** Box.
- ② Enter the file name in the **File name** Box.
- ③ Select the applicable series in the **Series** Box.
- ④ Select the applicable model in the **Controller** Box.

4. Click the **Create** Button.

Self Configuration

Set up the system by performing self configuration. Self configuration is used to automatically detect all the Modules that are installed in the MP3000-series Machine Controller and all the slave devices that are connected via the MECHATROLINK connector (such as Servo Drives), and then create the module configuration definition files based on that information. This allows you to quickly and easily set up the system. You can perform self configuration either when the power supply to the Machine Controller is turned ON or by using the MPE720.

Refer to the following manual for details on self configuration.

📖 *MP3000-series Basic Units User's Manual* (Manual No.: SIEP C880725 10)

📖 *MP3000 Series MP3300 Product Manual* (Manual No.: SIEP C880725 21)

Going Online

Set the conditions for communications between the PC on which MPE720 version 7 is installed and the Machine Controller.

Refer to the following manual for the procedure to set up communications.

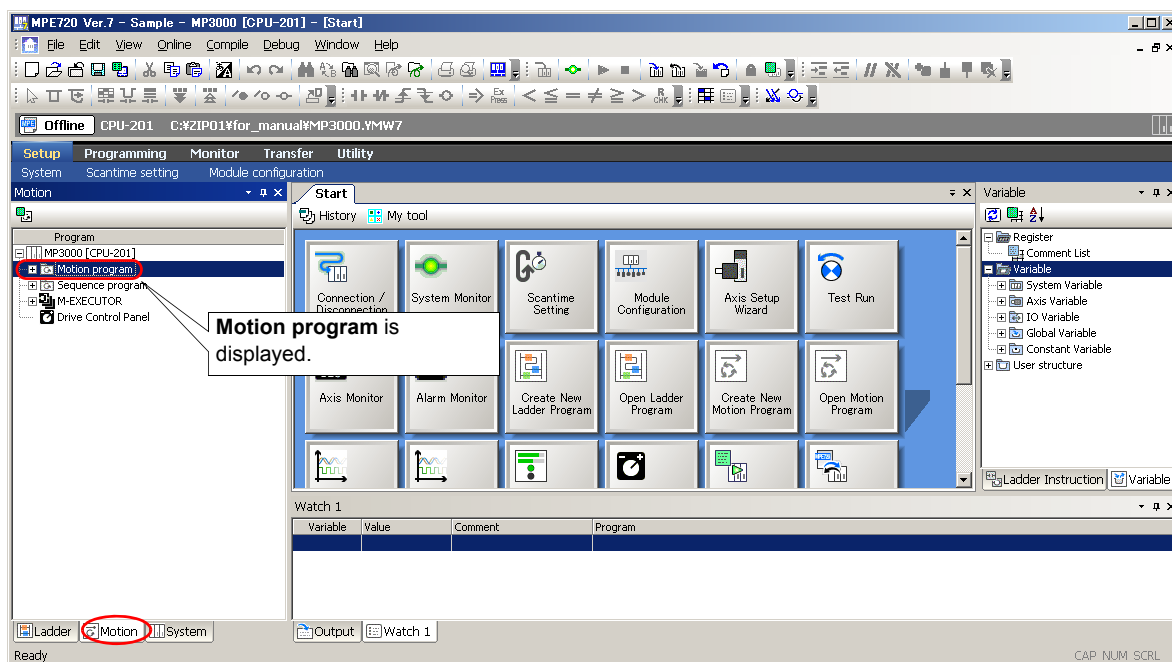
📖 *MP2000/MP3000 Series Machine Controller System Setup Manual* (Manual No.: SIEP C880725 00)

Group Definition Settings

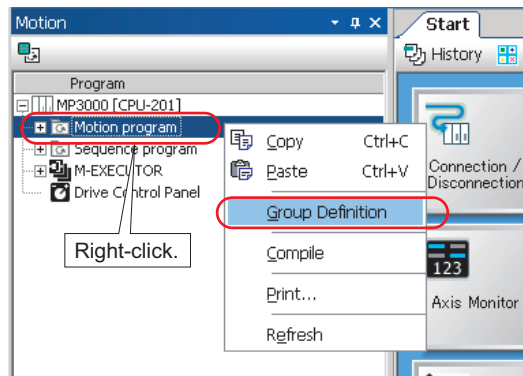
Before creating a motion program, group the axes together as required by the machine configuration.

1. Click the **Motion** Tab in the pane.

Motion Program is displayed in the tree hierarchy in the pane.



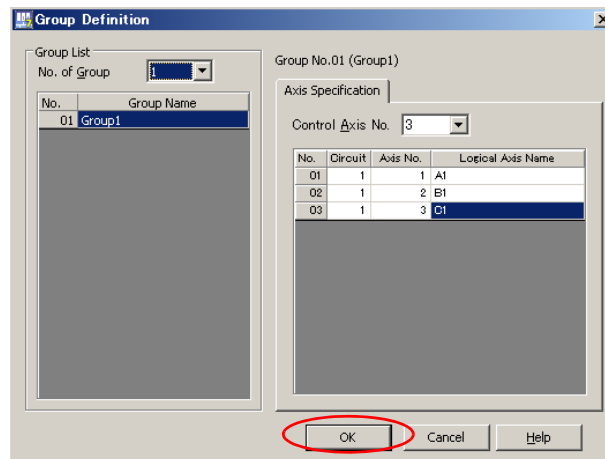
- Right-click **Motion Program** in the pane, and then select **Group Definition** from the menu.



- Set the detailed settings for the axes to use on the Axis Specification Tab Page and click the **OK** Button.

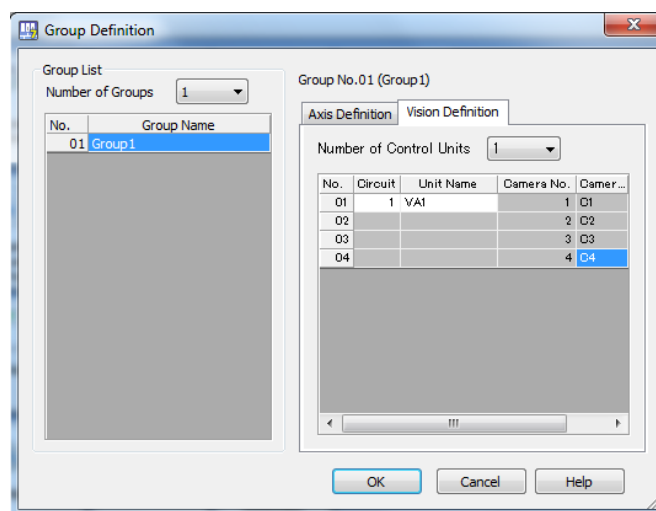
Note: Refer to the following section for details on group definitions.

5.2 Group Definition Details (page 5-9)



Information

The Group Definition Dialog Box also has a Vision Specification Tab Page.



Creating Programs

This section describes creating an example motion program under the following conditions in the Motion Editor.

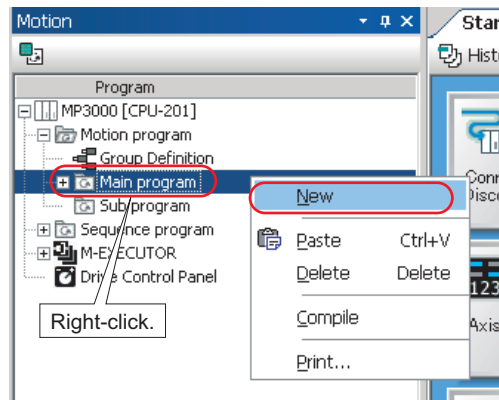
Conditions: Move the Servomotor 150,000 pulses and then stop.



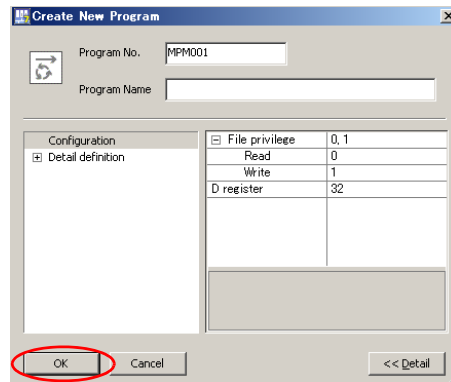
If you perform this task with an actual motor, be sure to set the speed, acceleration time, and travel distance to appropriate values.

Note

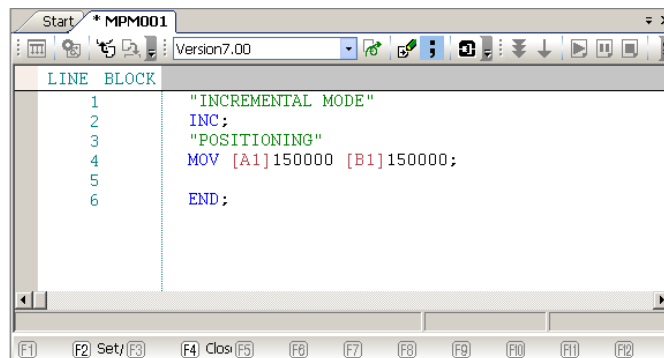
1. Right-click **Main Program** in the pane, and then select **New** from the menu.



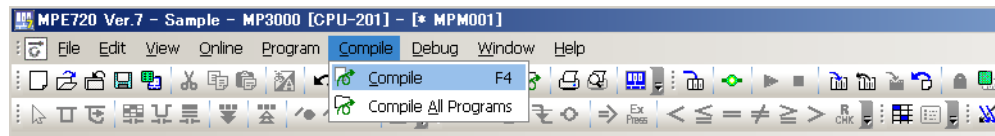
2. Click the **OK** Button.



3. Enter the motion program.



4. Select **Compile – Compile** from the menu bar to compile the program.



When the compilation is finished, the motion program will be saved automatically.




Important

If an error was displayed in the Error List Dialog Box during compilation, the motion program will not be saved.

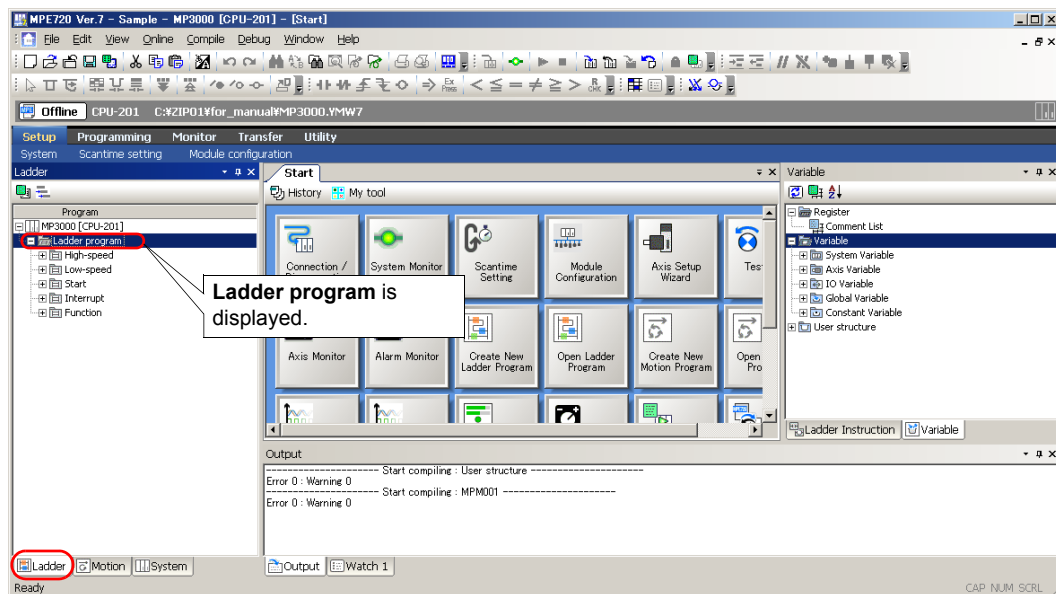
Registering Program Execution

You can call the motion programs that you have created either by using MSEE instructions in ladder programs or by registering the motion programs in the M-EXECUTOR program execution definitions. Refer to the following section for details on how to register a program for execution.

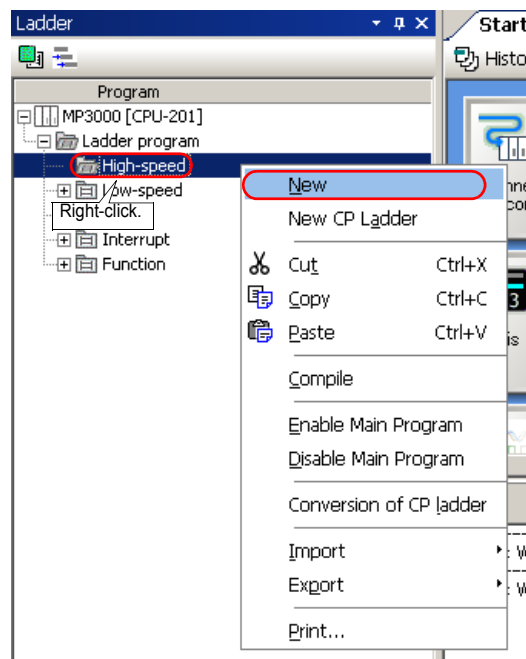
 *Program Execution Registration Methods (page 1-22)*

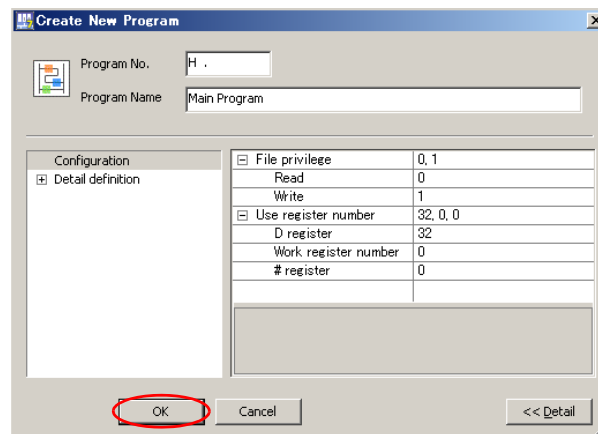

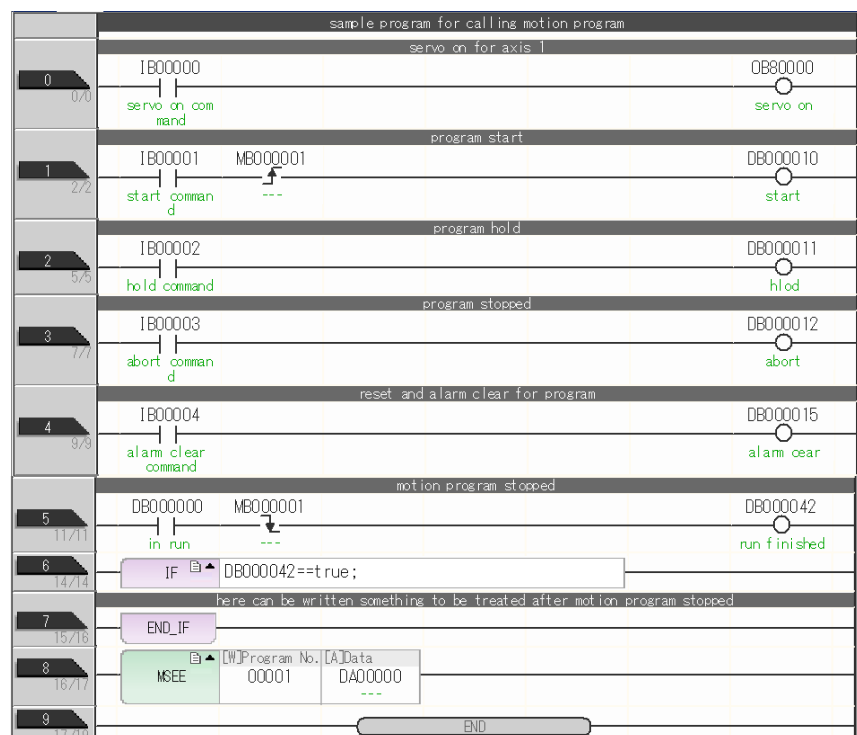
Calling Motion Programs from a Ladder Program

1. Click the **Ladder** Tab in the pane.
Ladder Program is displayed in the pane.



2. Right-click **High-speed** in the pane, and then select **New** from the menu.




3. Click the **OK** Button.4. Create the following ladder program. After you finish entering the ladder program, compile it by pressing the **F4** key on the keyboard or clicking the [] Icon on the toolbar.

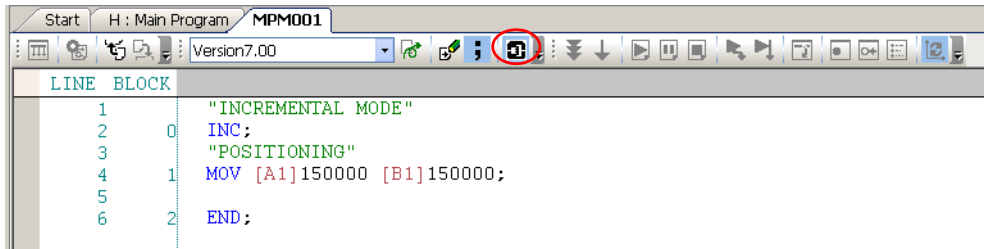
Information

- Make sure that bit 0 (Machine Controller Operation Ready) in the IW□□□00 monitor parameter is ON before turning ON the MB000000 (Servo ON command).
- If the Machine Controller Operation Ready bit is OFF, the Servo ON command cannot be accepted.

Calling a Motion Program with the M-EXECUTOR

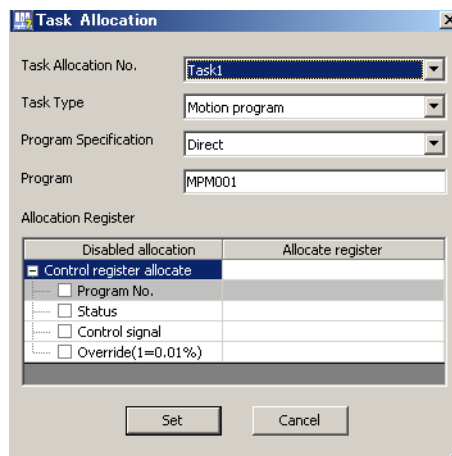
Use the following procedure to register a program in the M-EXECUTOR program execution definitions. However, be sure to transfer the program before performing this procedure.

1. Click the **Assign Task** () Icon in the Motion Editor Pane of the completed program.



The Task Allocation Dialog Box will be displayed.

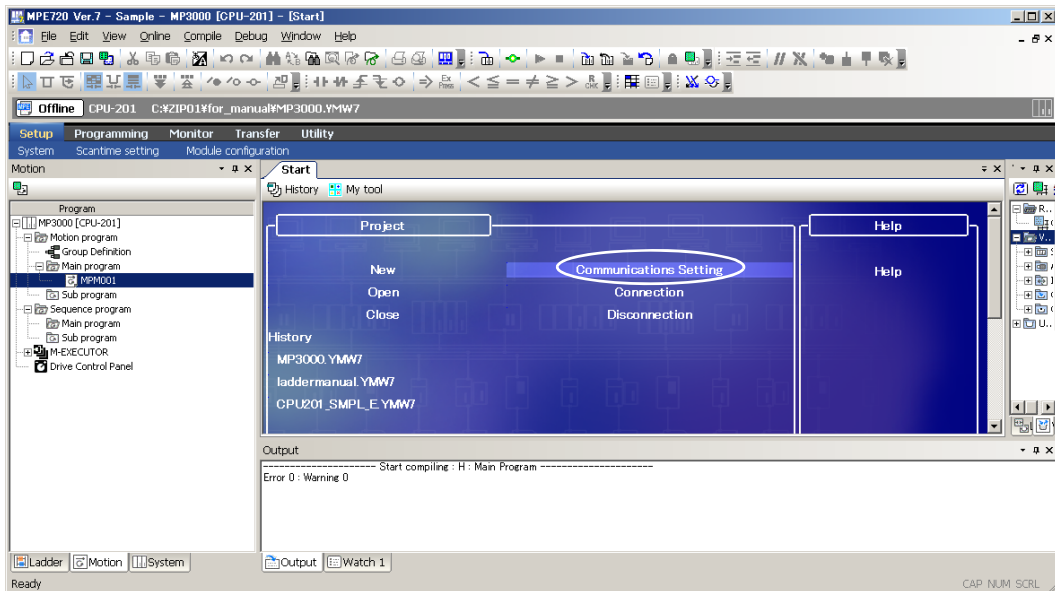
2. Click the **Set** Button to register the program.



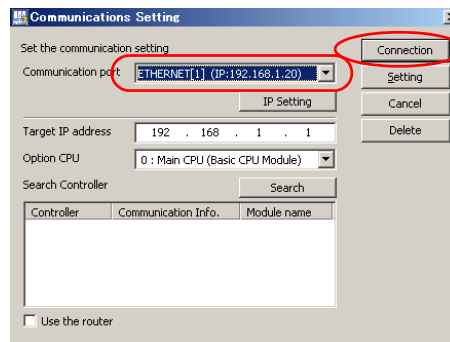
Transferring the Programs

Use the following procedure to transfer motion programs to the MP3000-series Machine Controller. This procedure is not necessary if you created the motion program online.

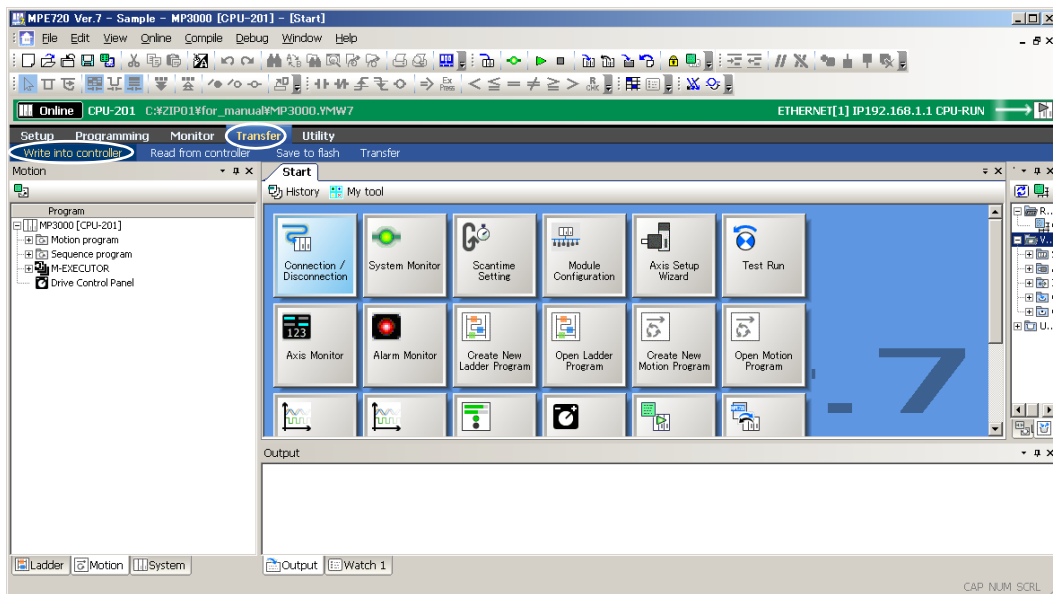
1. Click **Communications Setting** on the Start Tab Page.



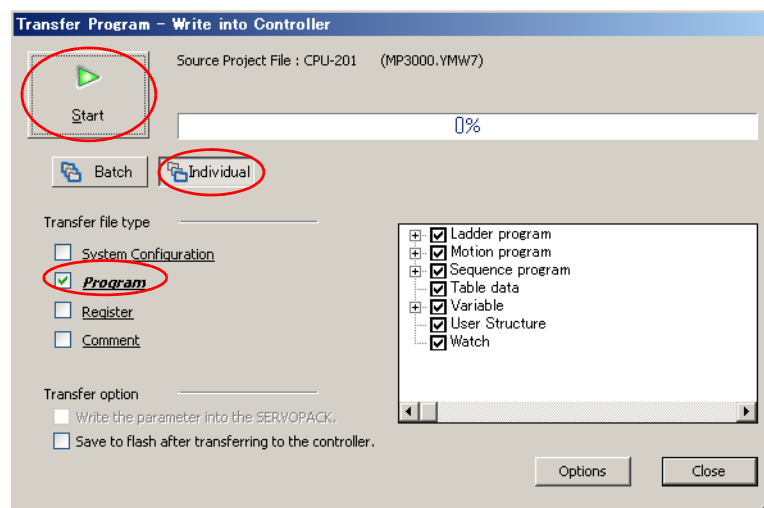
2. Select the desired communications port in the **Communication Port** Box on the Communications Setting Dialog Box. Click the **Connection** Button.



3. Select **Transfer – Write to Machine Controller** from the Launcher.

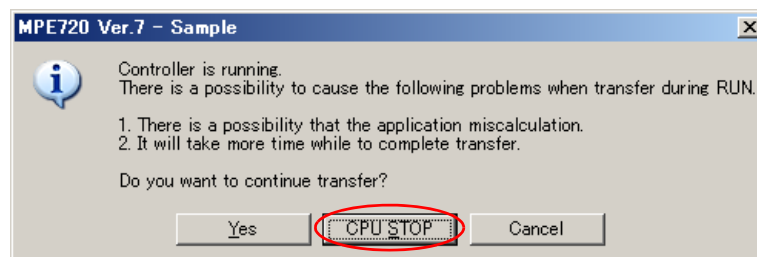


4. Click the **Individual** Button, then select only the **Program** Check Box. Then, click the **Start** Button.



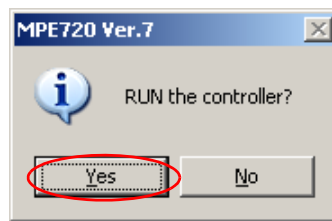
- Note: 1. When an individual transfer is selected, the same file in the Machine Controller will be overwritten with the selected project file data.
- 2. When a batch transfer is selected, the Machine Controller’s RAM will be cleared before the transfer, and all project file data will be written in the RAM.

5. Click the **CPU STOP** Button.



The transfer will start.

6. Click the **Yes** Button in the MPE720 Ver. 7 Dialog Box.



The Machine Controller switches to RUN Mode.

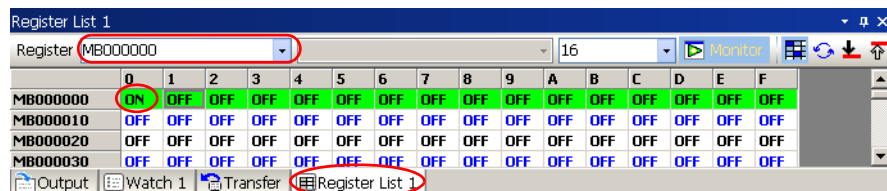
Debugging Programs

Debug the programs that you created.

1. Click the **Register List 1** Tab.

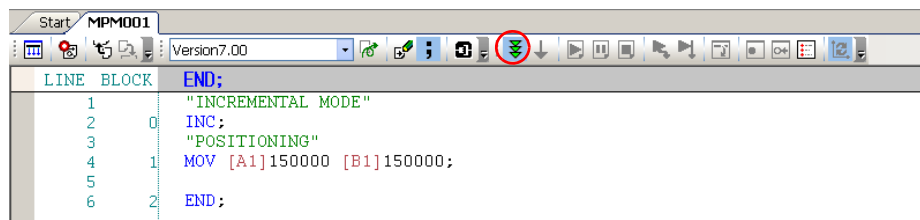
The register list is displayed.

Specify MB000000 for the register. Turn ON MB000000 as shown below to turn ON the power to the Servomotor.

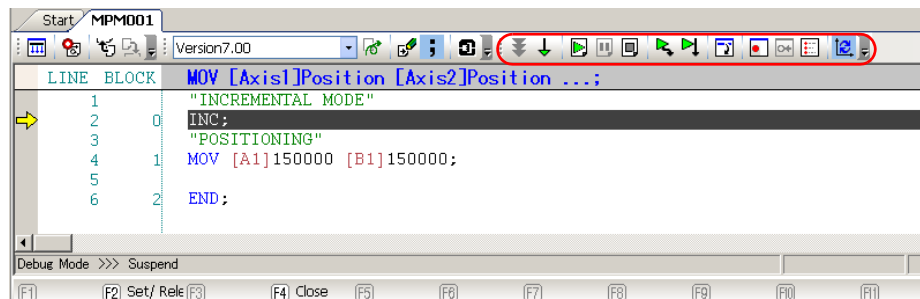


Note: When using the M-EXECUTOR to register the programs for execution, use the setting parameter to turn ON the power to the Servomotor.

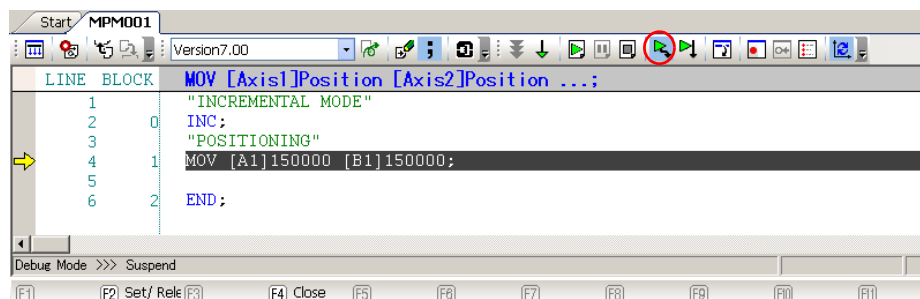
2. Click the Icon.



3. Operation changes to Debug Mode.



4. Click the Icon to execute the program line by line, and check the operation of the program.

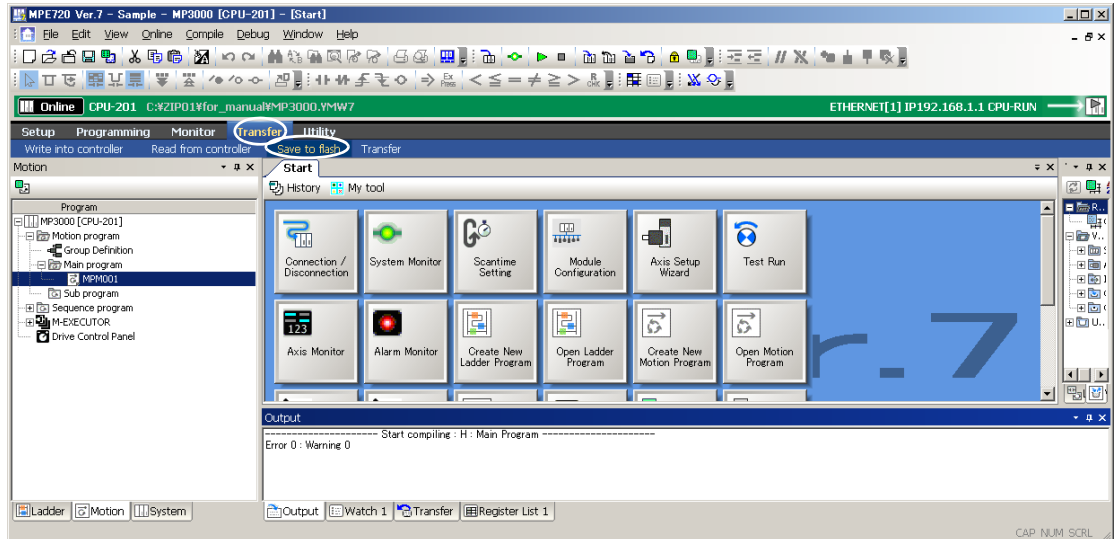


5. Step through the program until the END instruction. When debugging is completed, turn OFF the power to the Servomotor.

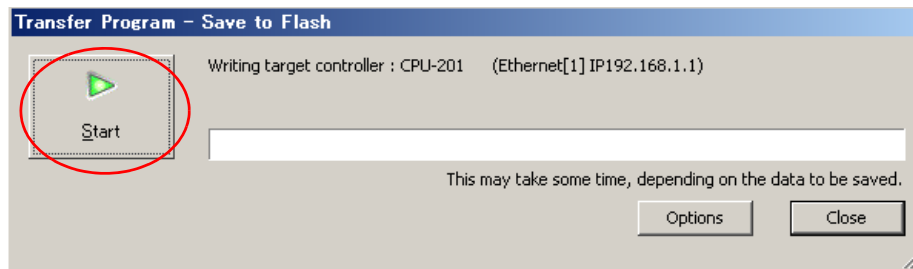
Saving the Programs to Flash Memory

You can save the Machine Controller RAM data to the flash memory of the Machine Controller.

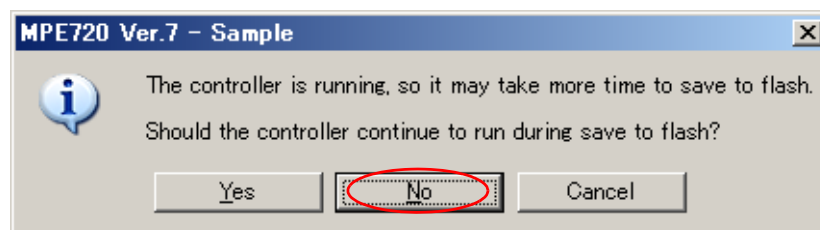
1. Select **Transfer – Save to Flash Memory**.



2. Click the **Start** Button.

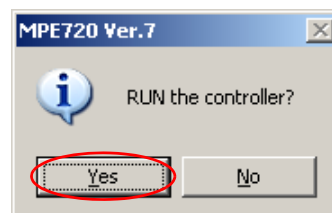


3. Click the **No** Button.



The MPE720 begins saving the data to flash memory.

4. Click the **Yes** Button.



The Machine Controller will switch to RUN Mode.

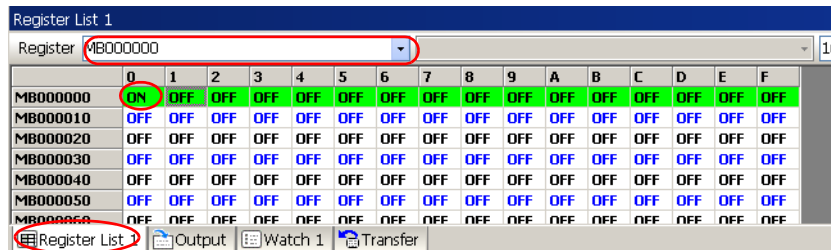
Executing the Programs

Use the following procedure to execute the programs that you created on the actual system. Turn ON the Request for Start of Program Operation Control Signal to execute the motion program.

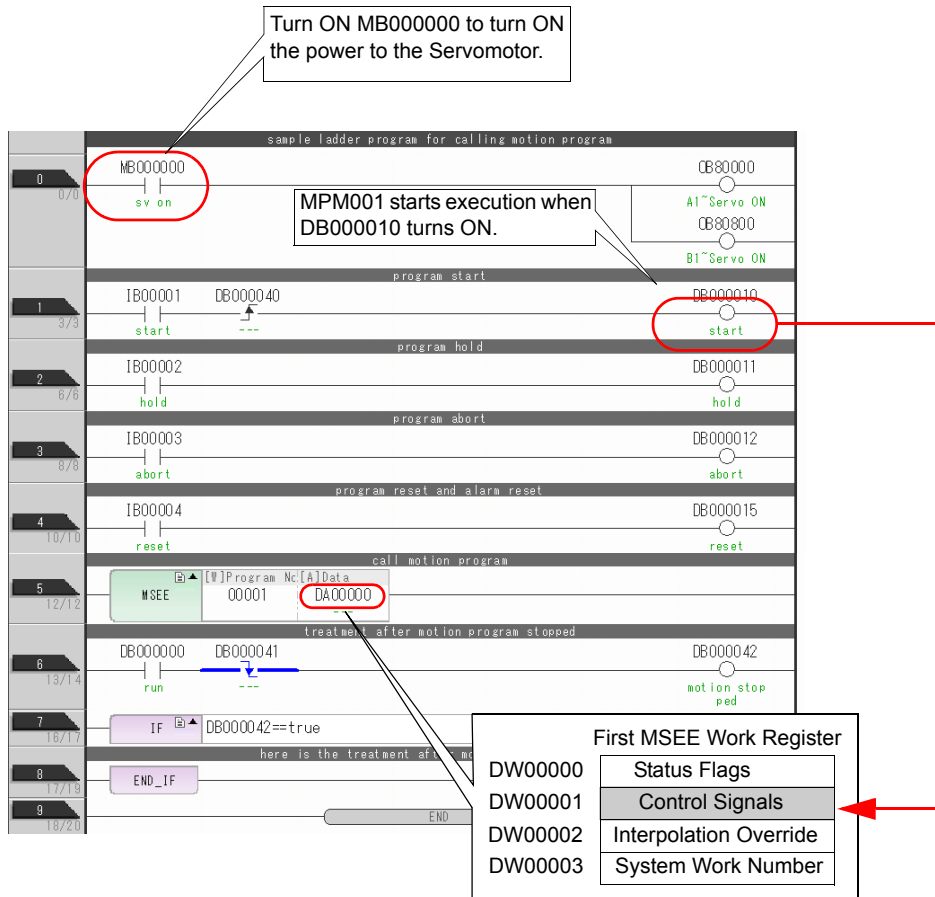
1. Click the Register List 1 Tab.

The register list is displayed.

Specify MB000000 for the register. Turn ON MB000000 as shown below to turn ON the power to the Servomotor.



2. Turn ON MB000001 in the register list to execute the MPM001 motion program.



Registers

4

This chapter describes in detail the registers that you can use in both motion programs and in sequence programs.

4.1	Registers	4-2
	Types of Registers	4-2
	Global Registers	4-5
	Local Registers	4-6
	Data Types	4-8
4.2	Using Registers	4-11
	System Registers (S Registers)	4-11
	Data Registers (M Registers)	4-12
	Data Registers (G Registers)	4-13
	Input Registers (I Registers)	4-14
	Output Registers (O Registers)	4-15
	C Registers	4-16
	D Registers	4-17
4.3	Using Indices i and j	4-18
4.4	Using Array Registers	4-20

4.1

Registers

This section describes registers.

In motion programs and sequence programs, registers are used in place of numeric values. When registers are used in actual operations, the numeric values that are stored in the register area are retrieved.

Types of Registers

There are 11 different types of registers. The types of registers that can be used depend on the program.

The seven types of registers shown in the following table (S, M, G, I, O, C, and D) can be used in motion programs and sequence programs.

S, M, G, I, O, and C registers are global registers that can be used in any program. D registers are local registers that are retained on an individual program basis. D registers are local registers that are retained on an individual drawing basis. They are unique within each drawing, and therefore the value of a D register in one drawing cannot be accessed from another drawing.

Types of Registers

Type	Name	Designation Method	Usable Range	Contents	Features
S	System registers (S registers)	SBnnnnnh, SWnnnnn, SLnnnnn, SQnnnnn, SFnnnnn, SDnnnnn, SAnnnnn	SW00000 to SW65534	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.	Shared by all programs.
M	Data registers (M registers)	MBnnnnnnnh, MWnnnnnnn, MLnnnnnnn, MQnnnnnnn, MFnnnnnnn, MDnnnnnnn, MAnnnnnnn	MW0000000 to MW1048575	These registers are used as interfaces between programs. They have a battery backup.	
G	G registers	GBnnnnnnnh, GWnnnnnnn, GLnnnnnnn, GQnnnnnnn, GFnnnnnnn, GDnnnnnnn, GAnnnnnn	GW0000000 to GW2097151	These registers are used as interfaces between programs. They do not have a battery backup.	
I	Input registers (I registers)	IBhhhhh, IWhhhhh, ILhhhhh, IQhhhhh, IFhhhhh, IDhhhhh, IAhhhhh,	IW00000 to IW07FFF, IW10000 to IW17FFF	These registers are used for input data.	
			IW08000 to IW0FFFF	These registers store the motion monitor parameters. These registers are used for motion control.	
			IW20000 to IW23FFF	These registers are used for CPU interface input data.	

Continued on next page.

Types of Registers

Continued from previous page.

Type	Name	Designation Method	Usable Range	Contents	Features
O	Output registers (O registers)	OBhhhhh, OWhhhhh, OLhhhhh, OQhhhhh, OFhhhhh, ODhhhhh, OAhhhhh,	OW0000 to OW07FFF, OW10000 to OW17FFF	These registers are used for output data.	Shared by all programs.
			OW08000 to OW0FFFF	These registers store the motion setting parameters. These registers are used for motion control.	
			OW20000 to OW23FFF	These registers are used for CPU interface output data.	
C	Constant registers (C registers)	CBnnnnnh, CWnnnnn, CLnnnnn, CQnnnnn, CFnnnnn, CDnnnnn, CAnnnnn	CW00000 to CW16383	These registers can be read in programs but they cannot be written. The values are set from the MPE720.	
D	D registers	DBnnnnnh, DWnnnnn, DLnnnnn, DQnnnnn, DFnnnnn, DDnnnnn, DAnnnnn	DW00000 to DW16383	These registers can be used for general purposes within a program. By default, 32 words are reserved for each drawing.	Program-specific

Continued on next page.

Types of Registers

Continued from previous page.

Type	Name	Designation Method	Usable Range	Contents	Features
#	# registers	#Bnnnnnh, #Wnnnnn, #Lnnnnn, #Qnnnnn, #Fnnnnn, #Dnnnnn, #Annnnn	#W00000 to #W16383	These registers can only be referenced. They can be referenced only within the local drawing.	Function-specific
X	Function input registers	XBnnnnnh, XWnnnnn, XLnnnnn, XQnnnnn, XFnnnnn, XDnnnnn	XW00000 to XW00016	These registers are used for inputs to functions. <ul style="list-style-type: none"> • Bit inputs: XB000000 to XB00000F • Integer inputs: XW00001 to XW00016 • Double-length integers: XL00001 to XL00015 • Quadruple-length integers: XQ00001 to XQ00013 • Real numbers: XF00001 to XF00015 • Double-length real numbers: XD00001 to XD00013 	
Y	Function output registers	YBnnnnnh, YWnnnnn, YLnnnnn, YQnnnnn, YFnnnnn, YDnnnnn	YW00000 to YW00016	These registers are used for outputs to functions. <ul style="list-style-type: none"> • Bit outputs: YB000000 to YB00000F • Integer outputs: YW00001 to YW00016 • Double-length integers: YL00001 to YL00015 • Quadruple-length integers: YQ00001 to YQ00013 • Real numbers: YF00001 to YF00015 • Double-length real numbers: YD00001 to YD00013 	
Z	Function internal registers	ZBnnnnnh, ZWnnnnn, ZLnnnnn, ZQnnnnn, ZFnnnnn, ZDnnnnn	ZW00000 to ZW00016	These are internal registers that are unique within each function. These registers are used for internal processing in functions.	

Note: n: decimal digit, h: hexadecimal digit

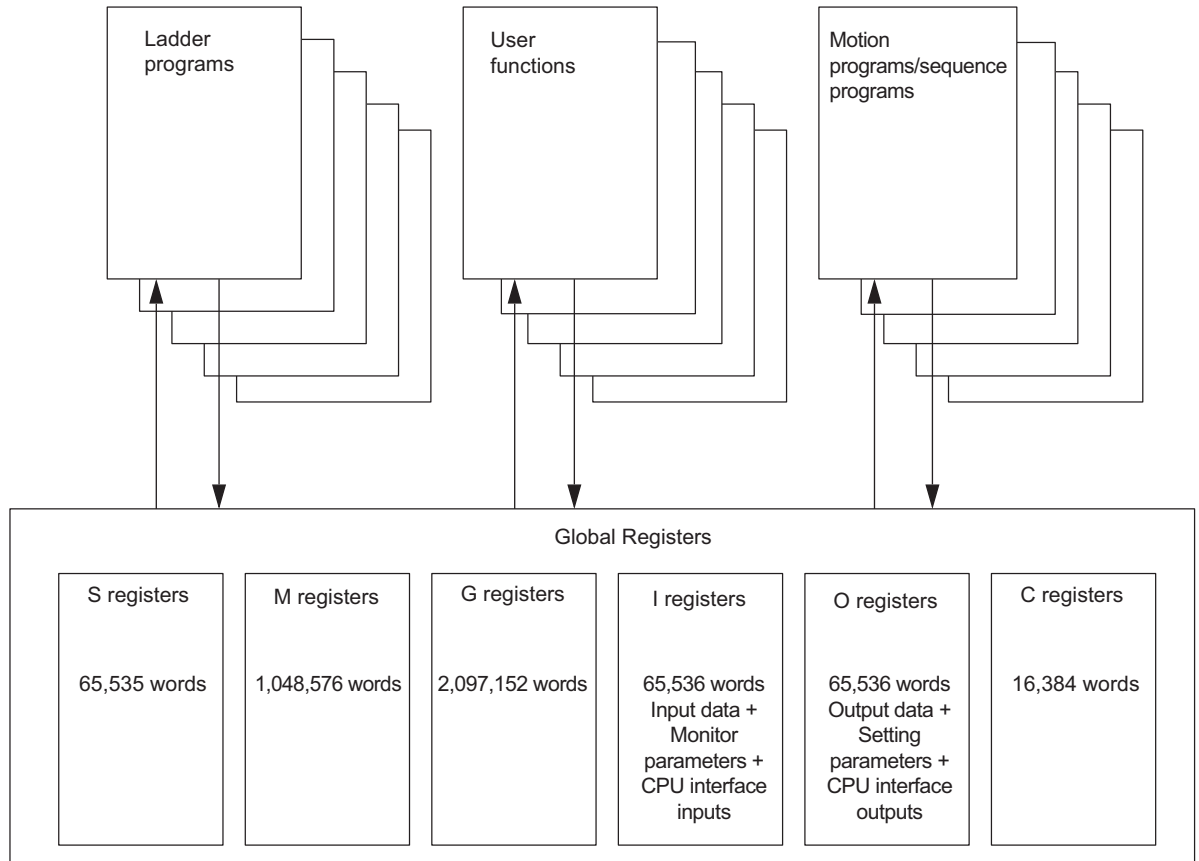


Important

registers cannot be used in motion programs or sequence programs. If you attempt to use a # register in either of these types of programs, a syntax error will occur when the program is saved.

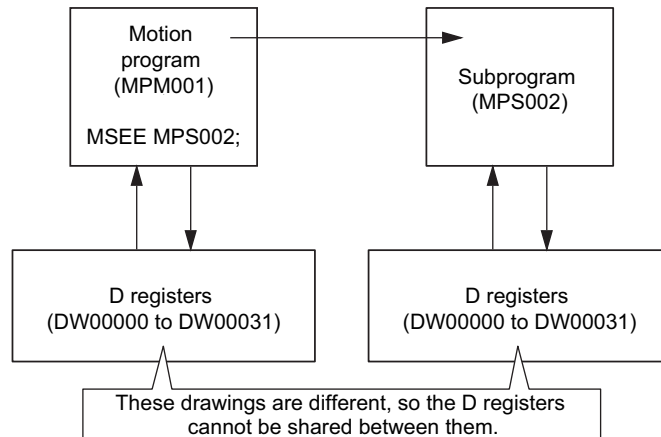
Global Registers

Global registers are shared by ladder programs, user functions, motion programs, and sequence programs. This allows the operation results of a ladder program to be used by other user functions, motion programs, or sequence programs. Memory space for global registers is reserved by the system for each register type.



Local Registers

Local registers can be used within each specific drawing. These registers cannot be shared with other drawings. Local registers are stored in the program memory for each drawing.



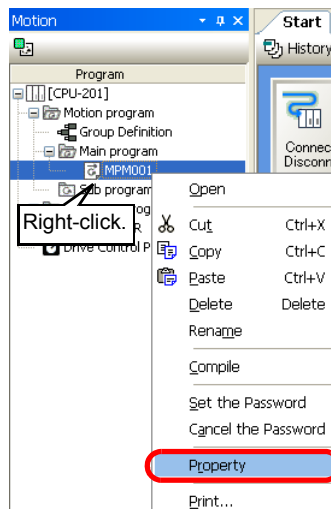
Note: With the default settings, 32 words of D registers are provided for each drawing.

The scope of registers that is used in each drawing is specified in the Program Property Dialog Box.

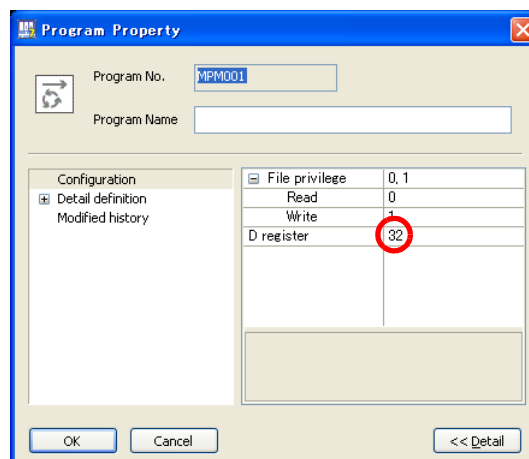
Up to 16,384 words of local registers can be used for one drawing.

Use the following procedure to extend the range of D registers.

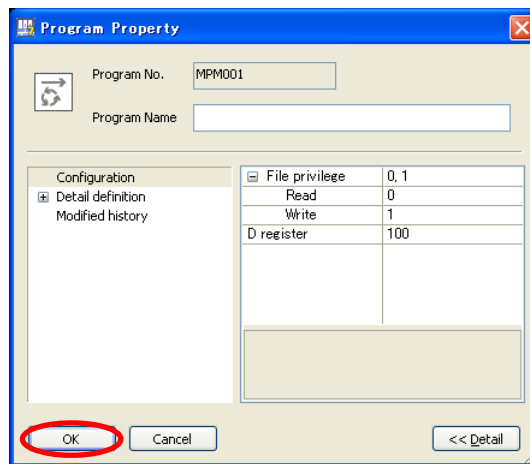
1. Right-click MPM001 in the Motion Pane, and then select **Property** from the menu.



2. Change the range for D registers from 32 to 100 in the Program Property Dialog Box.



3. Click the **OK** Button.



This concludes the procedure to extend the range of D registers.

Data Types

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

Symbol	Data Type	Range of Values	Data Size	Description
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)	1 word	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)	2 words	Used for numeric operations. The values in parentheses on the left are for logical operations.
Q	Quadruple-length integer ^{*1}	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (8000000000000000 to 7FFFFFFFFFFFFFFF hex)	4 words	Used for numeric operations. The values in parentheses on the left are for logical operations.
F	Real number	$\pm (1.175E^{-38}$ to $3.402E^{38})$ or 0	2 words	Used for advanced numeric operations. ^{*2}
D	Double-length real number ^{*1}	$\pm (2.225E^{-308}$ to $1.798E^{+308})$ or 0	4 words	Used for advanced numeric operations. ^{*2}
A	Address	0 to 2,097,152	–	Used only as pointers for addressing.

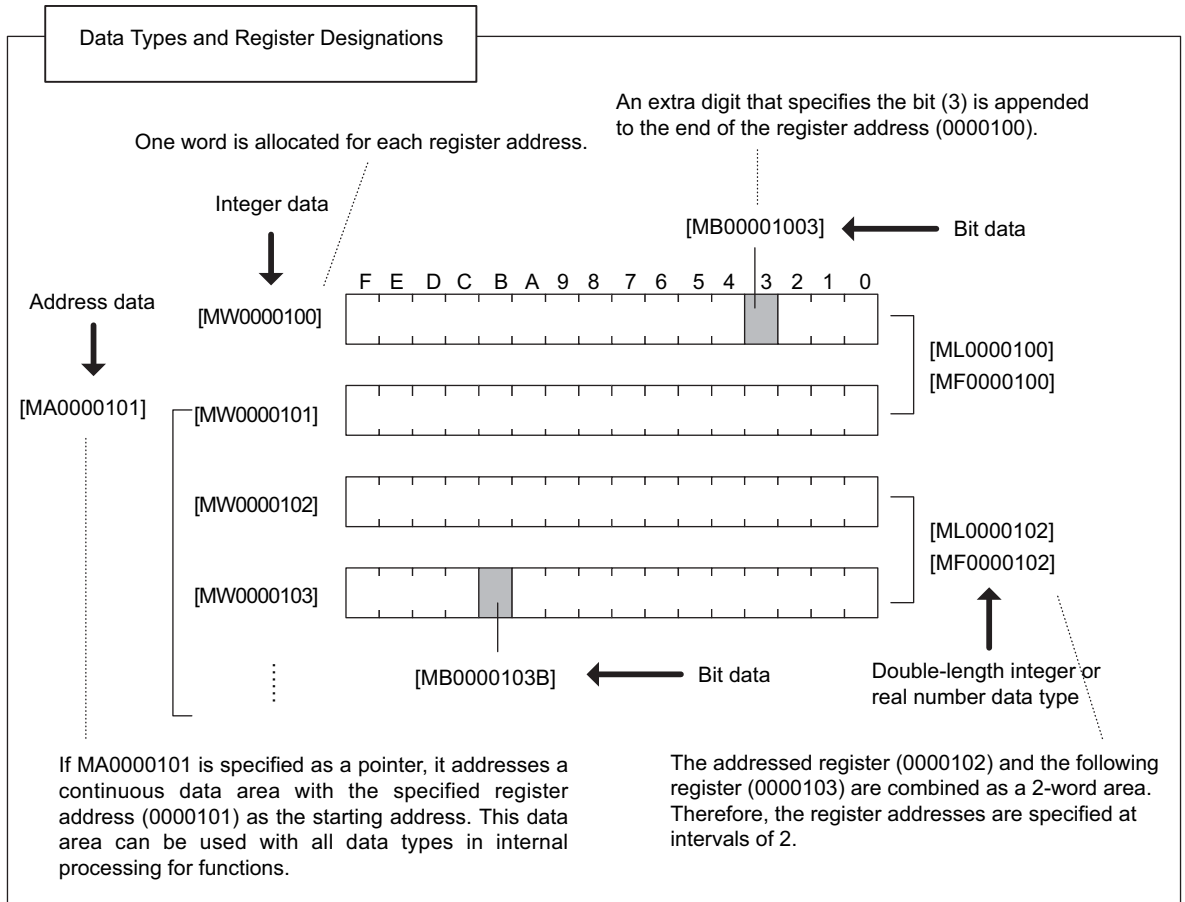
*1. These data types cannot be used for indirect designation of motion programs.

*2. Conforms to IEEE754 standards.



Important

The MP3000-series Machine Controller does 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, MB00001003, a bit address, and the MW0000100, an integer address, have different data types, but they both access the same register, MW0000100.



Terms

Pointer Designation

When an address is passed to a function as a parameter, this is referred to as pointer designation. When pointer designation is used, the continuous data area starting from the address of the specified register number can be used in internal processing for functions with all data types.



Precautions to Consider when Performing Register Operations

The following examples show what occurs if data is stored in a register of a different data type.

- **Format**

Substitute (=) is used for numeric operation instructions.

The destination register is written on the left, and the operation is written on the right.

MW00100 = MW00101 + MW00102;

- **Register Operations**

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 Program Properties Dialog Box.

MW00100 = MF00200 + MF00202;

(0124)(123.48) (0.02) The result of the operation may depend on the value of the register.
(0123)(123.49) (0.01)

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)

Storing Double-length Integer Data in an Integer Register

MW00100 = ML00200; The lower 16 bits of double-length integer data are stored in the destination register as they are.

(-00001)(65535)

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)

- **Examples of Syntax Errors**

Storing Integer Data in a Bit Register

MB000100 = 123;⇒ Syntax error

MB000100 = MW00100;⇒ Syntax error

4.2 Using Registers

This section describes how to use the different types of registers.

System Registers (S Registers)

System registers (S registers) are provided by the MP3000-series Machine Controller system. They can be used to read system error information, the current operating status, and other information.

These registers can be used in any motion program or sequence program.

Details

S registers are specified as follows:

SB000000 to SB65534F
SW000000 to SW65534
SL000000 to SL65532
SQ000000 to SQ65528
SF000000 to SF65532
SD000000 to SD65528

The register number is specified as a decimal number. However, when specifying a bit, the lowest digit of the register number is specified in hexadecimal.

Programming Examples

- Bit Designation
OB00010 = SB000402 | SB000403;
- Integer Designation
MW00100 = SW00041;
- Double-length Integer Designation
ML00100 = SL00062;



Note

The system registers (S) are read-only. If they are written to, system operations will be unpredictable.

Data Registers (M Registers)

M registers are general-purpose registers that can be used in ladder programs, user functions, motion programs, and sequence programs.

They are global registers that can be used to interface between motion programs, sequence programs, and ladder programs.

Details

M registers are specified as follows:

MB0000000 to MB1048575F

MW0000000 to MW1048575

ML0000000 to ML1048574

MQ0000000 to MQ1048572

MF0000000 to MF1048574

MD0000000 to MD1048572

M registers can be used in operations to store the operation results, or specified to give positioning coordinate values or speeds. The register number is specified as a decimal number.

Programming Examples

◆ Specifying the Position and Speed in Axis Movement Instructions with Registers

In the following programming example, the reference unit is mm and the number of digits below the decimal point is set to 3.

ML0000100 = 100000;	→ 100.000 mm
ML0000102 = 200000;	→ 200.000 mm
ML0000104 = 300000;	→ 300.000 mm
ML0000106 = 500000;	→ 500.000 mm/min

MVS [X]ML0000100 [Y]ML0000102 [Z]ML0000104 FML0000106;

◆ Using Registers in Operations

- Bit Designation
MB00001001 = IB0000100 & IB0000201;
- Integer Designation
MW0000101 = (MW0000101 | MW0000102) & FF0CH;
- Double-length Integer Designation
ML0000200 = ((ML0000202 * ML0000204) / ML0000206) * 3;
- Real Number Designation
MF0000200 = ((MF0000202 * MF0000204) / MF0000206) * 3.14;



Important

When the travel distance coordinate values or speeds are specified in registers in the following motion language instructions, double-length integer data must be used.

MOV, MVS, MCW/MCC, ZRN, SKP, MVT, EXM, POS, ACC, DCC, SCC, IAC, IDC, IFP, FMX, INP, IDH

Data Registers (G Registers)

Data registers (G registers) are general-purpose registers that can be used in ladder programs, user functions, motion programs, and sequence programs.

They are global registers that can be used in any motion program or sequence program, but are not backed up by battery.

Details

G registers are specified as follows:

GB0000000 to GB2097151F
 GW0000000 to GW2097151
 GL0000000 to GL2097150
 GQ0000000 to GQ2097148
 GF0000000 to GF2097150
 GD0000000 to GD2097148

The register number is specified as a decimal number. However, when specifying a bit, the lowest digit of the register number is specified in hexadecimal.

Programming Examples

The following example shows how to use these registers in operations.

- **Bit Designation**
 $GB00001001 = IB0000100 \& IB0000201;$
- **Integer Designation**
 $GW0000101 = (GW0000101 | GW0000102) \& FF0CH;$
- **Double-length Integer Designation**
 $GL0000200 = ((GL0000202 * GL0000204) / GL0000206) * 3;$
- **Real Number Designation**
 $GF0000200 = ((GF0000202 * GF0000204) / GF0000206) * 3.14;$



Important

When the travel distance coordinate values or speeds are specified in registers in the following motion language instructions, double-length integer data must be used.

MOV, MVS, MCW/MCC, ZRN, SKP, MVT, EXM, POS, ACC, DCC, SCC, IAC, IDC, IFP, FMX, INP, IDH

Input Registers (I Registers)

These registers are used for input data and for monitor parameters. Monitor parameters are read-only. If they are written to, operations will be unpredictable.

Details

I registers are specified as follows:

IB000000 to IB23FFFF

IW00000 to IW07FFF, IW10000 to IW17FFF ... Input data

IW08000 to IW0FFFF ... Monitor parameter

IW20000 to IW23FFF ... CPU interface input data

IL00000 to IL23FFF

IQ00000 to IQ23FFC

IF00000 to IF23FFE

ID00000 to ID23FFC

The register addresses of input data depend on the addresses set in the Module configuration definition.

The register number is specified as a hexadecimal number.

Programming Examples

This example shows how to read input data and monitor parameters.

- **Bit Designation**
MB00001000 = IB0000010 & IB0000105;
- **Integer Designation**
MW0000100 = IW08008;
- **Double-length Integer Designation**
ML0000100 = IL08004;

Output Registers (O Registers)

These registers are used for output data and for setting parameters.

Details

O registers are specified as follows:

OB000000 to OB23FFFF

OW00000 to OW07FFF, OW10000 to OW17FFF ... Output data

OW08000 to OW0FFFF ... Setting parameters

OW20000 to OW23FFF ... CPU interface output data

OL00000 to OL23FFF

OQ00000 to OQ23FFC

OF00000 to OF23FFE

OD00000 to OD23FFC

The register addresses of output data depend on the addresses set in the Module configuration definition.

The register number is specified as a hexadecimal number.

Programming Example

This example writes output data and setting parameters.

- Bit Designation
OB01000 = MB00001000 & IB0000100;
- Integer Designation
OW08008 = MW0000100;
- Double-length Integer Designation
OL08010 = ML0000100+ML0000200;

C Registers

C registers can be referenced only from a program. They are read-only.

Details

C registers are specified as follows:

CB000000 to CB16383F

CW000000 to CW16383

CL000000 to CL16382

CQ000000 to CQ16380

CF000000 to CF16382

CD000000 to CF16380

C registers cannot be written to from a program.

The register number is specified as a decimal number.

Programming Example

The following example shows how to use these registers in operations.

- **Bit Designation**
MB00001000 = CB001001;
- **Integer Designation**
MW0000100 = CW00100;
- **Double-length Integer Designation**
ML0000100 = CL00100;
- **Quadruple-length Integer Designation**
MQ0000100 = CQ00100;
- **Real Number Designation**
MF0000100 = CF00100;
- **Double-length Real Number Designation**
MD0000100 = CD00100;

D Registers

These registers are unique, internal registers for motion programs and sequence programs. They are unique within each program.

Details

D registers are specified as follows:

DB000000 to DB16383F
 DW000000 to DW16383 (maximum value)
 DL000000 to DL16382
 DQ000000 to DQ16380
 DF000000 to DF16382
 DD000000 to DD16380

The above registers can be used in operations to store operation results, or specified to give positioning coordinate values or speeds. The register number is specified as a decimal number. However, when specifying a bit, the lowest digit of the register number is specified in hexadecimal. Specify the size in the program configuration definitions (i.e., the Program Properties Dialog Box). The default size is 32 words.

Programming Example

◆ Specifying the Position and Speed in Axis Movement Instructions with Registers

In the following example, the reference unit is mm and the number of digits below the decimal point is set to 3.

DL00100 = 100000;	→ 100.000 mm
DL00102 = 200000;	→ 200.000 mm
DL00104 = 300000;	→ 300.000 mm
DL00106 = 500000;	→ 500.000 mm/min

MVS [A1]DL00100 [B1]DL00102 [C1]DL00104 FDL00106;

◆ Using Registers in Operations

- Bit Designation
 $DB001000 = IB0001001 \& MB00000101;$
- Integer Designation
 $DW00102 = (CW00103 | DW00104) \& DW00105;$
- Double-length Integer Designation
 $DL00106 = (DL00108 * ML0000011) / ML0000200;$
- Real Number Designation
 $DF00200 = (MF0000202 * DF00202) * 3.14;$



Important

When the travel distance coordinate values or speeds are specified in registers in the following motion language instructions, double-length integer data must be used.

MOV, MVS, MCW/MCC, ZRN, SKP, MVT, EXM, POS, ACC, DCC, SCC, IAC, IDC, IFP, FMX, INP, IDH

4.3 Using Indices i and j

There are two special registers, i and j, that are used to modify relay and register addresses. The functions of i and j are identical. They are used to handle register addresses like variables.

We will describe this with examples for each register data type.

■ 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, MB00000000i is the same as MB00000002.

i = 2;
 DB000000 = MB00000000i; \longleftrightarrow DB000000 = MB00000002;

■ 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 j = 30, MW0000001j is the same as MW00000031.

j = 30;
 DW000000 = MW0000001j; \longleftrightarrow DW000000 = MW00000031;

■ 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, ML0000000j is the same as ML00000001. Similarly, if j = 1, MF0000000j is the same as MF00000001.

Double-length Integer	Upper Word	Lower Word
	MW00000001	MW00000000
If j = 0, ML0000000j is ML00000000.		
	MW00000002	MW00000001
If j = 1, ML0000000j is ML00000001.		
Real Number	Upper Word	Lower Word
	MW00000001	MW00000000
If j = 0, MF0000000j is MF00000000.		
	MW00000002	MW00000001
If j = 1, MF0000000j is MF00000001.		



Note

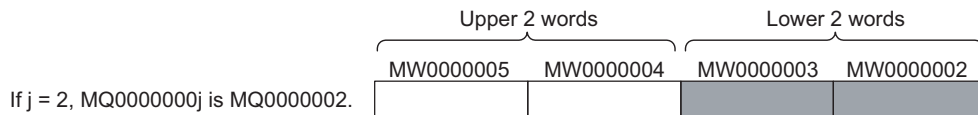
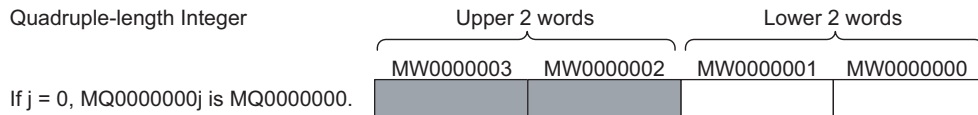
- Double-length integers and real numbers use a region that is 2 words in size. For example, when using ML0000000j with both j = 0 and j = 1, the one-word area of MW00000001 will overlap. Be careful of overlapping areas when indexing double-length integer or real number register addresses.
- The setting range for indices i and j is - 2,147,483,648 to 2,147,483,647.

■ Attaching an Index to a Quadruple-length Integer or a Double-length Real Number Register

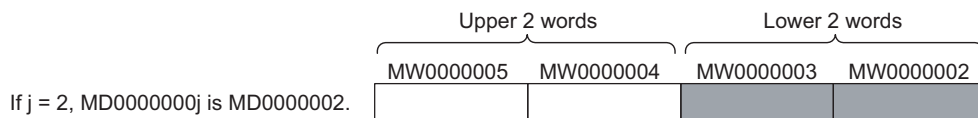
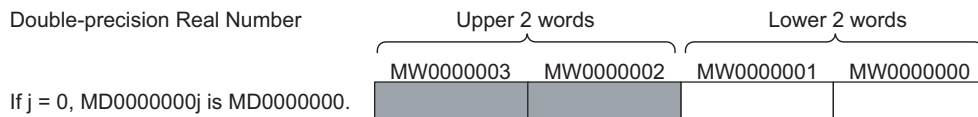
Using an index is the same as adding the value of i or j to the register address.

For example, if $j = 2$, MQ0000000j is the same as MQ0000002. Similarly, if $j = 2$, MD0000000j is the same as MD0000002.

Quadruple-length Integer



Double-precision Real Number



Note

Quadruple-length integers and double-precision real numbers use a region that is 4 words in size. For example, when using MQ0000000j with both $j = 0$ and $j = 2$, the two-word area of MW0000002 and MW0000003 will overlap. Be careful of overlapping areas when indexing quadruple-length integer or double-length real number register addresses.

■ Programming Examples

The following programming example uses indices.

Subscript j is used to calculate the total amount of 50 registers from ML0000100 to ML0000198. That amount is then stored in ML0000200.

```

:
:
ML0000200 = 0 ;
J = 0 ;
WHILE J < 100 ;
  ML0000200 = ML0000200 + ML0000100J ;
  J = J + 2 ;
WEND ;
:
:

```

Information

Indices i and j can be specified in either lowercase or uppercase letters.

```

j = 0 ;
J = 0 ;
DW00000 = MW0000000j ;
DW00000 = MW0000000J ;

```


4.4 Using Array Registers

Array registers are used to modify register addresses.

They are used to handle register addresses like variables.

As with indices, an offset can be added to the register address.

■ Attaching an Array Register to a Bit Register

Using an array register is the same as adding the value of the array register to the register address.

For example, if $DW00000 = 2$, $MB00000000[DW00000]$ is the same as $MB00000002$.

$DW00000 = 2;$
 $DB000020 = MB00000000[DW00000];$ Equivalent $DB000020 = MB00000002;$

■ Attaching an Array Register to a Register Other Than a Bit Register

Using an array register is the same as adding the word size of the data type of the array register times the value of the array register to the register address.

For example, if $DW00000 = 30$, $ML0000002[DW00000]$ is the same as $ML0000062$.

$DL00002 = ML00000 (30 \times 2 + 2) = ML0000062$

$DW00000 = 30;$
 $DL00002 = ML0000002[DW00000];$ Equivalent $DL00002 = ML0000062;$

■ Format

This section describes the formats of array registers.

$MOV[A1]ML00000[MW00100];$
 ① ②

Description	Use	Usable Registers
①	Array name	• All registers with any data type (excluding # and C registers)
②	Array elements	• All registers with integer and double-length integer data types (excluding # and C registers) • Constant • Subscript registers

■ Programming Examples

In the following example, an array register is used to calculate the total amount of 50 registers from $ML0000100$ to $ML0000198$. That amount is then stored in $ML0000200$.

```
ML0000200 = 0;
DW00000 = 0;
WHILE DW00000 < 50;
  ML0000200 = ML0000200 + ML0000100[DW00000];
  DW00000 = DW00000 + 1;
WEND;

END;
```

Programming Rules

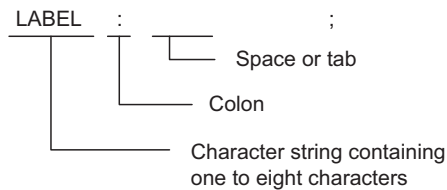
5

This chapter describes rules that must be followed when creating motion programs and sequence programs.

5.1	Entering Programs	5-2
	Motion Program Structure	5-2
	Block Format	5-2
	Notation for Constants and Registers	5-8
5.2	Group Definition Details	5-9
5.3	Operation Priority Levels	5-11
5.4	Instruction Types and Execution Scans	5-13
	Instruction Types	5-13
	Instruction Type Table	5-15
5.5	Programming with Variables	5-17
	Declaring Variables	5-17
	Variable Format	5-18
	Strings That Cannot Be Used in Variable Names	5-20
	Programming Examples	5-21

Labels

A label consists of a character string of one to eight alphanumeric characters or symbols, a colon (:), and a space or TAB.



Type	Valid Label Characters and Symbols
Letters	A to Z , a to z
Numbers*	0 to 9
Symbol	- (Hyphen)

*Labels cannot start with a number.

Labels are required when using the PFORK (parallel execution) or SFORK (selective execution) instructions. You do not need to use labels if the PFORK or SFORK instructions are not used.

Example


Label Notation Examples

```
PFORK LAB1, LAB2;
LAB1: ZRN [A1]0 [B1]0 [C1]0;
JOINTO LAB3;
LAB2: MVS [D1]100.0 [E1]200.0 [F1]300.0;
JOINTO LAB3;
LAB3: PJOINT;
```

Motion Language Instructions

This is where the motion language instruction is given.

Refer to the following chapter for details on the motion language instructions.

 Chapter 6 Motion Language Instructions

Logical Axis Names

Give the logical axis name that is set in the group definition in square brackets ([]).

MVS[A1]20.0 ;
 Logical axis name
 character string containing
 one to eight characters

Type	Valid Logical Axis Name Characters
Letters	A to Z, a to z
Numbers	0 to 9

Coordinate Words

A coordinate word is a numerical value or a variable that is placed after an axis name. A coordinate word specifies the reference position, speed, acceleration/deceleration, and other information.

◆ Using a Numeric Value for the Coordinate Word

Write a numerical value after the axis name to directly specify the coordinate word.

Both integers and real numbers can be used for the numerical value. However, special care must be taken when using integers.

For example, when the reference unit is set to 0.001 mm and a reference position of 1,000 is entered for the coordinate word, the Machine Controller interprets this as 1.000 mm. When you enter 1.000 as a real number, the Machine Controller interprets it as 1.000 mm.

MVS [A1]1000;→ 1.000 mm

Or

MVS [A1]1.000;→ 1.000 mm

Or

MVS [A1]1.;→ 1.000 mm

◆ Using a Register for the Coordinate Word

Write a double-length integer register address after the axis name to indirectly specify the coordinate word.

When the reference unit is set to 0.001 mm with indirect designation using a register, and the register value is set to 1000, the Machine Controller interprets the coordinate word as 1.000 mm in the same way as in the previous example.

ML00000 = 1000;

MVS [A1]ML00000;→ 1.000 mm

Information

The coordinate word unit depends on the motion language instruction and the motion parameter settings.

◆ Specific Characters

The meaning of each special character is given in the following table along with some usage examples.

Character	Meaning	Usage Example	Reference
M	Acceleration/ deceleration mode	ACCOMDE M2;	Set Interpolation Acceleration/Deceleration Mode (ACCMODE)
F	Interpolation feed speed	MVS [A1]1000 [B1]2000 F3000000; MVS [A1]1000 [B1]2000 FML00000;	Linear Interpolation (MVS)
FW	Continuous process control signal	MVS [A1]1000 FWMB00000000;	Linear Interpolation (MVS)
T	Maximum interpolation feed speed	FMX T30000000; FMX TML00000;	Set Maximum Interpolation Feed Speed (FMX)
	Time setting	TIM T100; TIM TML00000; TIM1MS T100; TIM1MS TMW00000000; MVT [A1]1000 [B1]2000 T100; MVT [A1]1000 [B1]2000 TML00000; IAC T100; IAC TML00000; IDC T100; IDC TML00000; IDH T100; IDH TML00000;	Dwell Time (TIM) Dwell Time (TIM1MS) Set-time Positioning (MVT) Change Interpolation Acceleration Time (IAC) Change Interpolation Deceleration Time (IDC) Interpolation Deceleration Time for Temporary Stop (IDH)
	Number of turns for circular interpolation	MCW [A1]1000 [B1]2000 U500 V500 T2 F3000000; MCW [A1]1000 [B1]2000 U500 V500 TML00000 F3000000;	Circular Interpolation with Specified Center Point (MCW and MCC) Circular Interpolation with Specified Radius (MCW and MCC)
TW	Continuous process control signal	MVS [A1]1000 TWMB00000000;	Linear Interpolation (MVS)
R	Radius of circle	MCW [A1]1000 [B1]2000 R500 F3000000; MCW [A1]1000 [B1]2000 RML00000 F3000000;	Circular Interpolation with Specified Center Point (MCW and MCC) Circular Interpolation with Specified Radius (MCW and MCC)
U	Circle center point coordinate 1 (horizontal axis)	MCW [A1]1000 [B1]2000 U500 V500 T2 F3000000; MCW [A1]1000 [B1]2000 UML00000 V500 T2 F3000000;	Circular Interpolation with Specified Center Point (MCW and MCC) Circular Interpolation with Specified Radius (MCW and MCC)
V	Circle center point coordinate 2 (vertical axis)	MCW [A1]1000 [B1]2000 U500 V500 T2 F3000000; MCW [A1]1000 [B1]2000 U500 VML00000 T2 F3000000;	Circular Interpolation with Specified Center Point (MCW and MCC) Circular Interpolation with Specified Radius (MCW and MCC)
P	Interpolation feed speed specified by percentage	IFP P50; IFP PML00000;	Set Interpolation Feed Speed Ratio (IFP)


Continued on next page.

Comments

There are two symbols that can be used to enter comments: quotation marks (“”) and double forward slashes (/).

◆ Using Double Forward Slashes for Comments

All characters from the double forward slash to the next line feed are interpreted as a comment.

Line feed
// Text string 

Example Comment Notation Example 2
// Perform a zero point return for all axes.
ZRN [A1]0 [B1]0 [C1]0;

// Linear interpolation of three axes.
MVS [A1]100.0 [B1]200.0 [C1]300.0;

Information Comments can include double-byte characters as well as single-byte alphanumeric characters.

◆ Using Double Quotation Marks for Comments

• Enclosing a Text String in Double Quotation Marks


A character string enclosed in double quotation marks is interpreted as a comment.

“Text string”

Example Comment Notation Example 1
ZRN [A1]0 [B1]0 [C1]0; “Perform a zero point return for all axes.”
MVS [A1]100.0 [B1]200.0 [C1]300.0; “Linear interpolation of three axes.”

• Placing a Text String after One Double Quotation Mark

All characters from the double quotation mark to the next line feed are interpreted as a comment.

Line feed
“Text string 

Example Comment Notation Example 2
“Perform a zero point return for all axes.
ZRN [A1]0 [B1]0 [C1]0;

“Linear interpolation of three axes.
MVS [A1]100.0 [B1]200.0 [C1]300.0;

Information Comments can include double-byte characters as well as single-byte alphanumeric characters.

Notation for Constants and Registers

This section describes how to use constants and registers.

Constants

The constants that you can use in motion programs are listed in the following table.

Type	Range	Notation Examples
Decimal integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	823, -2493, 123k, 123K
Hexadecimal integer	0000000000000000 hex to FFFFFFFF hex	FFFABCDE hex, 2345 hex, F hex
Real number	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Changes according to the setting of the number of digits below the decimal point.	763.0, 824.2, 234.56, -321.12345

Information

- The - (minus) sign cannot be omitted, but the + (plus) sign can be omitted.

[A1]+123 ⇒ [A1]123

[A1]-123 ⇒ [A1]-123

- A decimal integer is multiplied by 1,000 by adding K to the value. For a value such as position reference where there are many zeroes, using a K can make it easier to read.

[A1]123k ⇒ [A1]123000

[A1]123K ⇒ [A1]123000

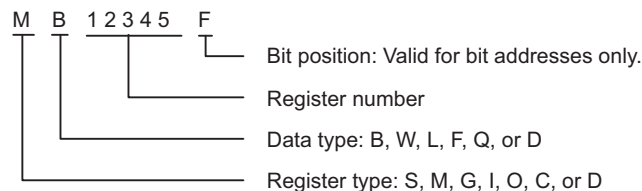
Register Notation

The registers that you can use in motion programs are listed in the following table.

Type	Register Type	Data Type					
		BIT	WORD	LONG	FLOAT	QUAD	DOUBLE
Global Registers	S registers	SB	SW	SL	SF	SQ	SD
	M registers	MB	MW	ML	MF	MQ	MD
	G registers	GB	GW	GL	GF	GQ	GD
	I registers	IB	IW	IL	IF	IQ	ID
	O registers	OB	OW	OL	OF	OQ	OD
	C registers	CB	CW	CL	CF	CQ	CD
Local Registers	D registers	DB	DW	DL	DF	DQ	DD

Example

Register Notation Example



Zeroes cannot be omitted in some constants and registers.

Example

Examples Where Zeroes Can Be Omitted

[A1]00123 ⇒ [A1]123

[A1]MW00010 ⇒ [A1]MW10

[A1]100.000 ⇒ [A1]100.

Example

Examples Where Zeroes Cannot Be Omitted

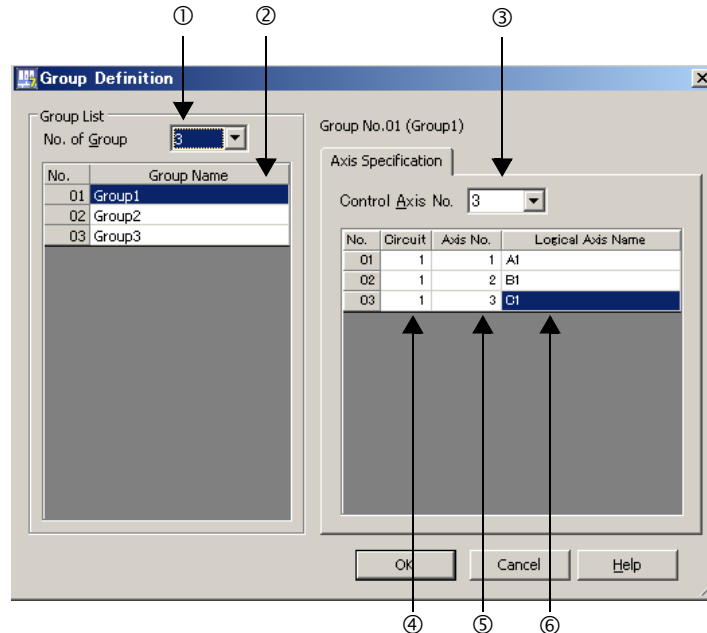
MPM001; (Program number at the beginning of a program)

MSEE MPS002;

5.2 Group Definition Details

A group definition allows you to treat more than one axis as a single group.

This section describes the Group Definition Dialog Box.



- ① **Number of Groups**
Set the number of groups for group operation.
Set this number to 1 for operation with one group.
For operation with more than one groups, set the number of groups for group operation.
- ② **Group Name**
Give the name of the group.
- ③ **Number of Controlled Axes**
Set the number of axes to control as a group.
- ④ **Circuit**
Set the circuit number for the Motion Control Function Module to use.
The circuit number can be checked in the Module configuration definition.

Circuit Number

Module	Function Module/Slave	Status	Circuit No/Axis Address	Motion Register	Register (Input/Output)	Disabled	Start - End	Size	Scan	Comm
01 CPU-201 : ---										
--- UNDEFINED ---										
PSA-12										
102-PLCD 00	01 CPU	Driving								
	02 218FD	Driving	Circuit No1	1		<input type="checkbox"/> Input <input type="checkbox"/> OutPut	0000 - 07FF[H]	2048		
	03 SVC32	Driving	Circuit No2	2	8000 - 8FFF[H]	<input type="checkbox"/> Input <input type="checkbox"/> OutPut	0800 - 0BFF[H]	1024		
	04 SVR32	Driving	Circuit No3	2	9000 - 9FFF[H]					
	05 M-EXECUTOR	Driving					0C00 - 0C7F[H]	128		
06 --- UNDEFINED ---										
01 --- UNDEFINED ---										
02 --- UNDEFINED ---										
03 --- UNDEFINED ---										
04 --- UNDEFINED ---										
05 --- UNDEFINED ---										
02 --- UNDEFINED ---										
03 --- UNDEFINED ---										
04 --- UNDEFINED ---										

⑤ Axis Number

Select the axis numbers of the axes to use.

You can check the axis numbers by clicking the + Button next to SVC32 in the Module Configuration.

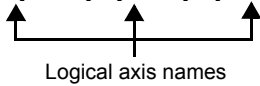
Module	Function Module/Slave	Status	Circuit No./ Start
01 CPU-201 : ---			
-- UNDEFINED --			
PSA-12			
00 CPU-201 00 CPU201 [Driving]	01 CPU	Driving	----
	02 218IFD	Driving	Circuit No
	03 SVC32	Driving	Circuit No
	01 SGDV-****21 A	----	41 [H] 00 [H]
	02 SGDV-****21 A	----	42 [H] 00 [H]
	03 SGDV-****21 A	----	43 [H] 00 [H]
	04 SGDV-****21 A	----	04 [H] 00 [H]
	04 SVR32	Driving	Circuit No

⑥ Logical Axis Name

Give the name of the specified axis.

The name that is defined here is used when writing a motion program.

MVS [A1]1000 [B1]2000 [C1]3000 F1000;



5.3 Operation Priority Levels

A priority level is assigned to each operator used in an operation that uses motion language instructions. Use parentheses () to specify the priority level for an operation involving three or more items. The priority levels of operators are shown in the following table.

Operator	Priority Level				
	1	2	3	4	5
Parentheses	()				
NOT		!			
AND			&		
OR					
XOR				^	
Numeric operations				*	+
				/	-

Example Numeric Operation Examples

- Operation Example
 $MW00100 = 1 + 2;$
 With this operation, $1 + 2$ is calculated, and the result (3) is stored in MW00100.
- Example of Operation Involving Three or More Items
 $MW00100 = 1 + (2 * 3);$
 With this operation, 2×3 is calculated first, and 1 is added to the result (6). The final result (7) is then stored in MW00100.
 Therefore, $MW00100 = 7.$



Note

Precautions for Operations Involving Three or More Items

Consider the following expression:

$$MW00100 = 1 + 2 * 3;$$

In this operation, first 2×3 is calculated. Then, 1 is added to the result of 6 for a final result of 7. The final result of 7 is then stored in MW00100. Therefore, $MW00100 = 7.$

Example Logic Operation Examples

- Operation Example
 $MW00100 = 0001 \text{ hex } | 0002 \text{ hex};$
 Here, ORs of the bits in 0001 hex and 0002 hex are taken, and the results are stored in MW00100.
- Example of Operation Involving Three or More Items
 $MW00100 = (1111 \text{ hex } | 2222 \text{ hex}) \& 00FF \text{ hex};$
 Here, the OR of the bits in 1111 hex and 2222 hex is performed first, and then ANDs of the bits of the OR results and 00FF hex are taken. The final results are then stored in MW00100.
 Therefore, $MW00100 = 0033 \text{ hex}.$



Note

Precautions for Operations Involving Three or More Items

Consider the following expression:

$$MW00100 = 1111 \text{ hex } | 2222 \text{ hex } \& 00FF \text{ hex};$$

Here, ANDs of the bits in 2222 hex and 00FF hex are taken. Then, ORs of the AND results and the bits in 1111 hex are taken. The final results are then stored in MW00100.
 Therefore, $MW00100 = 1133 \text{ hex}.$



Note

Precautions for the Version 6 Compatible Compiler Version

The priority levels for operations performed under the version 6 compatible compiler version are shown in the following table.

Operator	Priority Level			
	1	2	3	4
Parentheses	()			
NOT		!		
AND			&	
OR				
XOR				^
Arithmetic operations				+ - * /

Consider the following expression:

$$MW00100 = 1 + 2 * 3;$$

In this operation, first $1 + 2$ is calculated. Then, the result of 3 is multiplied by 3 for a final result of 9. The final result of 9 is then stored in MW00100. Therefore, $MW00100 = 9$.

Instruction Types

■ S-type Instructions

S-type instructions, including operation instructions, are executed in one scan.

A program in which S-type instructions are continuously written is executed within one scan.

■ M-type Instructions

M-type instructions include axis movement instructions and other instructions that require multiple scans to execute.

One scan is required to change from an S-type instruction to an M-type instruction.

■ T-type Instructions

T-type instructions include timer instructions, which require multiple scans to execute.

■ F-type Instructions

Multiple scans are required to transfer commands from the CPU Unit/CPU Module to an Option Unit.

One scan is required to change from an S-type instruction to an F-type instruction.

Instruction Type Table

The following table gives the instruction types.

Category	Instruction	S Type	M Type	T Type	F Type
Axis Setting Instructions	ABS	○			
	INC	○			
	ACC		○		
	DCC		○		
	SCC		○		
	VEL		○		
	FMX	○			
	IFMX	○			
	IFP	○			
	FUT	○			
	IAC	○			
	IDC	○			
	IDH	○			
	IUT	○			
	+ or -			○	
ACCMODE	○				
Axis Movement Instructions	MOV		○		
	MVS		○		
	MCW		○		
	MCC		○		
	ZRN		○		
	DEN		○		
	SKP		○		
	MVT		○		
	EXM		○		
Control Instructions	POS	○			
	MVM	○			
	PLD	○			
	PFN		○		
	INP		○		
	PFP		○		
	PLN	○			
Program Control Instructions	IF ELSE IEND	○			
	WHILE WEND	○			
	WHILE WENDX			○	
	PFORK JOINTO PJOINT	○			
	SFORK JOINTO SJOINT	○			
	MSEE			○	
	SSEE	○			
	UFC		○		
	FUNC	○			
END	○				

Continued on next page.

5.4 Instruction Types and Execution Scans

Instruction Type Table

Continued from previous page.

Category	Instruction	S Type	M Type	T Type	F Type
Program Control Instructions	RET	○			
	TIM			○	
	TIM1MS			○	
	IOW			○	
	EOX			○	
	SNGD/SNGE	○			
Numeric Operations	=	○			
	+	○			
	-	○			
	++	○			
	--	○			
	*	○			
	/	○			
MOD	○				
Logic Operations		○			
	&	○			
	^	○			
	!	○			
Numeric Comparison Instructions	==	○			
	<>	○			
	>	○			
	<	○			
	>=	○			
	<=	○			
Data Manipulations	SFR	○			
	SFL	○			
	BLK	○			
	CLR	○			
	SETW	○			
	ASCII	○			
Basic Functions	SIN	○			
	COS	○			
	TAN	○			
	ASN	○			
	ACS	○			
	ATN	○			
	SQT	○			
	BIN	○			
	BCD	○			
	S{ }	○			
	R{ }	○			
	PON	○			
	NON	○			
	TON	○			
	TON1MS	○			
TOF	○				
TOF1MS	○				
Vision Instructions	VCAPI				○
	VCAPS				○
	VFIL				○
	VANA				○
	VRES				○

5.5

Programming with Variables

When programming with variables, the user declares and uses text strings that are called variables to perform operations.

This allows for programming with variables that are independent of any registers, which increases program reusability and extendability.

Variables can be used only within the program (within a single drawing) where they were declared.

The screenshot shows a software window titled 'Start MPM001' with a toolbar and a code editor. The code editor displays a ladder logic program with the following code:

```

LINE  BLOCK
1      VAR;
2      // TODO : Add the variable here.
3      0  WORD Count = 1;
4      LONG X_Position %ML00100;      "X_Position = ML00100"
5      LONG Y_Position %ML00102;      "Y_Position = ML00102"
6      END_VAR;
7      // TODO : Add the program here.
8      1  INC;
9      2  WHILE Count <= 10;
10     3  MOV [A1]X_Position [B1]Y_Position;
11     4  Count = Count + 1;
12     5  WEND;
13
14     6  END;

```

At the bottom of the window, there is a status bar with function keys F1 through F12.



Note

You can program with variables only with compiler version 7.00.
A compiling error will occur for the version 6 compatible compiler.

Declaring Variables

Give the variable that you want to declare inside a block that starts with VAR and ends with END_VAR.

You can declare up to 1,000 variables in one program.

After END_VAR, you can use the declared variable in the same ways as a register.

```

VAR;
  The variable you want to declare goes here.
END_VAR;

```

Variable Format

A variable consists of a data type, a text string containing alphanumeric characters or symbols that is between 1 and 255 characters in length, and a semicolon (;).

The size of all variables that are declared cannot be more than 16,384 words per program.

```
VAR;
  LONG Data ;
  Data type Variable name End of block
END_VAR;
```

The following table lists the valid data types for variables.

Data Type	Contents
BOOL	Bit
WORD/SINT	A signed integer that is one word in size.
LONG/DINT	A signed integer that is two words in size.
QUAD/LLONG/LINT	A signed integer that is four words in size.
FLOAT/REAL	A single-precision floating point number.
DOUBLE/LREAL	A double-precision floating point number.
ADDRESS	An address.
<i>Structure_name</i>	A structure.

Note: Arrays cannot be used for ADDRESS data.

The following table lists the characters that can be used in a variable name.

Type of Variable Name	Usable Characters
Letters	A to Z, a to z
Numbers	0 to 9
Symbols	_ (underbar)

Note: Variable names cannot start with a number.

Specifying a Default Value

The format to specify a default value for a variable is as follows:

```
VAR;
  Data type Variable name Default value
END_VAR;
```

Information You cannot specify a register as a default value.

Example Examples of Specifying Default Values

```
VAR;
  BOOL Complete = 1;
  LONG Vel      = 1000;
  LONG Position[3] = {1000, 2000, 3000};
END_VAR;
```

Associating Variables with Registers

You can specify that a declared variable should match the value of a specified register.

The format to specify a default value for a variable is as follows:

```
VAR;
  Data type  Variable name  % register  = Default value ;
END_VAR;
```

Information

1. You can also omit the default value from a variable declaration.
2. All registers except for # and C registers can be used in this way.

Example

Examples of Associating a Declared Variable with a Specified Register

```
VAR;
  BOOL Complete %OB00010;
  LONG Vel      %ML00200 = 1000 ;
  LONG Position[3] %ML00300 = {1000, 2000, 3000} ;
END_VAR;
```

Specifying Constants

Use the following format to specify constants.

```
VAR;
  CONST Data type  Variable name = Constant value ;
END_VAR;
```

Information

Constants cannot be associated with a register.

Example

Examples of Specifying Constants

```
VAR;
  CONST WORD MotionCMD_NOP = 0 ;
  CONST WORD MotionCMD_HOME = 9 ;
  CONST LONG MaxSpeed      = 6000 ;
END_VAR;
```

The strings in the following table cannot be used in variable names

Strings That Cannot Be Used in Variable Names

The strings in the following table cannot be used in variable names

Strings			
ABS	FEND	NON	SWITCH
ACC	FLOAT	OFF	TAN
ACCMODE	FMX	ON	TCN
ACOS	FOR	PFN	TCR
ACS	GOTO	PFORK	TCS
ARCTAN	I	PJOINT	TIM
ASIN	IAC	PLD	TOF
ASN	IDC	PLN	TON
ATAN	IEND	PON	TPS
ATN	IF	POS	TRUE
AUTO	IFP	R{	TYPDEF
BCD	INC	REGISTER	UFC
BIN	INP	RET	UNION
BLK	INT	RETURN	UNSIGNED
BREAK	IOW	S{	VCR
CASE	J	SCC	VCS
CHAR	JOINTO	SFL	VEL
CLR	KCC	SFORK	VOID
CONST	KCW	SFR	VOLATILE
CONTINUE	LCC	SHORT	WAX
COS	LCW	SIGNED	WCD
DCC	LOG	SIN	WCE
DEFAULT	LOG10	SIZEOF	WCT
DO	LONG	SJOINT	WDA
DOUBLE	MCC	SKP	WDB
ELSE	MCW	SNGD	WDC
END	MOD	SNGE	WDD
ENUM	MOV	SPH	WEND
EOX	MSEE	SPL	WHILE
EXM	MUFC	SQRT	WPM
EXP	MVM	SQT	WSA
EXTERN	MVS	STATIC	ZRN
FALSE	MVT	STRUCT	

Programming Examples

The following is a programming example that uses variables.

This programming example moves the X and Y axes 50 reference units each to draw a circle with a radius of 50 reference units 10 times.

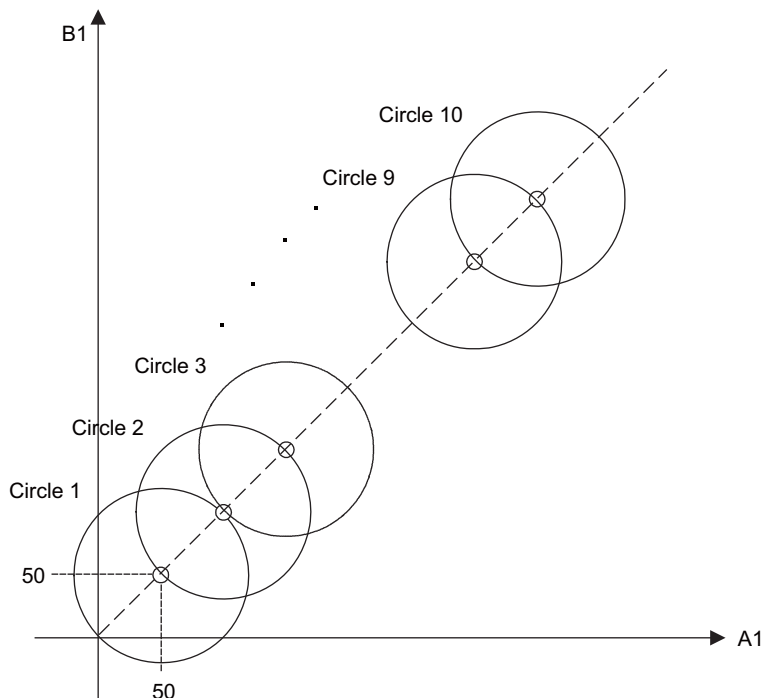
```

VAR;
    WORD Count;                                "Counter"
    CONST WORD CountNum = 10;                  "Number of loops"
    LONG X_radius %ML00100;                   "Radius of axis A1"
    LONG Y_radius %ML00102;                   "Radius of axis B1"
    LONG Speed = 8000;                         "Interpolation feed speed"
END_VAR;

ZRN [A1]0 [B1]0;
Count = 1;                                     "Preset counter"
INC;
PLN [A1][B1];
FMX T80000;
WHILE Count <= CountNum;                      "Loop for the specified number of times"
    MCW [A1]0 [B1]0 U X_radius V Y_radius F Speed; "Circular interpolation"
    MOV [A1]X_radius [B1]Y_radius;             "Positioning"
    Count = Count+1;                           "Increment counter"
WEND;

END;

```



Motion Language Instructions

6

This chapter describes the motion language instructions.

6.1	Axis Setting Instructions	6-4
	Absolute Mode (ABS)	6-7
	Incremental Mode (INC)	6-11
	Change Acceleration Time (ACC)	6-15
	Change Deceleration Time (DCC)	6-21
	Change S-curve Time Constant (SCC)	6-27
	Set Speed (VEL)	6-33
	Set Maximum Interpolation Feed Speed (FMX)	6-39
	Set Maximum Individual Axis Speeds for Interpolation (IFMX)	6-42
	Change Interpolation Feed Speed Unit (FUT)	6-45
	Set Interpolation Feed Speed Ratio (IFP)	6-47
	Change Interpolation Acceleration Time (IAC)	6-50
	Change Interpolation Deceleration Time (IDC)	6-52
	Change Interpolation Deceleration Time for Temporary Stop (IDH)	6-54
	Change Interpolation Acceleration/Deceleration Unit (IUT)	6-58
	Set Interpolation Feed Speed Axes (+ and -)	6-60
	Set Interpolation Acceleration/Deceleration Mode (ACCMODE)	6-63
6.2	Axis Movement Instructions	6-77
	Positioning (MOV)	6-81
	Linear Interpolation (MVS)	6-85
	Circular Interpolation with Specified Center Point (MCW and MCC)	6-90
	Circular Interpolation with Specified Radius (MCW and MCC)	6-95
	Helical Interpolation with Specified Center Point (MCW and MCC)	6-99

Helical Interpolation with Specified Radius (MCW and MCC)	6-102
Zero Point Return (ZRN)	6-104
Position after Distribution (DEN)	6-107
Linear Interpolation with Skip Function (SKP)	6-109
Set-time Positioning (MVT)	6-111
External Positioning (EXM)	6-113

6.3 Axis Control Instructions 6-115

Current Position Set (POS)	6-117
Move on Machine Coordinates (MVM)	6-119
Update Program Current Position (PLD)	6-120
In-position Check (PFN)	6-122
In-Position Range (INP)	6-124
Positioning Completed Check (PFP)	6-126
Coordinate Plane Setting (PLN)	6-128

6.4 Program Control Instructions 6-129

Branching Instructions (IF, ELSE, and IEND)	6-131
Repetition Instructions (WHILE, WEND)	6-134
Repetition with One Scan Wait (WHILE and WENDX)	6-137
Parallel Execution Instructions (PFORK, JOINTO, and PJOINT)	6-140
Selective Execution Instructions (SFORK, JOINTO, SJOINT)	6-143
Call Motion Subprogram (MSEE)	6-148
Call Sequence Subprogram (SSEE)	6-149
Call User Function from Motion Program (UFC)	6-150
Call User Function from Sequence Program (FUNC)	6-158
Program End (END)	6-159
Subprogram Return (RET)	6-160
Dwell Time (TIM)	6-161
Dwell Time (TIM1MS)	6-162
I/O Variable Wait (IOW)	6-163
One Scan Wait (EOX)	6-166
Disable Single-block Signal (SNGD) and Enable Single-block Signal (SNGE)	6-167

6.5 Numeric Operation Instructions 6-168

Substitute (=)	6-169
Add (+)	6-170
Subtract (-)	6-171
Extended Add (++)	6-172
Extended Subtract (--).	6-174
Multiply (*)	6-176
Divide (/)	6-177
Modulo (MOD)	6-178

6.6 Logic Operation Instructions 6-179

Inclusive OR ()	6-180
AND (&)	6-181

	Exclusive OR (^)	6-182
	NOT (!)	6-183
6.7	Numeric Comparison Instructions	6-184
	Numeric Comparison Instructions (==, <>, >, <, >=, <=) ..	6-186
6.8	Data Manipulations	6-189
	Bit Shift Right (SFR)	6-189
	Bit Shift Left (SFL)	6-191
	Move Block (BLK)	6-192
	Clear (CLR)	6-193
	Table Initialization (SETW)	6-194
	ASCII Conversion 1 (ASCII)	6-196
6.9	Basic Functions	6-198
	Sine (SIN)	6-200
	Cosine (COS)	6-201
	Tangent (TAN)	6-202
	Arc Sine (ASN)	6-203
	Arc Cosine (ACS)	6-204
	Arc Tangent (ATN)	6-205
	Square Root (SQT)	6-206
	BCD to Binary (BIN)	6-208
	Binary to BCD (BCD)	6-209
	Set Bit (S{ })	6-210
	Reset Bit (R{ })	6-211
	Rising-edge Pulse (PON)	6-212
	Falling-edge Pulse (NON)	6-214
	On-delay Timer: Measurement unit = 10 ms (TON)	6-216
	1-ms ON-Delay Timer (TON1MS)	6-217
	Off-delay Timer: Measurement unit = 10 ms (TOF)	6-218
	1-ms OFF-Delay Timer (TOF1MS)	6-219
6.10	Vision Instructions	6-220

6.1 Axis Setting Instructions

Axis setting instructions set the accelerations, decelerations, speeds, and other settings that are related to axis movement.

There are 16 axis setting instructions. You can use these instructions only in motion programs.

The following table lists the axis setting instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
ABS	Absolute Mode	ABS; or ABS MOV [<i>Logical_axis_name_1</i>] – [<i>Logical_axis_name_2</i>] – ;	Causes all subsequent coordinates to be treated as absolute values.	○	×
INC	Incremental Mode	INC; or INC MOV [<i>Logical_axis_name_1</i>] – [<i>Logical_axis_name_2</i>] – ;	Causes all subsequent coordinates to be treated as incremental values.	○	×
ACC	Change Acceleration Time	ACC [<i>Logical_axis_name_1</i>] <i>Acceleration_time</i> [<i>Logical_axis_name_2</i>] <i>Acceleration_time</i> [<i>Logical_axis_name_3</i>] <i>Acceleration_time</i> ... ;	Sets the acceleration times for positioning instructions. A maximum of 32 axes can be designated in one instruction block.	○	×
DCC	Change Deceleration Time	DCC [<i>Logical_axis_name_1</i>] <i>Deceleration_time</i> [<i>Logical_axis_name_2</i>] <i>Deceleration_time</i> [<i>Logical_axis_name_3</i>] <i>Deceleration_time</i> ... ;	Sets the deceleration times for positioning instructions. A maximum of 32 axes can be designated in one instruction block.	○	×
SCC	Change S-curve Time Constant	SCC [<i>Logical_axis_name_1</i>] <i>S-curve_time_constant</i> [<i>Logical_axis_name_2</i>] <i>S-curve_time_constant</i> ... ;	Sets the time constants for the moving average filters. A maximum of 32 axes can be designated in one instruction block. The filters are valid for both positioning instructions and interpolation instructions.	○	×
VEL	Set Speed	VEL [<i>Logical_axis_name_1</i>] <i>Feed_speed</i> [<i>Logical_axis_name_2</i>] <i>Feed_speed</i> [<i>Logical_axis_name_3</i>] <i>Feed_speed</i> ... ;	Sets the speeds for positioning instructions. A maximum of 32 axes can be designated in one instruction block.	○	×

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
				○	×
FMX	Set Maximum Interpolation Feed Speed	FMX $T_{\text{maximum_interpolation_feed_speed}}$;	Sets the maximum speed for interpolation instructions. The interpolation acceleration time is the time from a speed of zero to this speed. The interpolation deceleration time is the time from this speed to a speed of zero.	○	×
IFMX	Set Maximum Individual Axis Speeds for Interpolation	IFMX [<i>Logical_axis_name_1</i>] $_{\text{Maximum_individual_axis_speed_for_interpolation}}$ [<i>Logical_axis_name_2</i>] $_{\text{Maximum_individual_axis_speed_for_interpolation}}$	Sets the maximum speeds for the individual axes that are specified for interpolation instructions. You can set a different speed limit for each axis.	○	×
FUT	Change Interpolation Feed Speed Unit	FUT $U_{\text{interpolation_feed_speed_unit_number}}$;	Changes the speed unit for interpolation instructions.	○	×
IFP	Set Interpolation Feed Speed Ratio	IFP $P_{\text{interpolation_feeding_speed_ratio}}$;	Sets the speed for interpolation instructions. Specify the speed as a percentage of the maximum speed.	○	×
IAC	Change Interpolation Acceleration Time	IAC $T_{\text{interpolation_acceleration_time}}$;	Sets the acceleration time for interpolation instructions. Specify the time required to reach the maximum speed from a speed of 0.	○	×
IDC	Change Interpolation Deceleration Time	IDC $T_{\text{interpolation_deceleration_time}}$;	Sets the deceleration time for interpolation instructions. Specify the time required to decelerate to a speed of 0 from the maximum speed.	○	×
IDH	Change Interpolation Deceleration Time for Temporary Stop	IDH $T_{\text{interpolation_deceleration_time_for_temporary_stop}}$;	Sets the deceleration time for interpolation instructions when the axis is temporarily stopped. Specify the time required to decelerate to a speed of 0 from the maximum speed.	○	×
IUT	Change Interpolation Acceleration/Deceleration Unit	IUT $U_{\text{interpolation_acceleration/deceleration_unit_number}}$;	Changes the acceleration/deceleration unit for interpolation instructions (MVS, SKP, MCW, and MCC).	○	×

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
(+ or -)	Set Interpolation Feed Speed Axes	MVS [+ <i>Logical_axis_name_1</i>] <i>Reference_position</i> [+ <i>Logical_axis_name_2</i>] <i>Reference_position</i> [- <i>Logical_axis_name_3</i>] <i>Reference_position ...</i> ;	Specifies the axes to use as component axes for the interpolation feed speed. If “+” or nothing is given before the logical axis name, the axis is used as one of the component axes for the interpolation feed speed. If “-” is given before the logical axis name, the axis operates at a speed that is synchronized with the interpolation feed speed.	○	×
ACCMODE	Set Interpolation Acceleration/Deceleration Mode	ACCMODE <i>Mmode_number</i> ;	Sets the acceleration/deceleration mode for interpolation instructions. This allows you to specify processing multiple interpolation instructions in succession.	○	×

Absolute Mode (ABS)

The ABS instruction causes the coordinate words that control axis movement to be treated as final target positions.

After the ABS instruction is executed, it remains in effect until the INC instruction is executed. Absolute Mode is the default mode when program operation is started.

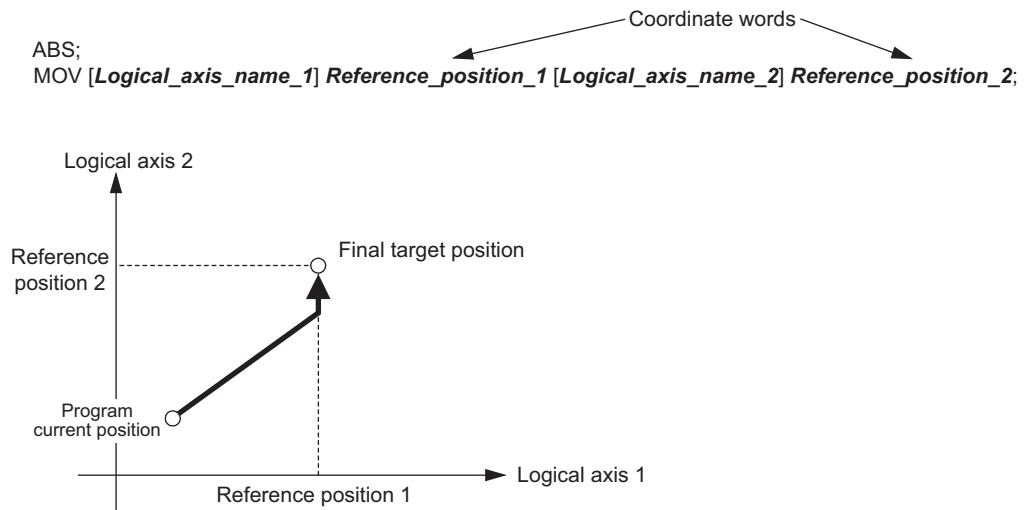


Fig. 6.1 Movement Mode for ABS Instruction

In this manual, a coordinate word that follows a logical axis name in an axis movement instruction is called the reference position or the position reference value.

⚠ CAUTION

- The same coordinate word will create a completely different travel operation in Absolute Mode and in Incremental Mode. Make sure that the ABS and INC instructions are used correctly before you start operation.
There is a risk of injury or device damage.



Terms

Program Current Position

This is the position in the work coordinate system when an axis is moved by an axis movement instruction.

Format

The format of the ABS instruction is as follows:

- When Specified Independently
ABS;
- When Specified in the Same Block as an Axis Movement Instruction
ABS MOV [*Logical_axis_name_1*] - [*Logical_axis_name_2*] - ;

Programming Example

A programming example that uses the ABS instruction is given below.

```
ABS; "Absolute Mode
MOV [A1]10000 [B1]40000;"Positioning
MOV [A1]50000 [B1]20000;"Positioning
END;
```

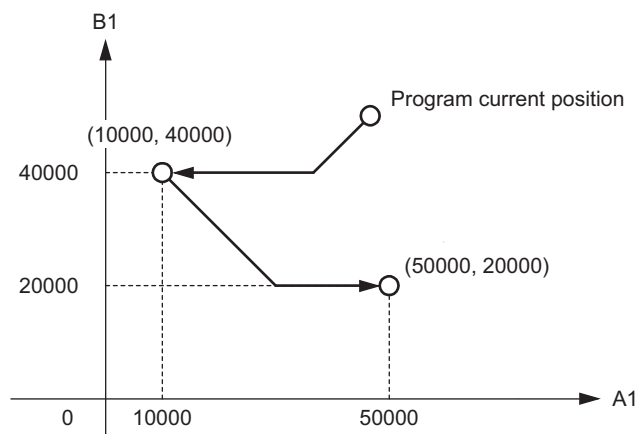


Fig. 6.2 Programming Example for the ABS Instruction

Additional Information on the ABS Instruction

◆ Related Motion Parameters

The ABS instruction is not related to any setting parameters.

The movement mode (Absolute Mode or Incremental Mode) for axis movement instructions is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify these modes.



Note

The movement mode (Absolute Mode or Incremental Mode) for axis movement instructions is not the same as the position reference type that is specified in bit 5 of the OW□□□09 setting parameter.


◆ Finite-length Axes and Infinite-length Axes

The position reference value of a coordinate word for a finite-length axis must be handled differently from one for an infinite-length axis.


The following table tells how to specify position reference values for finite-length and infinite-length axes.

Axis Type	Movement Mode for Axis Movement Instruction	Specification Method for Position Reference Values
Finite-length axis	Absolute Mode	Specify the final target position for the position reference value.
	Incremental Mode	Specify the relative travel distance for the position reference value.
Infinite-length axis	Absolute Mode	Specify the final target position to a value between 0 and POSMAX for the position reference value. The sign of the position reference value indicates the travel direction. You specify a positive direction with a positive value, and a negative direction with a negative value.
	Incremental Mode	Specify the relative travel distance for the position reference value.

Information

- Use bit 0 (Axis Selection) of fixed parameter No. 1 (Function Selection Flags 1) to select either a finite-length axis or an infinite-length axis.
Use finite-length axes or infinite-length axes as required according to the machine configuration. Refer to the following manual for information on setting motion parameters for the machine you are using.
 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)
- Use fixed parameter No. 10 (Infinite-length Axis Reset Position (POSMAX)) to set POSMAX.

The operation of a finite-length axis and an infinite-length axis in Absolute Mode is described below. Refer to the following section for details on operation in Incremental Mode.

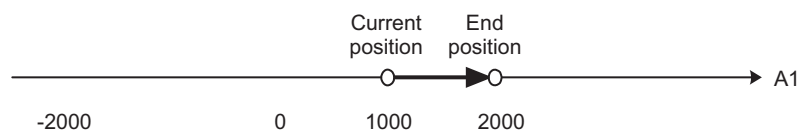
 *Incremental Mode (INC)* (page 6-11)

■ Using Absolute Mode for a Finite-length Axis

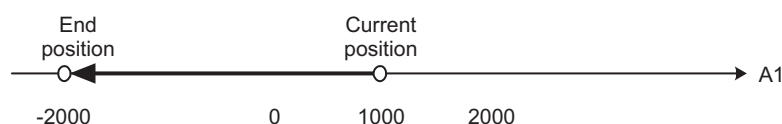
Specify the final target position for the position reference value.

For example, the following operation occurs from a current position of 1,000 when the final target position is set to 2,000 or -2,000.

```
ABS;
MOV [A1]2000;
```



```
ABS;
MOV [A1]-2000;
```



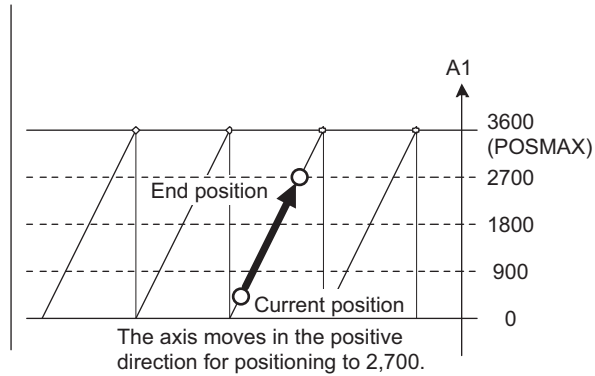
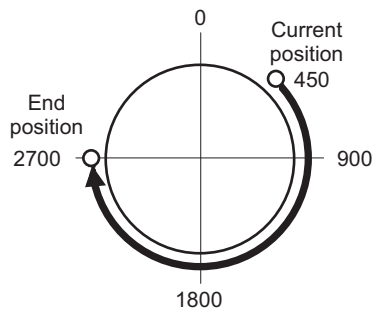
■ Using Absolute Mode for an Infinite-length Axis

Specify the final target position to a value between 0 and POSMAX for the position reference value.

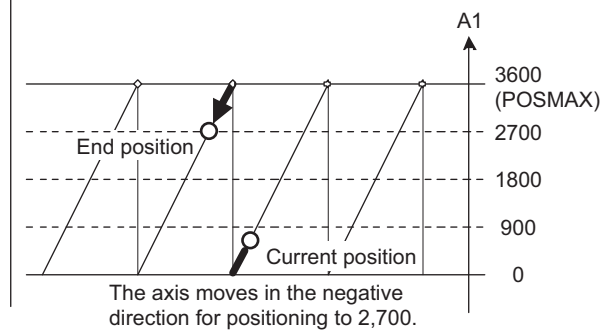
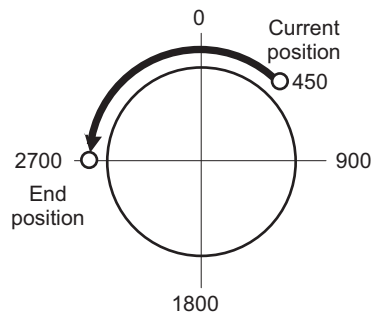
The sign of the position reference value indicates the travel direction. You specify a positive direction with a positive value, and a negative direction with a negative value.

For example, the following operation occurs from a current position of 450 for an infinite-length axis with a POSMAX of 3,600 when the final target position is set at 2,700 or -2,700.

ABS;
MOV [A1]2700;



ABS;
MOV [A1]-2700;



Note

1. When a position reference value of +0 is specified for an infinite-length axis in Absolute Mode, the axis moves in the negative direction.
Specify the POSMAX value to move the axis in the positive direction.
2. If the final target position (the absolute value of the position reference value) exceeds the POSMAX value for an infinite-length axis in Absolute Mode, an alarm will occur for the motion program.

Incremental Mode (INC)

The INC instruction causes the coordinate words that control axis movement to be treated as relative travel distances.

After the INC instruction is executed, it remains in effect until the ABS instruction is executed. Absolute Mode is the default mode when program operation is started.

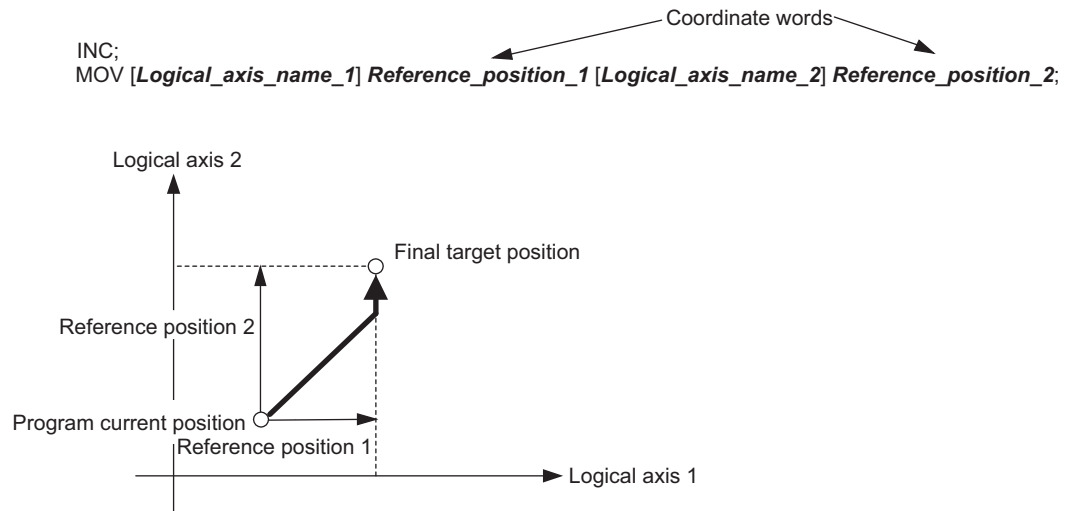


Fig. 6.3 Movement Mode for INC Instruction

In this manual, a coordinate word that follows a logical axis name in an axis movement instruction is called the reference position or the position reference value.

⚠ CAUTION

- The same coordinate word will create a completely different travel operation in Absolute Mode and in Incremental Mode. Make sure that the ABS and INC instructions are used correctly before you start operation.
There is a risk of injury or device damage.



Terms

Program Current Position

This is the position in the work coordinate system when an axis is moved by an axis movement instruction.

Format

The format of the INC instruction is as follows:

- When Specified Independently
INC;
- When Specified in the Same Block as an Axis Movement Instruction
INC MOV [*Logical_axis_name_1*] - [*Logical_axis_name_2*] - ;

Programming Example

A programming example that uses the INC instruction is given below.

```
INC; "Incremental Mode
MOV [A1]20000 [B1]30000;"Positioning
MOV [A1]20000 [B1]10000;"Positioning
END;
```

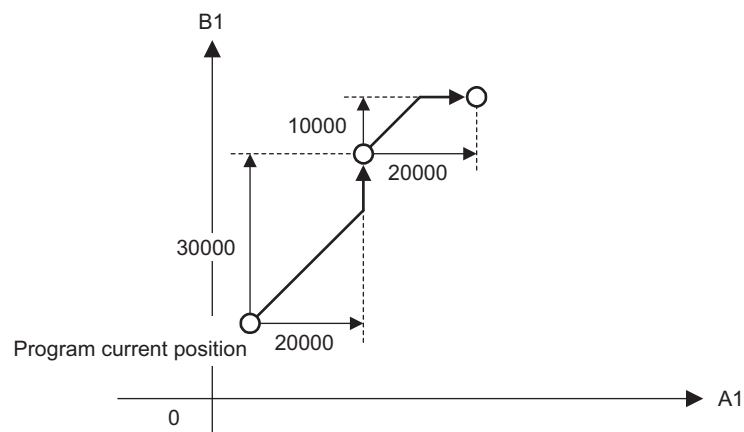


Fig. 6.4 Programming Example for the INC Instruction

Additional Information on the INC Instruction

◆ Related Motion Parameters

The INC instruction is not related to any setting parameters.

The movement mode (Absolute Mode or Incremental Mode) for axis movement instructions is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify these modes.


◆ Finite-length Axes and Infinite-length Axes

The position reference value of a coordinate word for a finite-length axis must be handled differently from one for an infinite-length axis.


The following table tells how to specify position reference values for finite-length and infinite-length axes.

Axis Type	Movement Mode for Axis Movement Instruction	Specification Method for Position Reference Values
Finite-length axis	Absolute Mode	Specify the final target position for the position reference value.
	Incremental Mode	Specify the relative travel distance for the position reference value.
Infinite-length axis	Absolute Mode	Specify the final target position to a value between 0 and POSMAX for the position reference value. The sign of the position reference value indicates the travel direction. You specify a positive direction with a positive value, and a negative direction with a negative value.
	Incremental Mode	Specify the relative travel distance for the position reference value.

Information

- Use bit 0 (Axis Selection) of fixed parameter No. 1 (Function Selection Flags 1) to select either a finite-length axis or an infinite-length axis.
Use finite-length axes or infinite-length axes as required according to the machine configuration. Refer to the following manual for information on setting motion parameters for the machine you are using.
 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)
- Use fixed parameter No. 10 (Infinite-length Axis Reset Position (POSMAX)) to set POSMAX.

The operation of a finite-length axis and an infinite-length axis in Incremental Mode is described below. Refer to the following section for details on operation in Absolute Mode.

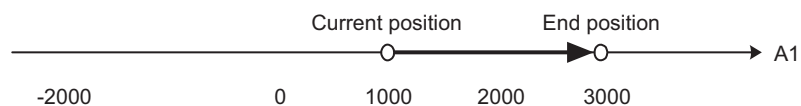
 *Absolute Mode (ABS)* (page 6-7)

■ Using Incremental Mode for a Finite-length Axis

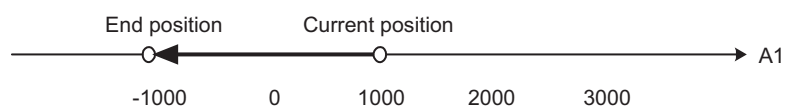
Specify the relative travel distance for the position reference value.

For example, the following operation occurs from a current position of 1,000 when the final target position is set to 2,000 or -2,000.

```
INC;
MOV [A1]2000;
```



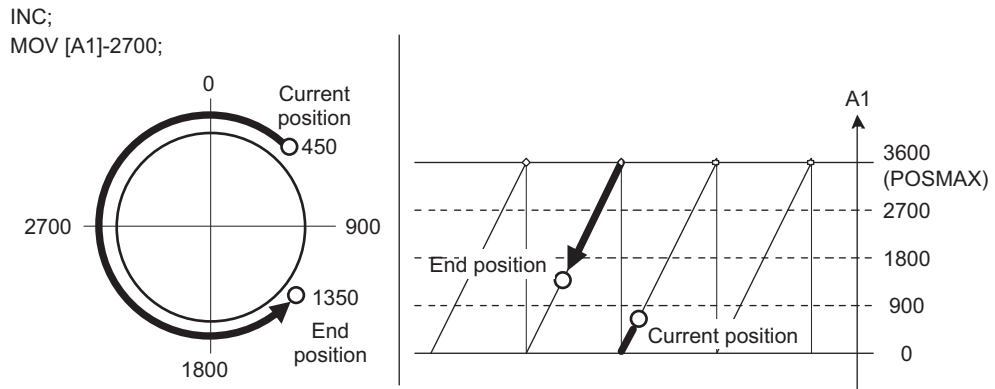
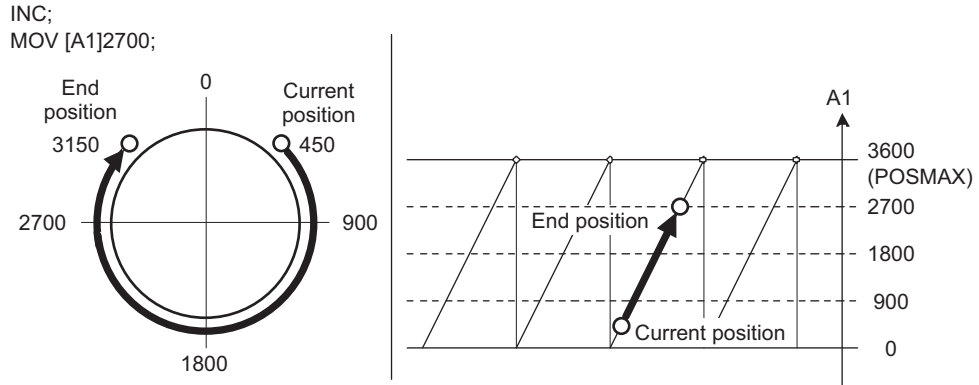
```
INC;
MOV [A1]-2000;
```



■ Using Incremental Mode for an Infinite-length Axis

Specify the relative travel distance for the position reference value.

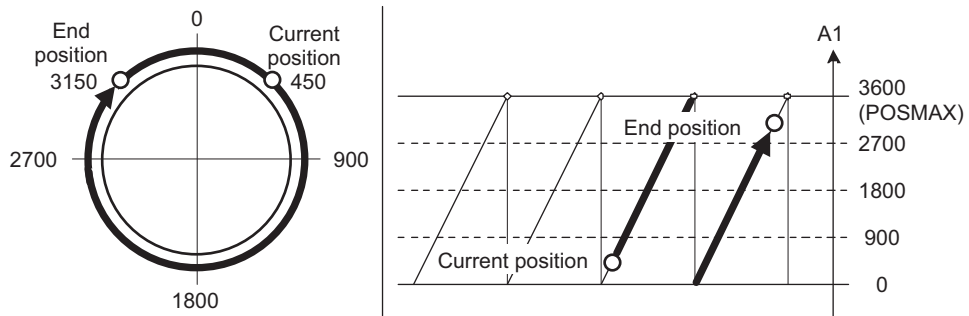
For example, the following operation occurs from a current position of 450 for an infinite-length axis with a POSMAX of 3,600 when the final target position is set at 2,700 or -2,700.



Information

If the absolute value of the position reference value (coordinate word) exceeds the POSMAX value, the position reference value (coordinate word) is used for the relative movement amount to move the axis in Incremental Mode.

INC;
MOV [A1]6300; "|6300|>3600(POSMAX)



Change Acceleration Time (ACC)

The ACC instruction changes the acceleration times or acceleration rates of the specified axes for all of the following axis movement instructions.

- MOV (Positioning)
- MVT (Set-time Positioning)
- EXM (External Positioning)

The values can be changed for up to 32 axes with one instruction. The acceleration time for any unspecified axis is not changed.

The acceleration times that are set by the ACC instruction remain in effect until they are changed by another ACC instruction.

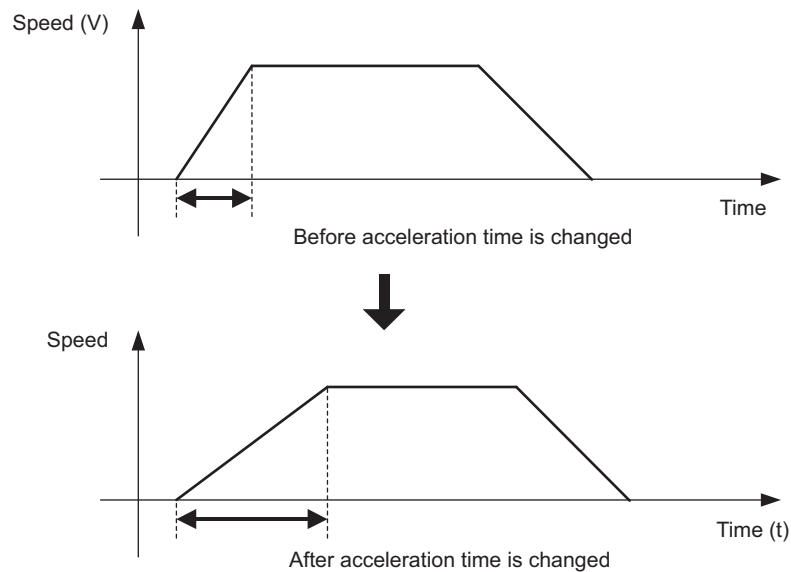


Fig. 6.5 Change Acceleration Time

Information

1. The ACC instruction changes the acceleration time for the MOV, EXM, and MVT positioning instructions. Use the IAC instruction to set the acceleration time for the MVS, MCW, MCC, and SKP interpolation instructions.
2. The ACC, DCC, and SCC instructions are supported by all Motion Control Function Modules.

Format

The format of the ACC instruction is as follows:

ACC [Logical_axis_name_1] Acceleration_time [Logical_axis_name_2] Acceleration_time [Logical_axis_name_3] Acceleration_time. . . ;

Item	Unit	Applicable Data
Acceleration time or acceleration rate	ms or reference units/s ²	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Note: The unit is set in bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of the OW□□□03 setting parameter.

Settings for the ACC Instruction

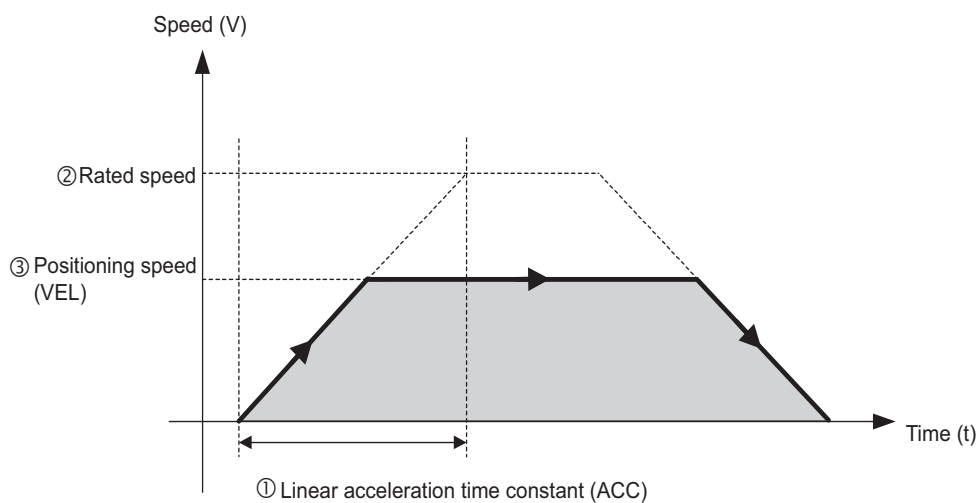
This section describes the settings for the ACC instruction.

Either acceleration times (ms) or acceleration rates (reference units/s²) can be selected for the setting unit of the ACC instruction.

The unit to use is set in bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of the OW□□□03 setting parameter.

Parameter Name	Acceleration/Deceleration Rate Unit
Function Settings 1	0: Reference units/s ²
Acceleration/Deceleration Rate Unit Selection	1: ms (default)

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 1 (ms)



① Acceleration Time

The settings in the ACC instruction are used as the acceleration times (the time required to reach the rated speed from a speed of 0). The valid range is 1 to 32,767 ms.

② Rated Speed

The rated speed for each axis is set in fixed parameter No. 34 (Rated Motor Speed).

③ Positioning Speed

This speed is used by the MOV, MVT, and EXM positioning instructions.

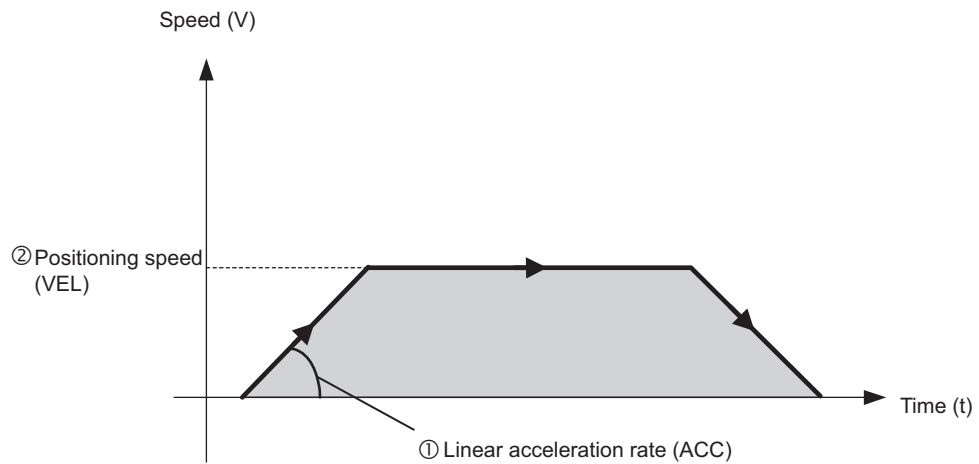
The positioning speed for each axis is set with the VEL instruction.

Information

For the MVT instruction, the positioning speed is not the reference value of the VEL instruction.

The MVT instruction changes the positioning speed according to the set positioning time and the amount of movement.

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 0 (Reference Units/s²)



① Linear Acceleration Rate

The settings in the ACC instruction are used as the linear acceleration rates.
The valid range is 1 to $2^{31} - 1$ (reference units/s²).

② Positioning Speed

This speed is used by the MOV, MVT, and EXM positioning instructions.
The positioning speed for each axis is set with the VEL instruction.

Information

For the MVT instruction, the positioning speed is not the reference value of the VEL instruction.

The MVT instruction changes the positioning speed according to the set positioning time and the amount of movement.

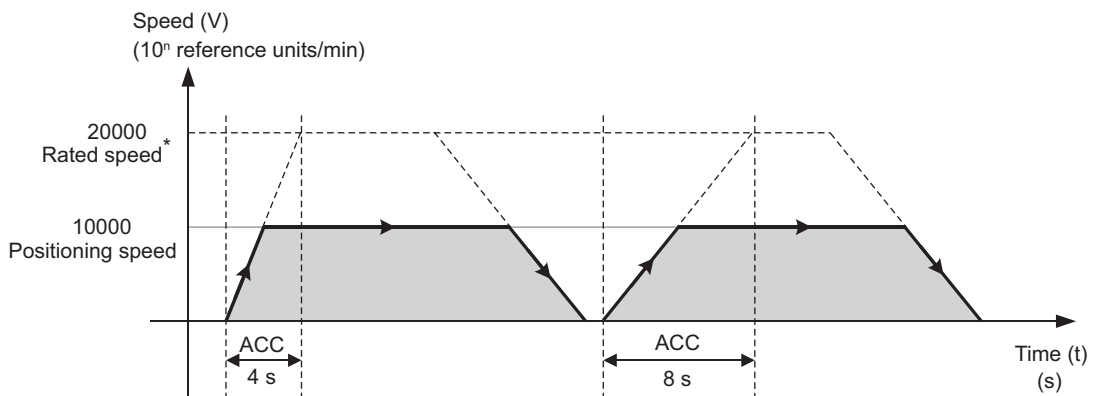
Programming Examples

Programming examples that use the ACC instruction are given below.

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 1 (ms)

The following example executes the MOV instruction to accelerate axis A1 from 0 to the rated speed in 4 seconds, and then executes the MOV instruction to accelerate axis A1 in 8 seconds.

INC;	"Incremental Mode
VEL [A1]10000;	"Set feed speed (10^n reference units/min).
DCC [A1]8000;	"Change deceleration time (ms).
ACC [A1]4000;	"Change acceleration time (ms).
MOV [A1]5000000;	"Positioning
DL00000 = 8000;	"Acceleration time (ms)
ACC [A1]DL00000;	"Change acceleration time (ms).
MOV [A1]5000000;	"Positioning
END;	



* The unit used for the rated speed (min^{-1}) must be converted to the same unit as the unit that is used for positioning speed (10^n reference units/min).

Fig. 6.6 Programming Example 1 for the ACC Instruction

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 0 (Reference Units/s²)

The following example executes the MOV instruction to accelerate axis A1 at a rate of 60.000 (mm/s²) and then executes the MOV instruction to accelerate axis A1 at a rate of 100.000 (mm/s²). In this example, 1 reference unit is 0.001 mm.

```

INC;                                "Incremental Mode
VEL [A1]18000;                       "Set feed speed (10n reference units/min).
DCC [A1]100000;                      "Change deceleration time (reference units/s2).
ACC [A1]60000;                       "Set acceleration rate (reference units/s2).
MOV [A1]5000000;                     "Positioning
DL00000 = 100000;                   "Acceleration rate (reference units/s2)
ACC [A1]DL00000;                    "Set acceleration rate (reference units/s2).
MOV [A1]5000000;                     "Positioning
END;

```

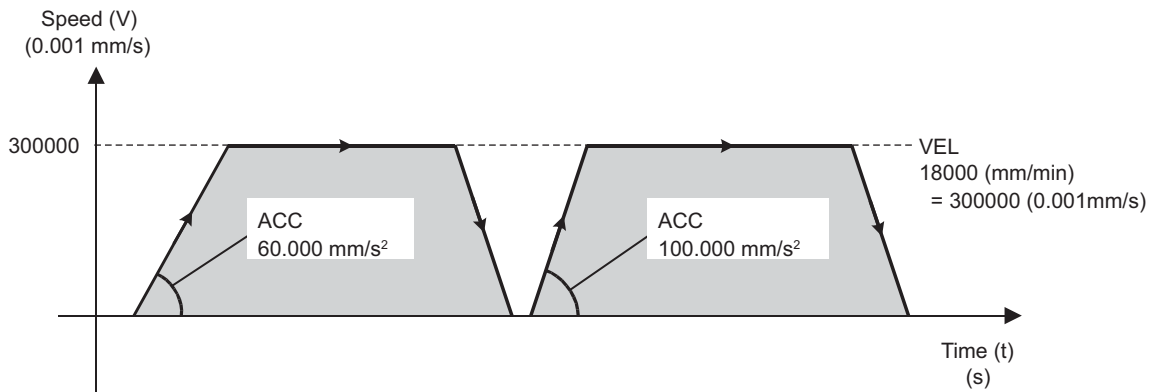


Fig. 6.7 Programming Example 2 for the ACC Instruction

Additional Information on the ACC Instruction

◆ Related Motion Parameters

The ACC instruction changes the acceleration times in the setting parameters.

Parameter Name	Register Address	Description
Linear Acceleration Rate/ Acceleration Time Constant	OL□□□36	Sets the linear acceleration rate or linear acceleration time constant.


The acceleration times can be changed by directly changing the settings of the OL□□□36 (Linear Acceleration Rate/Acceleration Time Constant) setting parameters instead of by using the ACC instruction.

Refer to the following table for details on how to directly change the acceleration time settings.

Motion Control Function Modules	Specification	Setting Procedure
SVR, SVR32, SVA-01, PO-01	The axes move according to the acceleration times that are set in the OL□□□36 (Linear Acceleration Rate/Acceleration Time Constant) setting parameters.	Set the acceleration times in the OL□□□36 (Linear Acceleration Rate/Acceleration Time Constant) setting parameters.
SVC, SVC32, SVC-01, SVB-01	The axis moves at the acceleration rate that is set in the SERVOPACK parameters.	Set the acceleration times in the OL□□□36 (Linear Acceleration Rate/Acceleration Time Constant) setting parameters. Then, set the OW□□□08 (Motion Commands) setting parameters to 10 (Change Acceleration Time) to write the new acceleration times to the SERVOPACK.

Note: The SVC, SVC32, SVC-01, and SVB-01 Function Modules can automatically set the acceleration rates in the SERVOPACK parameters to the values of the OL□□□36 (Linear Acceleration Rate/Acceleration Time Constant) setting parameters. If this automatic writing function is enabled, you do not need to set the OW□□□08 (Motion Commands) setting parameters to 10 (Change Acceleration Time).

Refer to the following manual for details on how to use the automatic writing function.

 *MP3000 Series Motion Control User's Manual* (Manual No.: S1EP C880725 11)

◆ Acceleration Times and Deceleration Times

With the following combinations of the Motion Control Function Module and SERVOPACK models, the acceleration time and deceleration time for an axis cannot be set separately. If you set the acceleration time, the deceleration time will be automatically set. The acceleration time and deceleration time for an axis can be set separately using the ACC and DCC instructions for any SERVOPACK model other than the SGD-N or SGDB-N.

Motion Control Function Module	SERVOPACK	Remarks
SVB-01	SGD-N	• With the SVB-01 Function Module, an axis moves at the acceleration/ deceleration rate that is set in the SERVOPACK parameters.
	SGDB-N	• The SGD-N and SGDB-N SERVOPACKs use the same parameter to set both the acceleration time and deceleration time.

Change Deceleration Time (DCC)

The DCC instruction changes the deceleration times or deceleration rates of the specified axes for all of the following axis movement instructions.

- MOV (Positioning)
- MVT (Set-time Positioning)
- EXM (External Positioning)

The values can be changed for up to 32 axes with one instruction. The deceleration time for any unspecified axis is not changed.

The deceleration times that are set by the DCC instruction remain in effect until they are changed by another DCC instruction.

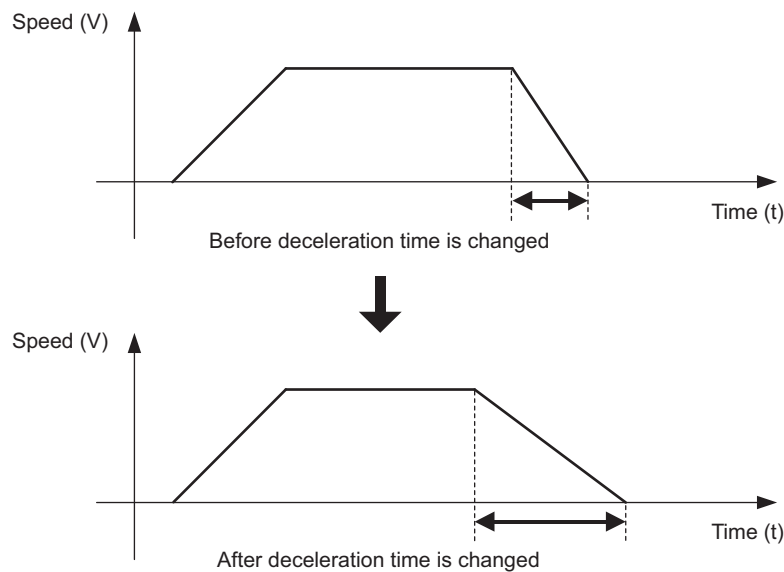


Fig. 6.8 Change Deceleration Time

Information

1. The DCC instruction changes the deceleration time for the MOV, EXM, and MVT positioning instructions. Use the IDC instruction to set the deceleration time for the MVS, MCW, MCC, and SKP interpolation instructions.
2. The ACC, DCC, and SCC instructions are supported by all Motion Control Function Modules.

Format

The format of the DCC instruction is as follows:

DCC [*Logical_axis_name_1*] *Deceleration_time* [*Logical_axis_name_2*] *Deceleration_time* [*Logical_axis_name_3*] *Deceleration_time* . . . ;

Item	Unit	Applicable Data
Deceleration time or deceleration rate	ms or reference units/s ²	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Note: The unit is set in bits 4 to 7 of the OW□□□03 setting parameter.

Settings for the DCC Instruction

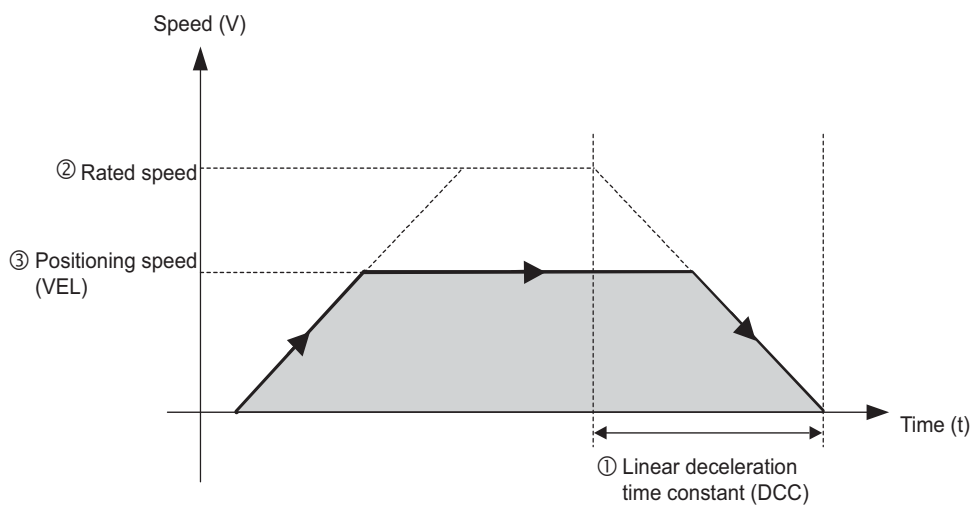
This section describes the settings for the DCC instruction.

Either deceleration times (ms) or deceleration rates (reference units/s²) can be selected for the setting unit of the DCC instruction.

The unit to use is set in bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of the OW□□□03 setting parameter.

Parameter Name	Acceleration/Deceleration Unit
Function Settings 1	0: Reference units/s ²
Acceleration/Deceleration Rate Unit Selection	1: ms (default)

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 1 (ms)



① Linear Deceleration Time Constant

The settings in the DCC instruction are used as linear deceleration times (the time required to reach a speed of 0 from the rated speed).

The valid range is 1 to 32,767 ms.

② Rated Speed

The rated speed for each axis is set in fixed parameter No. 34 (Rated Motor Speed).

③ Positioning Speed

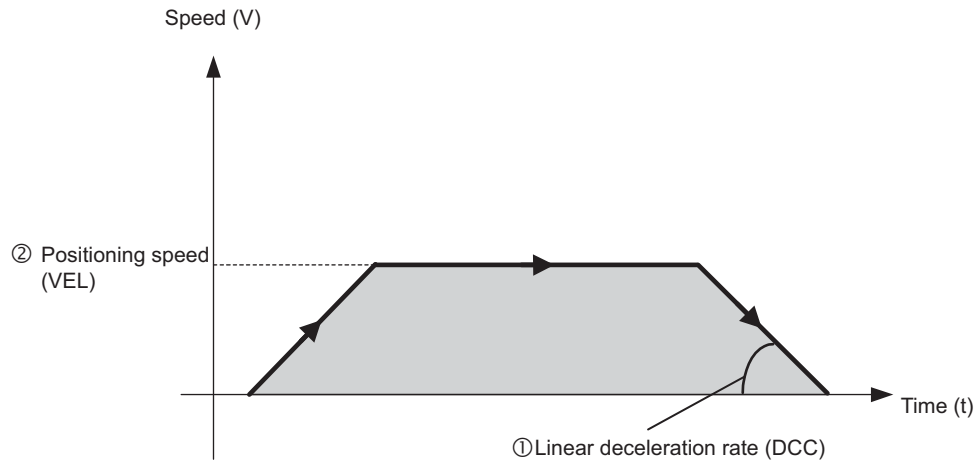
This speed is used by the MOV, MVT, and EXM positioning instructions.

The positioning speed for each axis is set with the VEL instruction.

Information For the MVT instruction, the positioning speed is not the reference value of the VEL instruction.

The MVT instruction changes the positioning speed according to the set positioning time and the amount of movement.

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□3 Setting Parameter Are Set to 0 (Reference Units/s²)



① Linear Deceleration Rate

The settings in the DCC instruction are used as linear deceleration rates.
The valid range is 1 to $2^{31} - 1$ (reference units/s²).

② Positioning Speed

This speed is used by the MOV, MVT, and EXM positioning instructions.
The positioning speed for each axis is set with the VEL instruction.

Information

For the MVT instruction, the positioning speed is not the reference value of the VEL instruction.

The MVT instruction changes the positioning speed according to the set positioning time and the amount of movement.

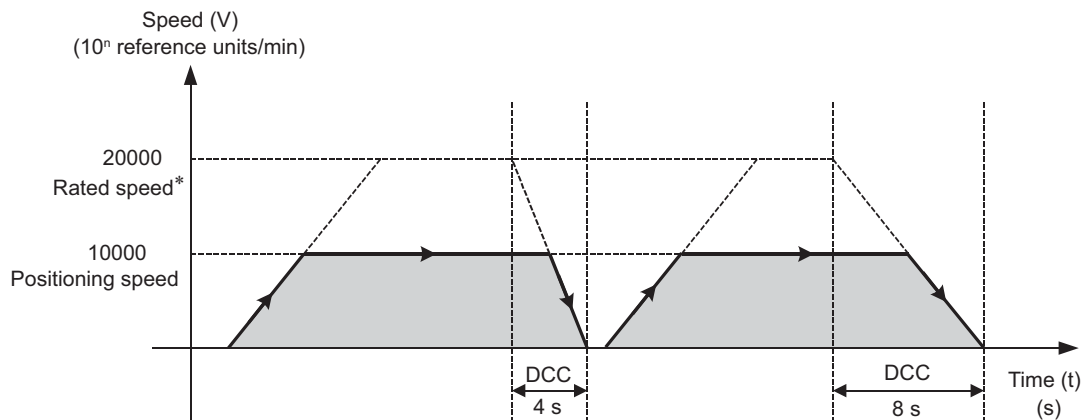
Programming Examples

Programming examples that use the DCC instruction are given below.

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□03 Setting Parameter Are Set to 1 (ms)

The following example executes the MOV instruction to decelerate axis A1 from the rated speed to a speed of 0 in 4 seconds, and then executes the MOV instruction to decelerate axis A1 from the rated speed to a speed of 0 in 8 seconds.

INC;	"Incremental Mode
VEL [A1]10000;	"Set feed speed (10^n reference units/min).
ACC [A1]8000;	"Change acceleration time (ms).
DCC [A1]4000;	"Change deceleration time (ms).
MOV [A1]5000000;	"Positioning
DL00000 = 8000;	"Deceleration time (ms)
DCC [A1]DL00000;	"Change deceleration time (ms).
MOV [A1]5000000;	"Positioning
END;	



* The unit used for the rated speed (min^{-1}) must be converted to the same unit as the unit that is used for positioning speed (10^n reference units/min).

Fig. 6.9 Programming Example 1 for the DCC Instruction

- When Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) in the OW□□□3 Setting Parameter Are Set to 0 (Reference Units/s²)

The following example executes the MOV instruction to decelerate axis A1 at a rate of 60.000 (mm/s²), and then executes the MOV instruction to decelerate axis A1 at a rate of 100.000 (mm/s²). In this example, 1 reference unit is 0.001 mm.

```

INC;                                "Incremental Mode
VEL [A1]18000;                       "Set feed speed (10n reference units/min).
ACC [A1]100000;                      "Set acceleration rate (reference units/s2).
DCC [A1]60000;                       "Set deceleration rate (reference units/s2).
MOV [A1]5000000;                     "Positioning
DL00000 = 100000;                   "Deceleration rate (reference units/s2)
DCC [A1] DL00000;                   "Set deceleration rate (reference units/s2).
MOV [A1]5000000;                     "Positioning
END;

```

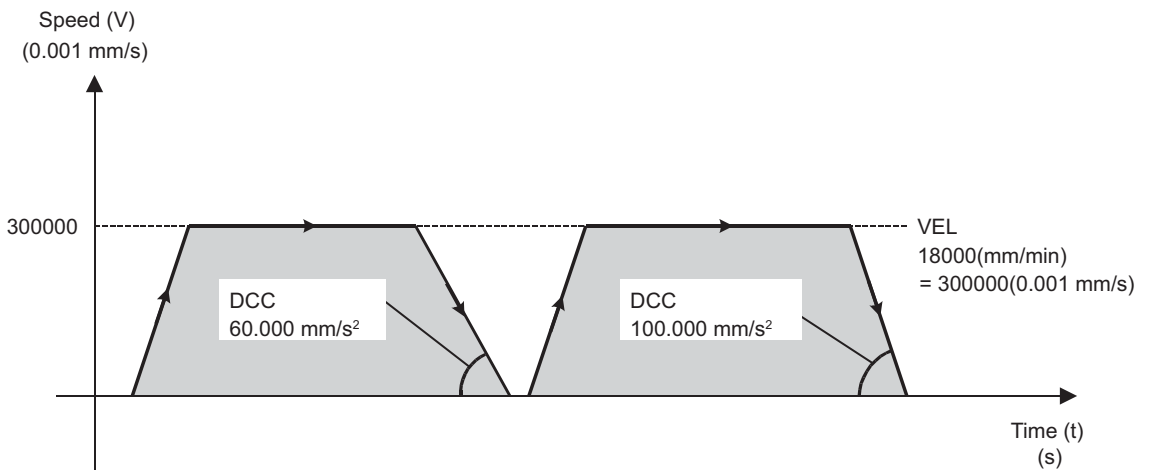


Fig. 6.10 Programming Example 2 for the DCC Instruction

Additional Information on the DCC Instruction

◆ Related Motion Parameters

The DCC instruction changes the deceleration times in the setting parameters.

Parameter Name	Register Address	Description
Linear Deceleration Rate/ Deceleration Time Constant	OL□□□38	Sets the linear deceleration rate or linear deceleration time constant.

The deceleration times can be changed by directly changing the settings of the OL□□□38 (Linear Deceleration Rate/Deceleration Time Constant) setting parameters instead of by using the DCC instruction.

Refer to the following table for details on how to directly change the deceleration time settings.

Motion Control Function Modules	Specification	Setting Procedure
SVR, SVR32, SVA-01, PO-01	The axes move according to the deceleration times that are set in the OL□□□38 (Linear Deceleration Rate/Deceleration Time Constant) setting parameters.	Set the deceleration times in the OL□□□38 (Linear Deceleration Rate/Deceleration Time Constant) setting parameters.
SVC, SVC32, SVC-01, SVB-01	The axis moves at the deceleration rate that is set in the SERVOPACK parameters.	Set the deceleration times in the OL□□□38 (Linear Deceleration Rate/Deceleration Time Constant) setting parameters. Then, set the OW□□□08 (Motion Commands) setting parameters to 11 (Change Deceleration Time) to write the new deceleration times to the SERVOPACK.

Note: The SVC, SVC32, SVC-01, and SVB-01 Function Modules can automatically set the deceleration rates in the SERVOPACK parameters to the values of the OL□□□38 (Linear Deceleration Rate/Deceleration Time Constant) setting parameters. If this automatic writing function is enabled, you do not need to set the OW□□□08 (Motion Commands) setting parameters to 11 (Change Deceleration Time).

Refer to the following manual for details on how to use the automatic writing function.

📖 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)

◆ Acceleration Times and Deceleration Times

With the following combinations of the Motion Control Function Module and SERVOPACK models, the acceleration time and deceleration time for an axis cannot be set separately. If you set the acceleration time, the deceleration time will be automatically set. The acceleration time and deceleration time for an axis can be set separately using the ACC and DCC instructions for any SERVOPACK model other than the SGD-N or SGDB-N.

Motion Control Function Module	SERVOPACK	Remarks
SVB-01	SGD-N	• With the SVB-01 Function Module, an axis moves at the acceleration/ deceleration rate that is set in the SERVOPACK parameters.
	SGDB-N	• The SGD-N and SGDB-N SERVOPACKs use the same parameter to set both the acceleration time and deceleration time.

Change S-curve Time Constant (SCC)

The SCC instruction changes the S-curve time constants for axis movement instructions.

The S-curve time constant parameter for the S-curve acceleration/deceleration function suppresses mechanical vibration during acceleration and deceleration.

The values can be changed for up to 32 axes with one instruction. The S-curve time constant for any unspecified axis is not changed.

The S-curve time constants that are set by the SCC instruction remain in effect until they are changed by another SCC instruction.

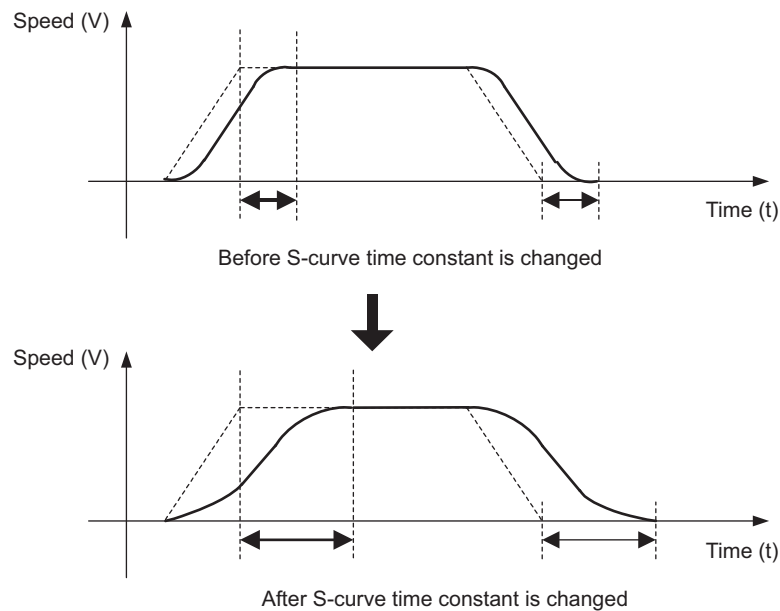


Fig. 6.11 Change S-curve Time Constant

Format

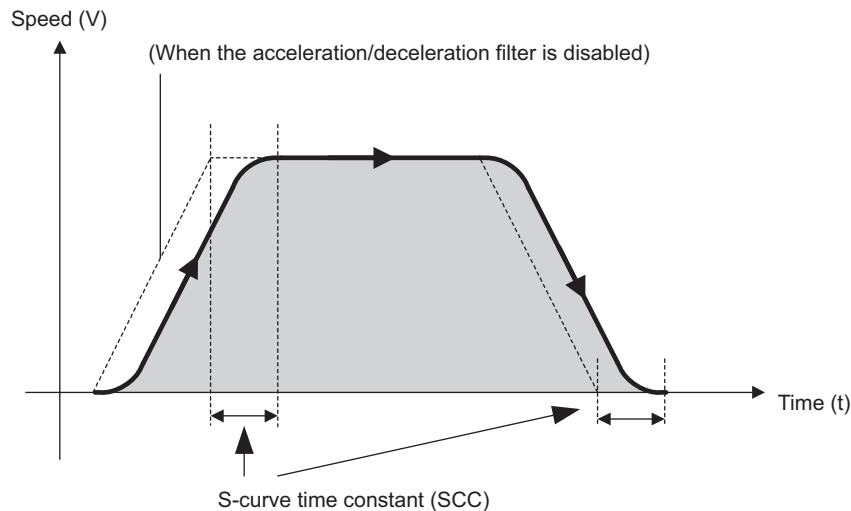
The format of the SCC instruction is as follows:

```
SCC [Logical_axis_name_1] S-curve_time_constant [Logical_axis_name_2] S-curve_time_constant
... ;
```

Item	Unit	Applicable Data
S-curve time constant	ms	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the SCC Instruction

This section describes the settings for the SCC instruction.



Specify a numerical value or register for the S-curve time constant for each axis by using the SCC instruction.

The setting range of the S-curve time constants depends on the Motion Control Function Module that is used, as shown below.

- For the SVR, SVR32, PO-01, and SVA-01 Motion Control Function Modules, the setting range is the same as the setting range for the $OW\Box\Box\Box3A$ (Filter Time Constant) setting parameter.
- For the SVC, SVC32, SVC-01, and SVB-01 Motion Control Function Modules, the setting range is the same as the setting range for the moving average time in the SERVOPACK parameters.

Refer to the following table for details on the setting range of the S-curve time constant.

Motion Control Function Modules	SCC Instruction Setting Range (ms)	Remarks
SVA-01	0 to 6,553	–
SVC, SVC32, SVC-01, SVB-01	0 to 510	For SGD-N, SGDB-N, SGDH+NS110/NS115, SGDS, SGDX, and SGD V SERVOPACKs
	–	The S-curve acceleration/deceleration cannot be used with the SGD J SERVOPACK because it does not have a parameter for the average movement time.
PO-01	0 to 6,553	–
SVR or SVR32	0 to 6,553	–



Note

1. If a reference value of more than 6,553 ms is input, a motion program alarm will occur regardless of which Motion Control Function Module is used.
2. If a reference value exceeds the upper limit (511 to 6,553 ms) when the SVC, SVC32, SVC-01, or SVB-01 Motion Control Function Module is used, bit 1 of the $IL\Box\Box\Box02$ (Setting Parameter Error) monitor parameter is set to 1, and the upper limit (510 ms) is set for the moving average time in the SERVOPACK parameters.

Programming Example

A programming example that uses the SCC instruction is given below.

The following example executes a MOV instruction with an S-curve time constant of 250 ms and a MOV instruction with an S-curve time constant of 500 ms.

For this example, the setting parameters are set as follows:

- Bits 0 to 3 (Speed Unit Selection) of the OW□□□03 setting parameter are set to 0 (reference units/s).
- Bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of the OW□□□03 setting parameter are set to 0 (reference units/s²).

```

INC;                                "Incremental Mode
VEL [A1]10000;                       "Set feed speed (reference units/s).
ACC [A1]20000;                       "Set acceleration rate (reference units/s2).
DCC [A1]20000;                       "Set deceleration rate (reference units/s2).
SCC [A1]250;                         "Change S-curve time constant (ms).
MOV [A1]20000;                       "Positioning
DL00000 = 500;                       "S-curve time constant (ms)
SCC [A1]DL00000;                   "Change S-curve time constant (ms).
MOV [A1]20000;                       "Positioning
END;

```

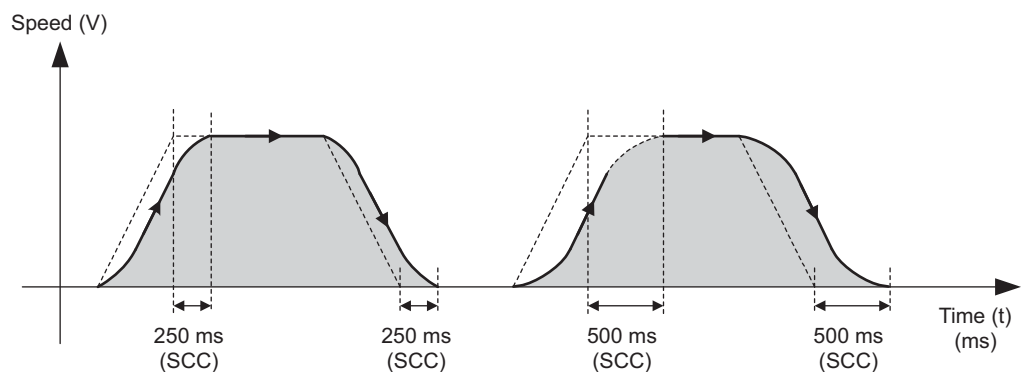


Fig. 6.12 Programming Example for the SCC Instruction

Additional Information on the SCC Instruction

◆ Related Motion Parameters

The SCC instruction changes the S-curve time constants in the setting parameters.

Parameter Name	Register Address	Description
Filter Time Constant	OW□□□3A	Sets the acceleration/deceleration filter time constants (1 = 0.1 ms). <ul style="list-style-type: none"> • Make sure that reference pulse distribution has been completed (i.e., that bit 0 of IW□□□0C is 1) before you change the filter time constant. • Change the time constant only after you select the filter type to use in bits 8 to B (Filter Type Selection) of the OW□□□03 setting parameter.

The S-filter time constants can be changed by directly changing the settings of the OW□□□3A (Filter Time Constant) setting parameters instead of by using the SCC instruction. Refer to the following table for details on how to directly change the S-curve time constants.

Motion Control Function Modules	Specification	Setting Procedure
SVR, SVR32, SVA-01, PO-01	If S-curve acceleration/deceleration is enabled, the axes move according to the S-curve time constants that are set in the OW□□□3A (Filter Time Constant) setting parameters.	Set the S-curve time constants in the OW□□□3A (Filter Time Constant) setting parameters.
SVC, SVC32, SVC-01, SVB-01	If S-curve acceleration/deceleration is enabled, the axes move according to the moving average filter time constants in the SERVOPACK parameters.	Set the S-curve time constants in the OW□□□3A (Filter Time Constant) setting parameters. Then, set the OW□□□08 (Motion Commands) setting parameters to 12 (Change Filter Time Constant) to write the new S-curve time constants to the SERVOPACK.(*).

* The SVC, SVC32, SVC-01, and SVB-01 Function Modules can automatically set the moving average filter time constants in the SERVOPACK parameters to the values of the OW□□□3A (Filter Time Constant) setting parameters. If this automatic writing function is enabled, you do not need to set the OW□□□08 (Motion Commands) setting parameters to 12 (Change Filter Time Constant).

Refer to the following manual for details on how to use the automatic writing function.

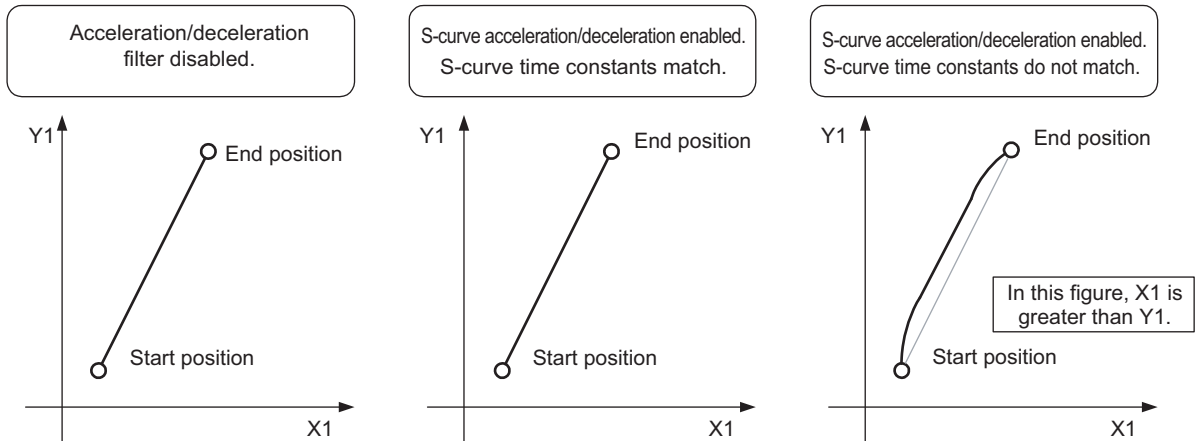
📖 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)

◆ Movement Paths for Interpolation Instructions and S-curve Acceleration/Deceleration

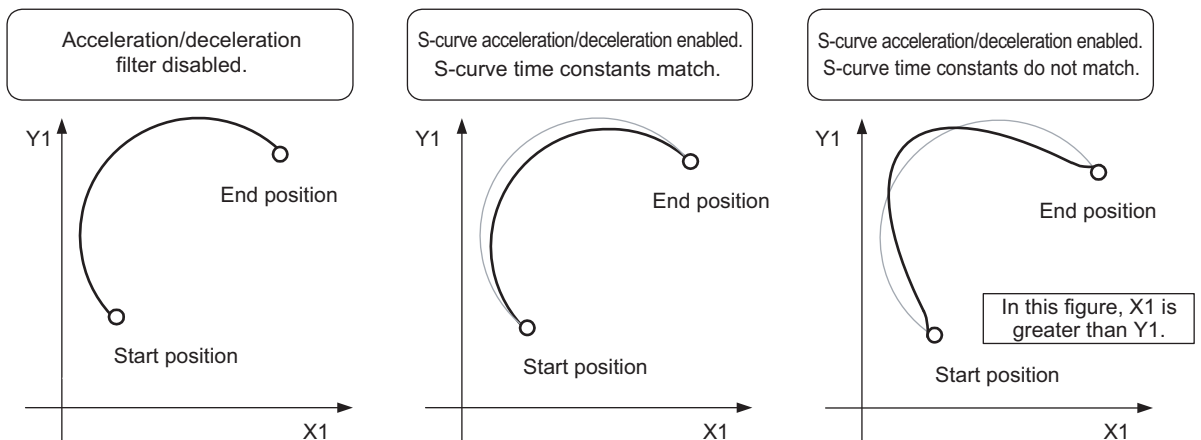
The S-curve acceleration/deceleration rate affects the movement path for the MVS, MCW, MCC, and SKP interpolation instructions.

- To achieve the same movement path as when the S-curve acceleration/deceleration is disabled for linear interpolation, set the same S-curve time constant for all of the axes that are involved in the interpolation.
- When S-curve acceleration/deceleration is enabled for circular interpolation, the movement path will not be the same as when S-curve acceleration/deceleration is disabled.

Linear Interpolation Movement Paths



Circular Interpolation Movement Paths



◆ Filter Type Selection

Before you enable S-curve acceleration/deceleration, set the filter type for each axis by setting bits 8 to B (Filter Type Selection) if OW□□□03 to 2 (Moving Average Filter).

Parameter Name	Register Address	Filter Type
Function Settings 1 Filter Type Selection	OW□□□03 Bits 8 to B	0: No filter (default) 1: Exponential acceleration/deceleration filter 2: Moving average filter

If you are using the SVC, SVC32, SVC-01, or SVB-01 Motion Control Function Modules and have the automatic writing function disabled, set the OW□□□08 (Motion Commands) setting parameter to 13 (Change Filter Type) to write the settings to the SERVOPACK parameters.

The following programming example shows how to change the filter type from the motion program.

```
:
:
```

```
"See if changing the filter type is OK.
```

```
IOW IW8008 = = 0;
```

```
"Wait for there to be no motion command in progress.
```

```
IOW IB800C0 = = 1;
```

```
"Wait for reference pulse distribution to be completed.
```

```
"Select the Moving Average Filter for the filter type.
```

```
DW00000 = OW8003 & F0FFH; "Retain all information other than the Filter Type Selection.
```

```
OW8003 = DW00000 | 0200H; "Filter type = Moving average filter
```

```
"Write the filter type from the built-in SVB/SVB-01 Module to the SERVOPACK.
```

```
OW8008 = 13;
```

```
Request changing the filter type.
```

```
IOW IW8008 = = 13;
```

```
"Wait for the Change Filter Type operation to become active.
```

```
IOW IB80098 = = 1;
```

```
"Wait for execution of the motion command to be completed.
```

```
OW8008 = 0;
```

```
"Clear the request.
```

```
IOW IW8008 = = 0;
```

```
"Wait for there to be no motion command in progress.
```

```
:
:
```

Information

When using the SVR, SVR32, PO-01, or SVA-01 Motion Control Function Module, the above programming is not required.

The above programming is also not required even when using the SVC, SVC32, SVC-01, or SVB-01 Motion Control Function Modules if automatic writing to the SERVOPACK parameters is enabled.

Information

Refer to the following manuals for details on how to automatically write settings to the SERVOPACK parameters for the SVC, SVC32, SVC-01, or SVB-01 Motion Control Function Module.

📖 *MP2000 Series Built-in SVB/SVB-01 Motion Module User's Manual* (Manual No.: SIEP C880700 33)

📖 *MP2000 Series Built-in SVC/SVC-01 Motion Module User's Manual* (Manual No.: SIEP C880700 41)

📖 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)

Set Speed (VEL)

The VEL instruction changes the feed speeds of the specified axes for all of the following axis movement instructions.

- MOV (Positioning)
- EXM (External Positioning)

In this manual, the above axis movement instructions and the MVT (Set-time Positioning) instruction are referred to as positioning instructions, and the term positioning speed refers to a feed speed for those instructions.

The values can be changed for up to 32 axes with one instruction. The positioning speed for any unspecified axis is not changed.

The positioning speeds that are set by the VEL instruction remain in effect until they are changed by another VEL instruction.

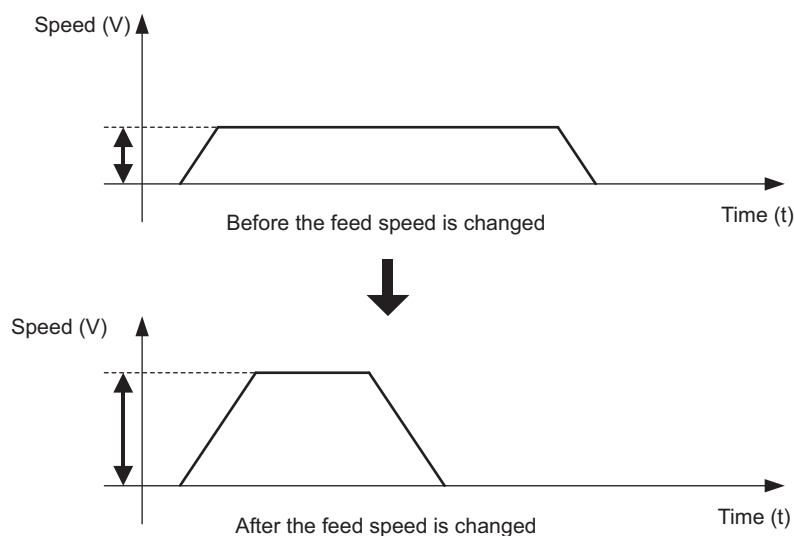


Fig. 6.13 Set Speed

Information The VEL instruction changes the positioning speed for the MOV and EXM positioning instructions. Use an F reference or the IFP instruction to set the feed speed for the MVS, MCW, MCC, or SKP interpolation instruction.

Format

The format of the VEL instruction is as follows:

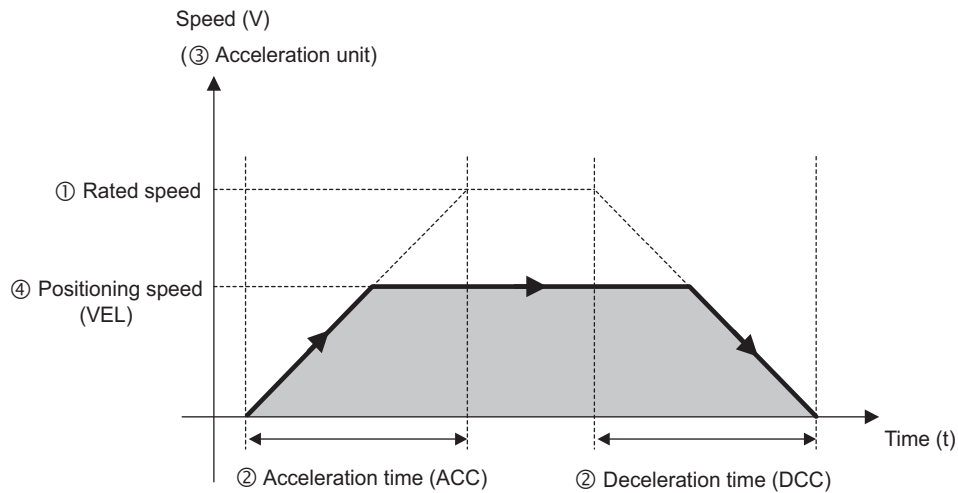
VEL [*Logical_axis_name_1*] *Positioning_speed* [*Logical_axis_name_2*] *Positioning_speed* ... ;

Item	Unit	Applicable Data
Positioning speed	10 ⁿ reference units/min, reference units/s, 0.01% (percentage of rated speed), or 0.0001% (percentage of rated speed)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Note: The unit is set in bits 0 to 3 (Speed Unit Selection) of the OW□□□03 setting parameter.

Settings for the VEL Instruction

This section describes the settings for the VEL instruction.



① Rated Speed

The rated speed for each axis is set in fixed parameter No. 34 (Rated Motor Speed).

② Acceleration Times and Deceleration Times

Use the ACC and DCC instructions to set the acceleration/deceleration times for each axis.

The times that are set with the ACC instruction designate the amount of time required to accelerate to or decelerate from the rated speed.

③ Speed Unit

The speed unit for each axis is set in bits 0 to 3 (Speed Unit Selection) of the OW□□□03 setting parameter.

The default setting for this parameter is 1 (10ⁿ reference units/min).

Parameter Name	Register Address	Speed Unit	Reference Range
Function Settings 1 Speed Unit Selection	OW□□□03 Bits 0 to 3	0: Reference units/s	0 to 2 ³¹ - 1 (reference units/s)
		1: 10 ⁿ reference units/min	0 to 2 ³¹ - 1 (10 ⁿ reference units/min)
		2: 0.01%	0 to 32,767 (0.01%)
		3: 0.0001%	0 to 3,276,700 (0.0001%)

Information The setting unit for the VEL instruction when bit 1 (10ⁿ reference units/min) is selected for the OW□□□03 setting parameter is determined by fixed parameter No. 4 (Reference Unit Selection).

Fixed Parameter No. 4 (Reference Unit Selection)	Speed Unit (10 ⁿ reference units/min)	Remarks
pulse	1 = 1,000 pulse/min	<ul style="list-style-type: none"> When the Reference Unit Selection is set to <i>Pulses</i>, n = 3. When the Reference Unit Selection is set to any setting other than <i>Pulses</i>, n = fixed parameter No. 5 (Number of Digits Below Decimal Point).
mm	1 = 1 mm/min	
deg	1 = 1 deg/min	
inch	1 = 1 inch/min	
μm	1 = 1 μm/min	

④ Positioning Speed

The positioning speed for each axis is set by specifying a numerical value or register in the VEL instruction.

Programming Example

A programming example that uses the VEL instruction is given below.

The following example executes the MOV instruction with a positioning speed that is 40% of the rated speed, and then executes the MOV instruction with a positioning speed that is 20% of the rated speed.

```

INC;                "Incremental Mode
ACC [A1]5000;       "Change acceleration time (ms).
DCC [A1]5000;       "Change deceleration time (ms).
VEL [A1]4000;       "Change the feed speed (0.01%).
MOV [A1]3000000;    "Positioning
VEL [A1]2000;       "Change the feed speed (0.01%).
MOV [A1]3000000;    "Positioning
END;
```

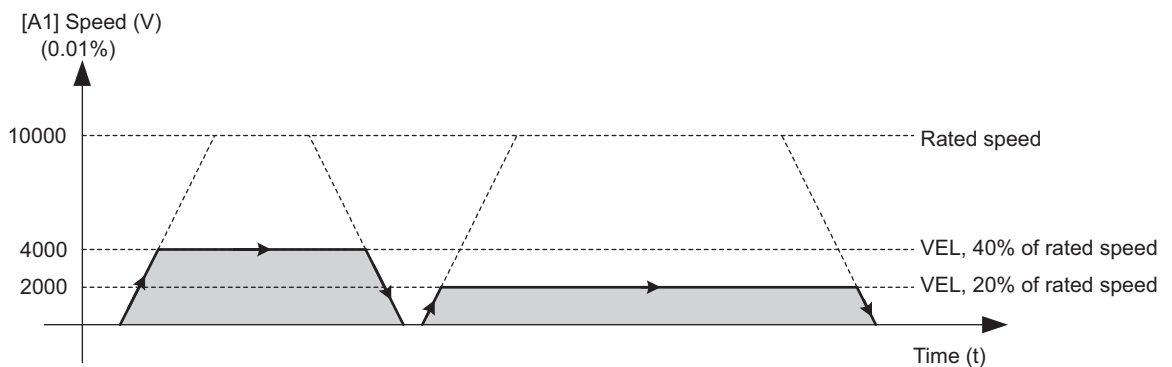


Fig. 6.14 Programming Example for the VEL Instruction

Additional Information on the VEL Instruction

This section describes three additional items about the VEL instruction.

◆ Related Motion Parameters

The VEL instruction changes the positioning speeds in the setting parameters.

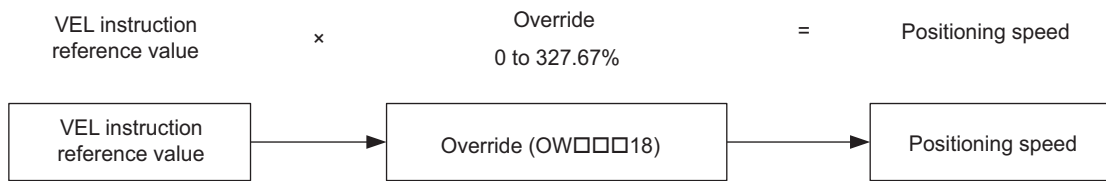
Parameter Name	Register Address	Description
Speed Reference Setting	OL□□□10	Sets the speed reference.

The positioning speeds can be changed by changing the settings of the OL□□□10 (Speed Reference Setting) setting parameters instead of by using the VEL instruction.

◆ Overrides

You can use the OW□□□18 (Override) setting parameters to specify what percentage of the positioning speed specified by a VEL instruction to actually execute (i.e., the output ratio). The unit for the Override parameters is 0.01%.

The default value for the OW□□□18 (Override) setting parameters is 10,000 (100.00%).



Overrides

An override allows you to change the output ratio of the axis movement speed reference for interpolation motion language instructions.

The OW□□□18 (Override) setting parameters can be changed during axis movement.

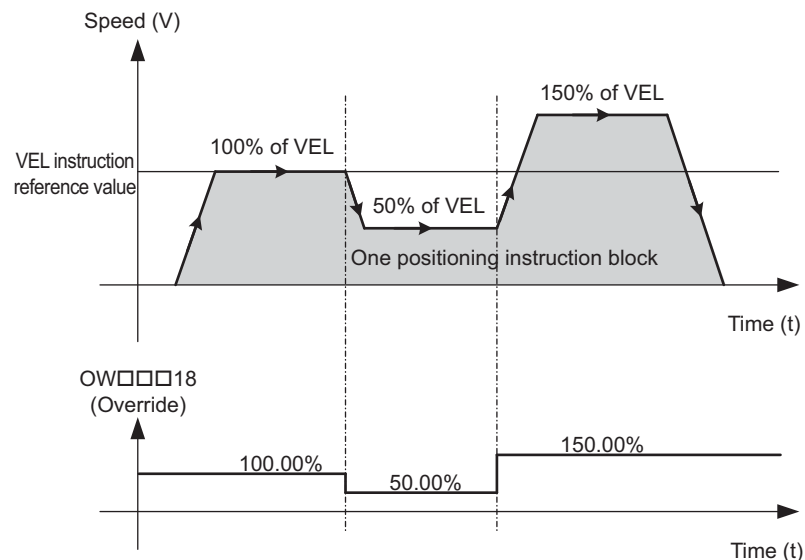


Fig. 6.15 OW□□□18 (Override) and Positioning Instructions

Information

1. The SVR and SVR32 Function Modules do not support the OW□□□18 (Override) setting parameters.
2. For the MVT instruction, the positioning speed used as the base for the override is not the VEL instruction reference value. The positioning speed changed by executing the MVT instruction is used as the base speed for the override.
3. If you use an override for the MVT instruction, positioning will not be completed within the specified time. The positioning speed during execution of the MVT instruction is calculated with an override value of 100%.
4. The speed unit of the rated speed that is specified in the motion fixed parameters is different from the speed unit that is used for VEL instruction in a motion program.

Speed	Speed Unit
Fixed Parameter No. 34 (Rated Motor Speed)	Revolutions/min
Positioning speed (VEL)	Reference units/s, 10 ⁿ reference units/min, 0.01%, or 0.0001%

Refer to the following section for how to calculate the rated speed according to the speed unit of the VEL instruction.

Motor Speed Specifications (page 6-37)

◆ Motor Speed Specifications

In addition to the VEL instruction reference range, the rated motor speed and maximum speed must be taken into consideration to determine the set value for the VEL instruction. To avoid causing an over-speed, check the speed specifications of your motor before you set a value for the VEL instruction.

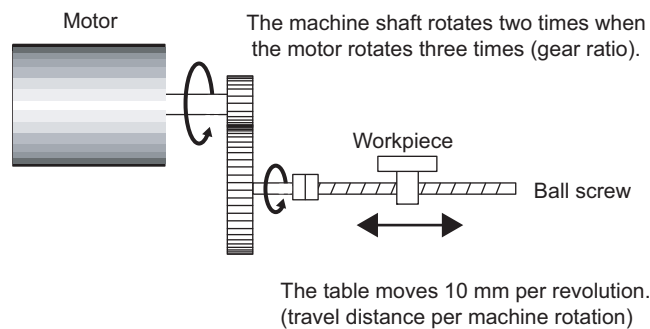
Information For rotational motors, the speed specification is expressed in rotations per specified time period. The rated speed when the speed unit is 10^n reference units/min is calculated according to the fixed parameter settings, as shown below.

• Parameter Setting Example: When Electronic Gear Is Enabled

The electronic gear is enabled if fixed parameter No. 4 (Reference Unit Selection) is set to any unit other than pulses.

Fixed Parameters

- No. 4: Reference Unit Selection = mm
- No. 5: Number of Digits Below Decimal Point = 3
- No. 6: Travel Distance per Machine Rotation = 10,000 reference units
- No. 8: Servomotor Gear Ratio Term = 3
- No. 9: Machine Gear Ratio Term = 2
- No. 34: Rated Motor Speed = 3,000 revolutions/min



When the electronic gear is enabled, n in the speed unit reference (10^n reference units/min) is the number of digits below the decimal point. Therefore, the speed unit is as follows:

$$(10^n \text{ reference units/min}) = (10^3 \times 0.001 \text{ mm/min}) = (\text{mm/min})$$

The machine shaft rotation speed when the motor rotates at the rated speed is as follows:

$$\begin{aligned} &\text{Rated motor speed (revolutions/min)} \times \text{Gear ratio} \\ &= 3,000 \times (2/3) = 2,000 \text{ (revolutions/min)} \end{aligned}$$

If the number of rotations of the machine shaft is converted into reference units (0.001 mm),

$$\begin{aligned} &\text{Travel distance per machine rotation (0.001 mm/revolution)} \times 2,000 \text{ (revolutions/min)} \\ &= 10,000 \times 2,000 = 20,000,000 \text{ (0.001 mm/min)} \end{aligned}$$

If the speed unit is (mm/min),

$$20,000,000 \text{ (0.001 mm/min)} = 20,000 \text{ (mm/min)}$$

- **Parameter Setting Example: Electronic Gear Disabled, SVA-01 Function Module**
The electronic gear is disabled if fixed parameter No. 4 (Reference Unit Selection) is set to *Pulses*.

Fixed Parameters

- No. 4: Reference Unit Selection = Pulses
- No. 22: Pulse Counting Mode Selection = A/B × 4 (× 4)
- No. 34: Rated Motor Speed = 3,000 revolutions/min
- No. 36: Number of Pulses per Motor Rotation (before multiplication) = 16,384 pulses/revolution

When the electronic gear is disabled, n in the speed unit reference (10^n reference units/min) is 3.
Therefore, the speed unit is as follows:

$$(10^3 \text{ reference units/min}) = (10^3 \text{ pulses/min}) = (1,000 \text{ pulses/min})$$

If the rated motor speed is converted into pulses,

Rated motor speed (revolutions/min) × (Number of pulses per motor rotation (pulses/revolution) × multiplier)

$$= 3,000 \times (16,384 \times 4) = 196,608,000 \text{ (pulses/min)}$$

With a speed unit of 1,000 pulses/min,

$$196,608,000 \text{ (pulses/min)} = 196,608 \text{ (1,000 pulses/min)}$$

- **Parameter Setting Example: Electronic Gear Disabled, SVC, SVC32, SVR, SVR32, SVC-01, SVB-01, or PO-01 Function Module**

The electronic gear is disabled if fixed parameter No. 4 (Reference Unit Selection) is set to *Pulses*.

Fixed Parameters

- No. 4: Reference Unit Selection = Pulses
- No. 34: Rated Motor Speed = 3,000 revolutions/min
- No. 36: Number of Pulses per Motor Rotation = 65,536 pulses/revolution

When the electronic gear is disabled, n in the speed unit reference (10^n reference units/min) is 3.
Therefore, the speed unit is as follows:

$$(10^3 \text{ reference units/min}) = (10^3 \text{ pulses/min}) = (1,000 \text{ pulses/min})$$

If the rated motor speed is converted into pulses,

Rated motor speed (revolutions/min) × Number of pulses per motor rotation (pulses/revolution)

$$= 3,000 \times 65,536 = 196,608,000 \text{ (pulses/min)}$$

With a speed unit of 1,000 pulses/min,

$$196,608,000 \text{ (pulses/min)} = 196,608 \text{ (1,000 pulses/min)}$$

Fixed parameters other than those given in the above examples may also need to be set correctly in order to ensure proper axis operation.

Refer to the following manual for details on motion parameters.

 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEPSIEP C880725 11)

Set Maximum Interpolation Feed Speed (FMX)

The FMX instruction sets the maximum speed for the MVS, MCW, MCC, and SKP interpolation instructions.

The maximum interpolation feed speed that is set by the FMX instruction remains in effect until it is changed by another FMX instruction.

The maximum interpolation feed speed is not set when program operation starts.

The FMX instruction must be executed before any of the following interpolation instructions are executed.

- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Skip Function)
- IFP (Set Interpolation Feed Speed Ratio)
- IAC (Change Interpolation Acceleration Time)
- IDC (Change Interpolation Deceleration Time)
- IDH (Change Interpolation Deceleration Time for Temporary Stop)

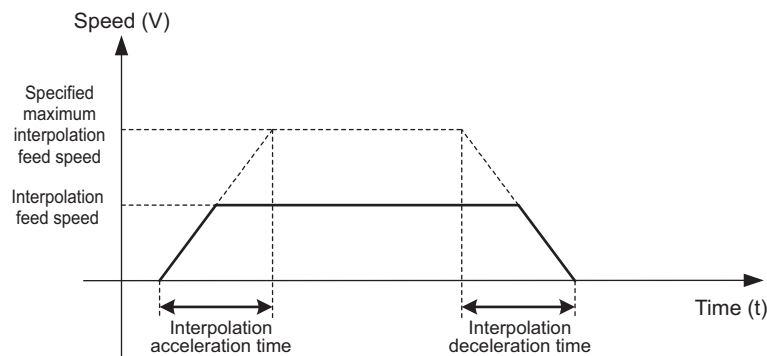


Fig. 6.16 Set Maximum Interpolation Feed Speed



Note

A motion program alarm will occur if any interpolation instruction (MVS, MCW, MCC, SKP, IFP, IAC, IDC, or IDH) is executed before the FMX instruction is executed.

Information

1. Interpolation instructions are processed with the assumption that the maximum interpolation feed speed is set in advance. For example, the IAC, IDC, and IDH instructions all designate the time required to reach the maximum interpolation feed speed from a speed of 0. Therefore, the maximum interpolation feed speed must be set before any of these instructions can be executed.
2. The FMX instruction is not related to any setting parameters. The maximum interpolation feed speed that is specified by the FMX instruction is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify the maximum interpolation feed speed.

Format

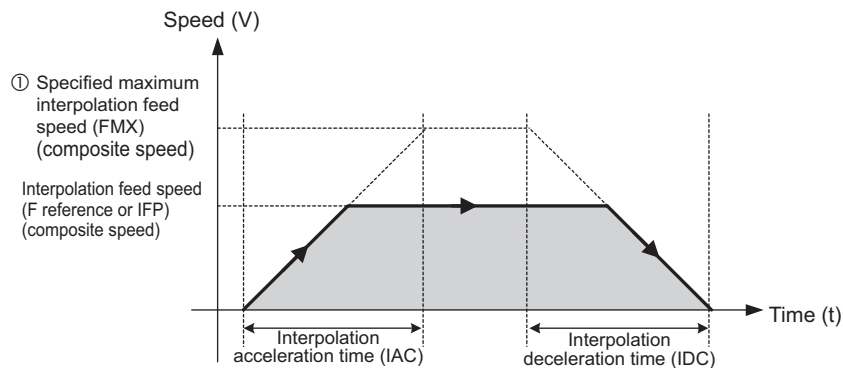
The format of the FMX instruction is as follows:

```
FMX Tmaximum_interpolation_feed_speed;
```

Item	Unit	Applicable Data
Maximum interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the FMX Instruction

This section describes the settings for the FMX instruction.



① Specified Maximum Interpolation Feed Speed

The maximum interpolation feed speed is set by specifying a register or a numerical value after the character "T" in the FMX instruction. The valid range for the maximum interpolation feed speed is 1 to $2^{31} - 1$ (reference units/min).

The maximum interpolation feed speed that is set is used for all interpolation instructions.

Therefore, the FMX instruction must be executed at the beginning of the motion program before the MVS, MCW, MCC, or SKP interpolation instruction can be used.

Programming Example

A programming example that uses the FMX instruction is given below.

```
INC;                                "Incremental Mode
FMX T300000;                         "Set maximum interpolation feed speed.
IAC T4000;                            "Change interpolation acceleration time (ms).
IDC T4000;                            "Change interpolation deceleration time (ms).
IFP P75;                              "Set interpolation feed speed ratio (%).
MVS [A1]30000 [B1]30000;              "Linear interpolation
MVS [A1]30000 [B1]30000 F150000;    "Linear interpolation (F reference)
END;
```

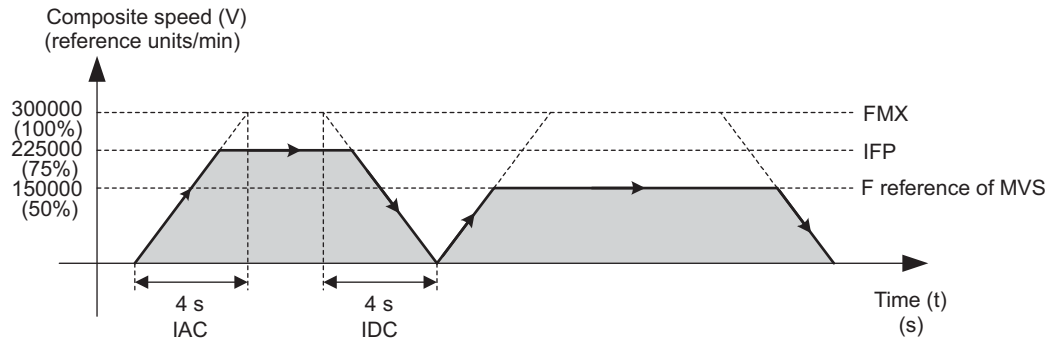


Fig. 6.17 Programming Example for the FMX Instruction

Set Maximum Individual Axis Speeds for Interpolation (IFMX)

The IFMX instruction sets the maximum feed speeds for individual axes that are used in the MVS, SKP, MCW, and MCC interpolation instructions.

The maximum individual axis feed speeds that are set by the IFMX instruction remain in effect until they are changed by another IFMX instruction.

If an actual axis feed speed exceeds a value that was set with the IFMX instruction, a motion program alarm will occur and all axes will stop immediately.

The maximum individual axis feed speeds for interpolation are not set when program operation starts. The individual axes will operate without any speed limits.

A timing chart for linear interpolation of two axes (A1 and B1) when the IFMX instruction has been executed to set the maximum feed speed only for the A1 axis is given below.

Composite Speed (A1 and B1 Axes)

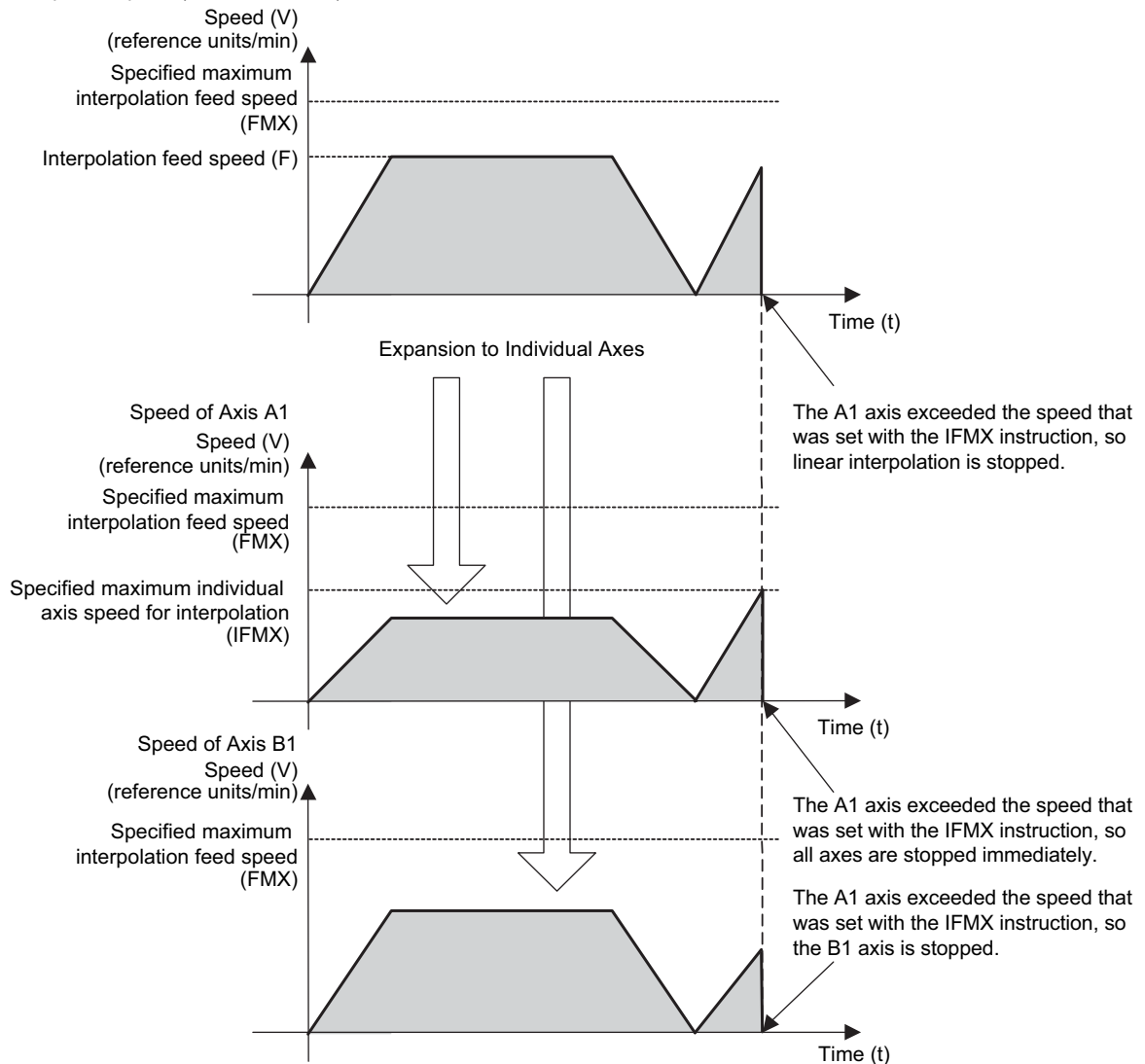


Fig. 6.18 Maximum Individual Axis Speed Settings for Interpolation

Information

1. If the IFMX instruction is not executed or if a maximum speed of 0 is set, the individual axes will operate without any speed limits.
2. The unit of the set value of the IFMX instruction is converted in the motion program from reference units/min to reference units/scan. When the unit is converted, the resulting value is rounded down to the nearest integer to determine if the axis speed has exceeded the maximum speed. This is different from processing for the interpolation feed speed (F). Therefore, depending on the high-speed scan time and interpolation feed speed, axis operation may occur even if the axis exceeds the speed limit that was set with the IFMX instruction but does not exceed the interpolation feed speed (F). The interpolation feed speed will never be exceeded.

The following formula is used to convert the interpolation feed speed and the set values of the IFMX instruction (reference units/min).

Interpolation feed speed (or speed limit) [reference units/scan] = F value (or set value in IFMX instruction)/60 (s)/1,000 (ms) × Ts, where Ts = high-speed scan time

Format

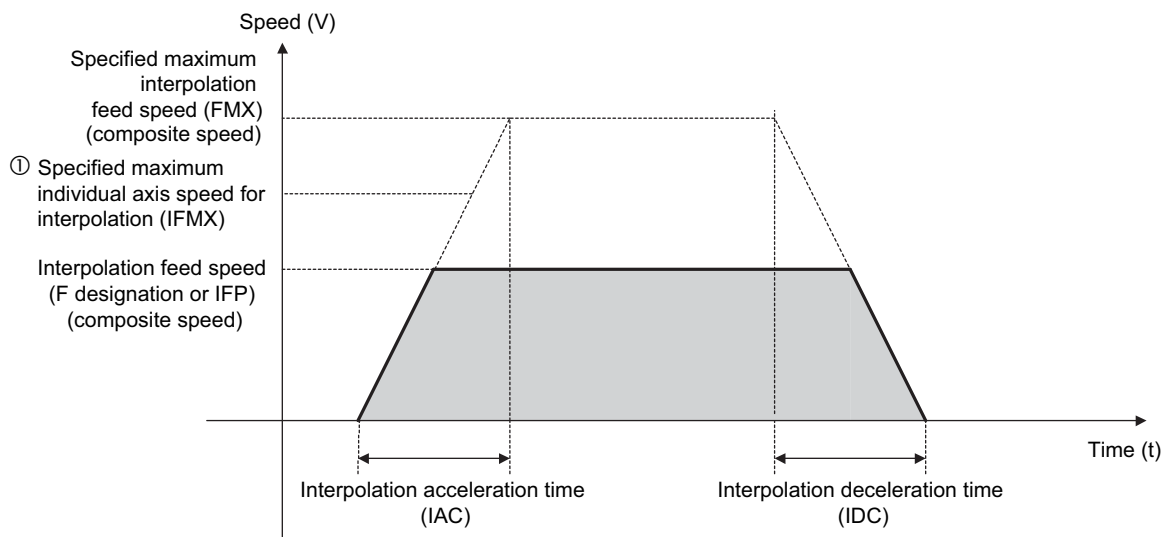
The format of the IFMX instruction is as follows:

```
IFMX
[Logical_axis_name_1]Maximum_Individual_axis_speed_for_interpolation [Logical_ax-
is_name_2]Maximum_Individual_axis_speed_for_interpolation ...;
```

Item	Unit	Applicable Data
Maximum individual axis speed for interpolation	Reference units/min or reference units/s (specified with FUT instruction)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the IFMX Instruction

This section describes the settings for the IFMX instruction.



① Maximum Individual Axis Speed for Interpolation

The maximum individual axis feed speeds during interpolation are set by specifying registers or numerical values in the IFMX instruction.

The setting range for the IFMX instruction is 0 to $2^{31} - 1$ (reference units/min).

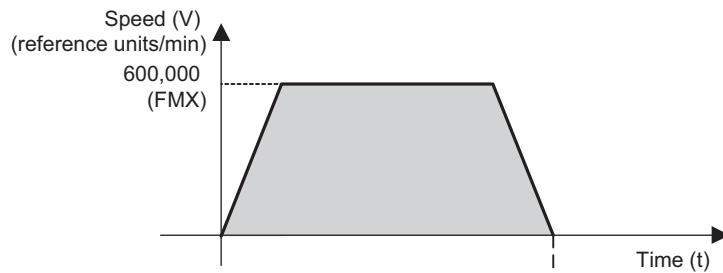
If you set 0 for the IFMX instruction, the maximum individual axis feed speeds for interpolation will not be set and the individual axes will operate without any speed limits.

Programming Example

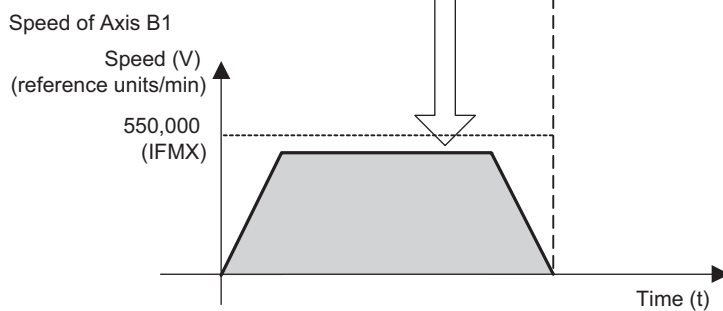
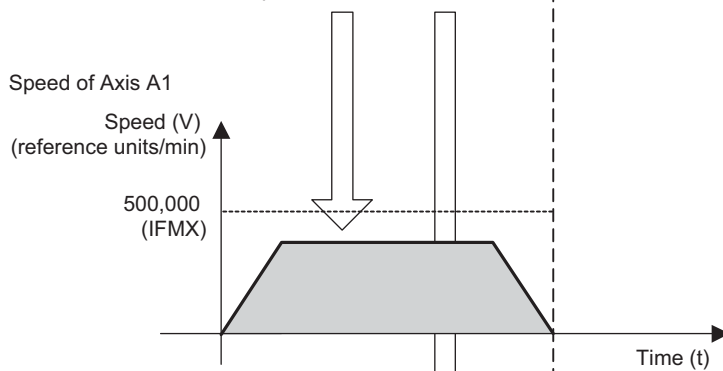
A programming example that uses the IFMX instruction is given below.

FMX T600000;	"Set maximum interpolation feed speed."
IFMX [A1]500000 [B1]550000;	"Set maximum individual axis feed speeds for interpolation."
INC;	"Incremental Mode"
IAC T500;	"Interpolation acceleration time = 500 ms"
IDC T500;	"Interpolation deceleration time = 500 ms"
MVS [A1]30000 [B1]40000 F600000;	"Linear interpolation instruction"
END;	

Composite Speed (A1 and B1 Axes)



Expansion to Individual Axes



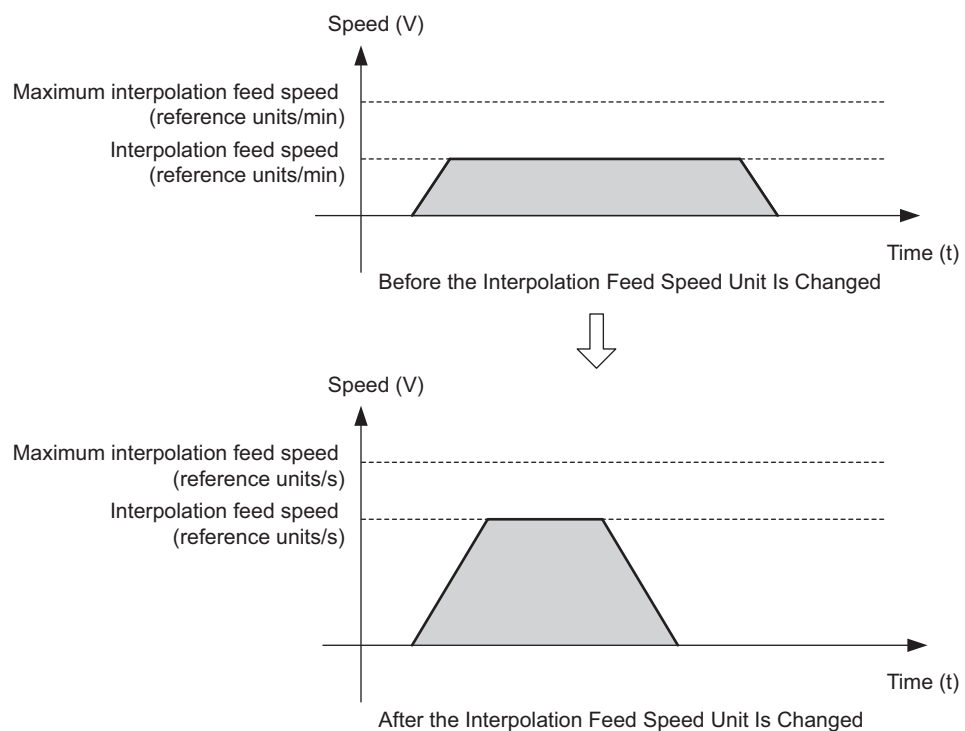
Change Interpolation Feed Speed Unit (FUT)

The FUT instruction can be used to change the speed unit for the following interpolation instructions.

- Set Maximum Interpolation Feed Speed (FMX)
- Set Maximum Individual Axis Speeds for Interpolation (IFMX)
- Linear Interpolation (MVS)
- Circular Interpolation (MCW/MCC)
- Helical Interpolation (MCW/MCC)
- Linear Interpolation with Skip Function (SKP)

The interpolation feed speed unit that is selected is retained until it is set again with the FUT instruction.

The interpolation feed speed unit is reference units/min when program operation starts.



Information

1. If the FUT instruction has not been executed, the interpolation feed speed unit is reference units/min.
2. If the FUT instruction is set out of range, a compiler error will occur.
3. You can use the FUT instruction with the following versions.

Machine Controller or MPE720	Applicable Versions
MP3000-series Machine Controller	Ver. 1.08 or later
MPE720 Version 7	Version 7.23 or later

Format

The format of the FUT instruction is as follows:

FUT Interpolation feed speed unit number;

Item	Unit	Applicable Data
Interpolation feed speed unit number	–	Directly designated value 0: Reference units/min 1: Reference units/s



Note

When the FUT instruction is executed to change the unit, the values for FMX, IFMX, F, and IFP are initialized to 0. After you change the unit, set the interpolation feed speeds according to the new unit.

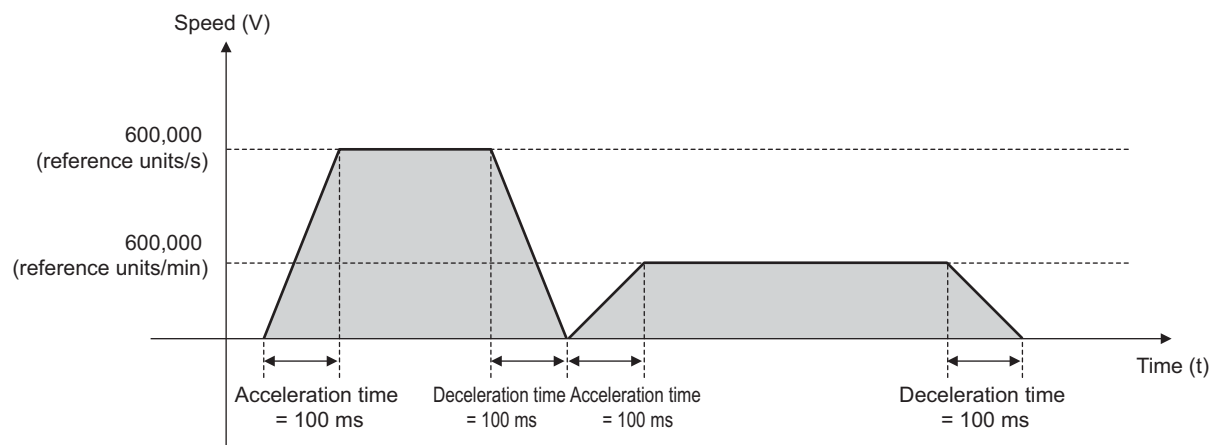
Programming Example

A programming example that uses the FUT instruction is given below.

```

FUT U1;           "Change interpolation feed speed from reference units/min to reference units/s."
INC;             "Incremental Mode"
FMX T600000;    "Maximum interpolation feed speed (reference units/s)"
IAC T100;       "Acceleration time = 100 ms"
IDC T100;       "Deceleration time = 100 ms"
MVS [A1]10000 F600000; "Linear interpolation feed speed = 600,000 reference units/s"
FUT U0;         "Change interpolation feed speed from reference units/s to reference units/min."
FMX T600000;    "Maximum interpolation feed speed (reference units/min)"
MVS [A1]10000 F600000; "Linear interpolation feed speed = 600,000 reference units/min"
END;

```



Set Interpolation Feed Speed Ratio (IFP)

The IFP instruction sets the feed speed for the following axis movement instructions. The feed speed is specified as a percentage of the maximum interpolation feed speed.

- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Linear Interpolation with Skip Function)

In this manual, the above axis movement instructions are referred to as interpolation instructions, and the term interpolation feed speed refers to the feed speed for those instructions. The interpolation feed speed that is set by the IFP instruction remains in effect until it is changed by another IFP instruction or until an F reference is made in an interpolation instruction.

The interpolation feed speed is not set when program operation starts. Set the interpolation feed speed by executing the Set Interpolation Feed Speed Ratio (IFP) instruction or by specifying an F reference before executing any interpolation instructions.

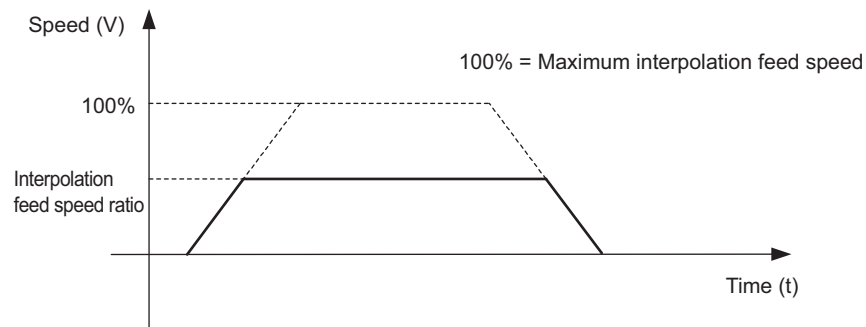


Fig. 6.19 Set Interpolation Feed Speed Ratio



Important

1. You must execute the Set Maximum Interpolation Feed Speed (FMX) instruction before you execute the IFP instruction. A motion program alarm will occur if the IFP instruction is executed without first executing the FMX instruction.
2. A motion program alarm will occur if an interpolation instruction is executed without setting the interpolation feed speed even once.

Information

1. F references can be used to specify the interpolation feed speed by writing a numerical value or register following the character F in interpolation instructions. The interpolation feed speed is specified in reference units/min.
2. If an IFP instruction is executed after an F reference, the interpolation feed speed specified by the F reference will be canceled. If an F reference is made after an IFP instruction is executed, the interpolation feed speed specified by the IFP instruction will be canceled.
3. The IFP instruction sets the feed speed for the MVS, MCW, MCC, and SKP interpolation instructions. Use the VEL instruction to set the feed speed for the MOV and EXM positioning instructions.
4. The IFP instruction is not related to any setting parameters. The interpolation feed speed ratio that is specified by the IFP instruction is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify the interpolation feed speed ratio.

Format

The format of the IFP instruction is as follows:

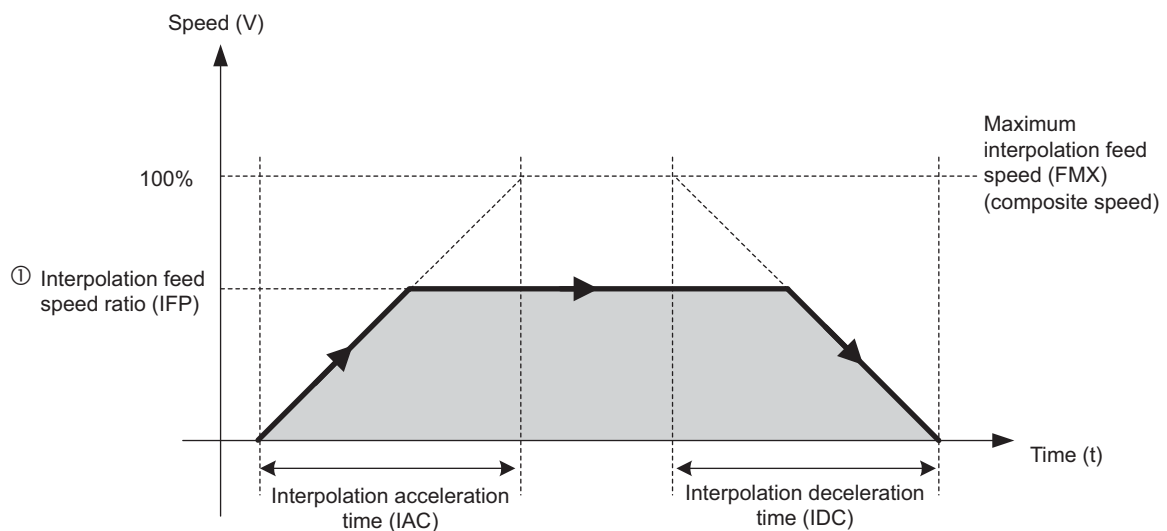
IFP *Pinterpolation_feeding_speed_ratio*;

Item	Unit	Applicable Data
Interpolation feed speed ratio	%	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Information You cannot place an IFP instruction in the same block as any interpolation instruction (MVS, MCW, MCC, or SKP).

Settings for the IFP Instruction

This section describes the settings for the IFP instruction.



① Interpolation Feed Speed Ratio

The interpolation feed speed ratio is set by specifying a register or a numerical value following the character “P” in the IFP instruction.


The time set with the IFP instruction designates the ratio of the interpolation feed speed to the maximum interpolation feed speed.

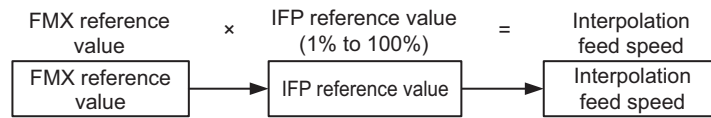
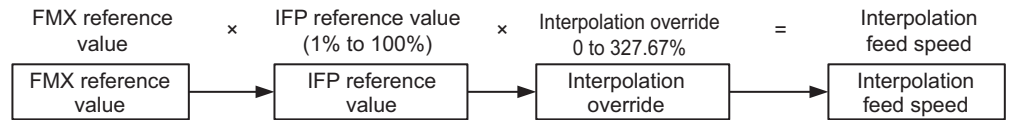
The interpolation feed speed is the composite speed of all axes specified by the MVS, MCW, MCC, and SKP interpolation instructions.

The valid range for the interpolation feed speed ratio is 1% to 100%.

You can select whether to apply an interpolation override to the interpolation feed speed.

Refer to the following section for how to use interpolation overrides.

 *Work Registers (page 1-23)*

Example When Not Specifying an Interpolation Override**Example** When Specifying an Interpolation Override

Important

A motion program alarm will occur if a value that exceeds 100% is specified for the IFP reference value (%).

Information

1. The interpolation feed speed can be specified by using either an IFP instruction or an F reference.

Refer to the following section for details on the interpolation feed speed.

Linear Interpolation (MVS)–Settings for the MVS Instruction (page 6-86)

2. If the interpolation feed speed with interpolation override applied exceeds the FMX reference value, the output value of the interpolation feed speed will be reset to the FMX reference value.

Programming Example

A programming example that uses the IFP instruction is given below.

```

INC;                               "Incremental Mode
FMX T300000;                        "Set maximum interpolation feed speed (reference units/min).
IAC T4000;                          "Change interpolation acceleration time (ms).
IDC T4000;                          "Change interpolation deceleration time (ms).
IFP P75;                             "Set interpolation feed speed ratio (%).
MVS [A1]30000 [B1]30000;           "Linear interpolation
DL00000 = 50;                       "Interpolation feed speed ratio (%)
IFP PDL00000;                      "Set interpolation feed speed ratio (%).
MVS [A1]30000 [B1]30000;           "Linear interpolation
END;

```

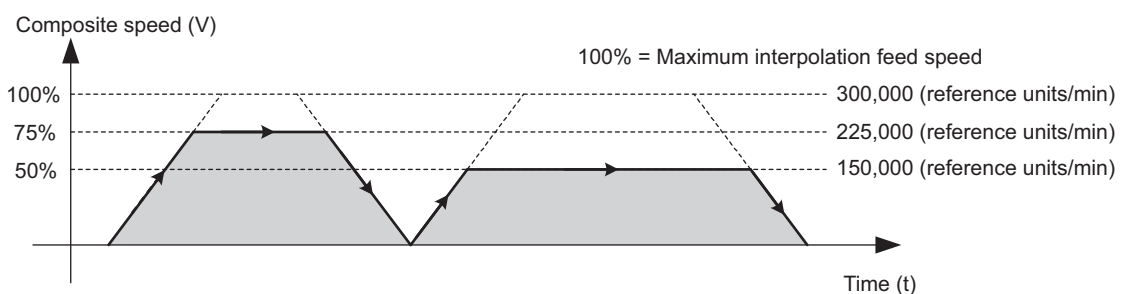


Fig. 6.20 Programming Example for IFP Instruction

Change Interpolation Acceleration Time (IAC)

The IAC instruction changes the interpolation acceleration times for the following axis movement instructions.

- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Linear Interpolation with Skip Function)

The FMX instruction must be executed first before an IAC instruction is executed.

The acceleration time that is set by the IAC instruction remains in effect until it is changed by another IAC instruction.

The interpolation acceleration time is set to 0 ms when program operation starts.

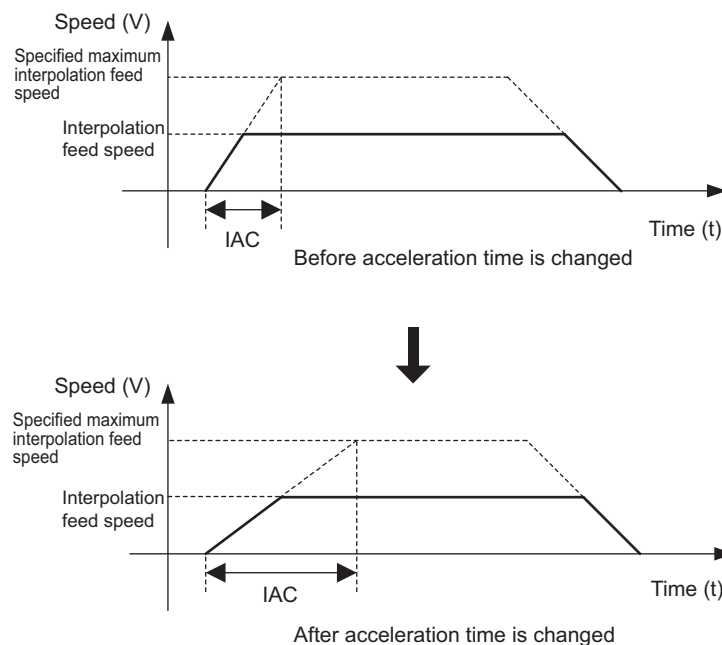


Fig. 6.21 Change Interpolation Acceleration Time

Information

1. The IAC instruction changes the acceleration time for the MVS, MCW, MCC, and SKP interpolation instructions.
Use the ACC instruction to set the acceleration time for the MOV, EXM, and MVT positioning instructions.
2. The IAC instruction is not related to any setting parameters.
The interpolation acceleration time specified by the IAC instruction is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify the interpolation acceleration time.

Format

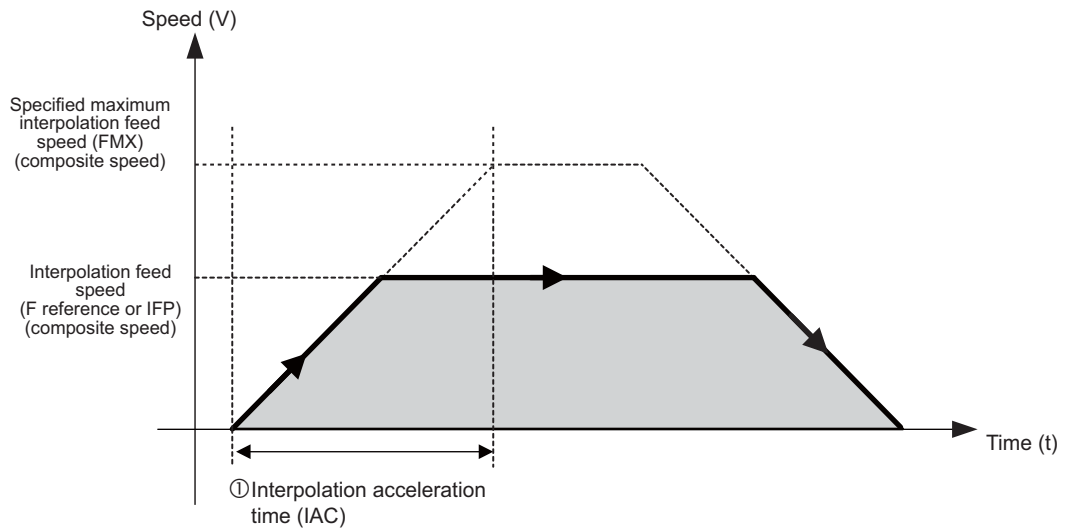
The format of the IAC instruction is as follows:

```
IAC Tinterpolation_acceleration_time;
```

Item	Unit	Applicable Data
Interpolation acceleration time	ms or reference units/s ² (specified with IUT instruction)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the IAC Instruction

This section describes the settings for the IAC instruction.



① Interpolation acceleration time

The interpolation acceleration time is set by specifying a register or a numerical value following the character "T" in the IAC instruction.

The time set with the IAC instruction designates the amount of time required to accelerate from a speed of 0 to the maximum interpolation feed speed.

The valid range for the interpolation acceleration time is 0 to 32,767 ms.

Programming Example

A programming example that uses the IAC instruction is given below.

INC;	"Incremental Mode
FMX T300000;	"Set maximum interpolation feed speed (reference units/min).
IDC T4000;	"Change interpolation deceleration time (ms).
IAC T2000;	"Change interpolation acceleration time (ms).
MVS [A1]30000 [B1]30000 F150000;	"Linear interpolation
DL00000 = 4000;	"Interpolation acceleration time (ms)
IAC TDL00000;	"Change interpolation acceleration time (ms).
MVS [A1]30000 [B1]30000;	"Linear interpolation
END;	

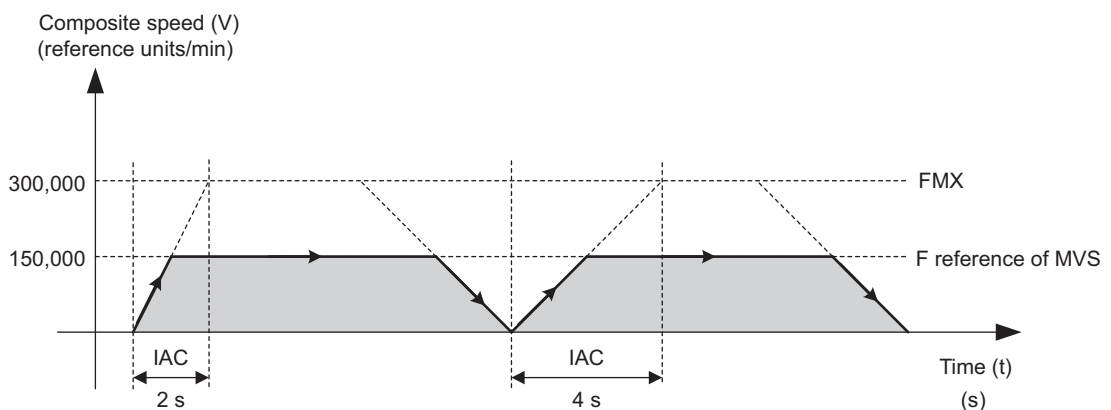


Fig. 6.22 Programming Example for IAC Instruction

Change Interpolation Deceleration Time (IDC)

The IDC instruction changes the interpolation deceleration time for the following axis movement instructions.

- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Linear Interpolation with Skip Function)

The FMX instruction must be executed first before an IDC instruction is executed. The deceleration time that is set by the IDC instruction remains in effect until it is changed by another IDC instruction.

The interpolation deceleration time is set to 0 ms when program operation starts.

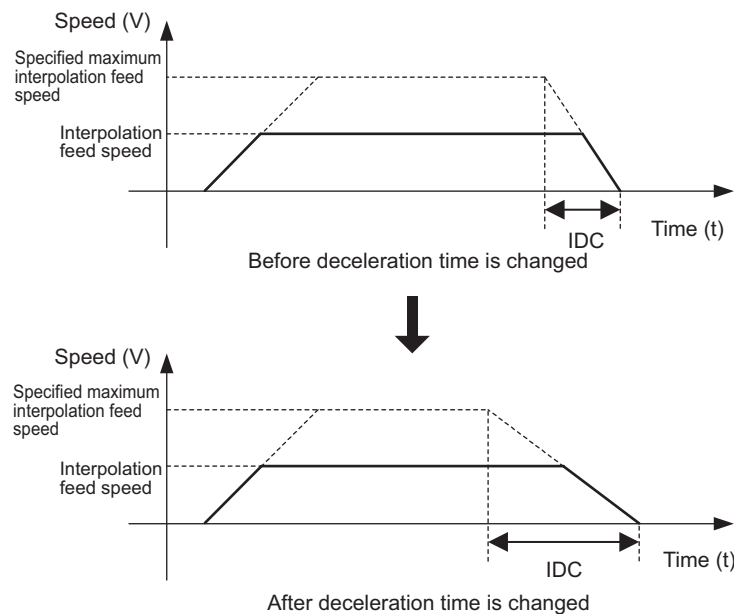


Fig. 6.23 Change Interpolation Deceleration Time

Information

1. The IDC instruction changes the deceleration time for the MVS, MCW, MCC, and SKP interpolation instructions.
Use the DCC instruction to set the deceleration time for the MOV, EXM, and MVT positioning instructions.
2. The IDC instruction is not related to any setting parameters.
The interpolation deceleration time specified by the IDC instruction is treated as control data that is reserved exclusively for motion programs. There is no setting parameter that you can use to specify the interpolation deceleration time.

Format

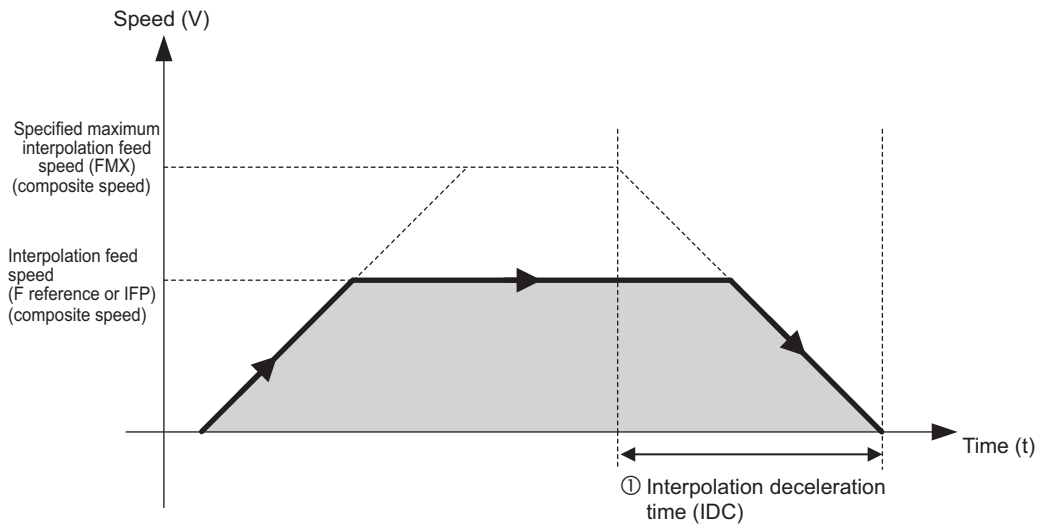
The format of the IDC instruction is as follows:

```
IDC T $interpolation\_deceleration\_time$ ;
```

Item	Unit	Applicable Data
Interpolation deceleration time	ms or reference units/s ² (specified with IUT instruction)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the IDC Instruction

This section describes the settings for the IDC instruction.



① Interpolation Deceleration Time

The interpolation deceleration time is set by specifying a register or a numerical value following the character "T" in the IDC instruction.

The time set with the IDC instruction designates the amount of time required to decelerate from the maximum interpolation feed speed to a speed of 0.

The valid range for the interpolation deceleration time is 0 to 32,767 ms.

Programming Example

A programming example that uses the IDC instruction is given below.

INC;	"Incremental Mode
FMX T300000;	"Set maximum interpolation feed speed (reference units/min).
IAC T4000;	"Change interpolation acceleration time (ms).
IDC T2000;	"Change interpolation deceleration time (ms).
MVS [A1]30000 [B1]30000 F150000;	"Linear interpolation
DL00000 = 4000;	"Interpolation deceleration time (ms)
IDC TDL00000;	"Change interpolation deceleration time (ms).
MVS [A1]30000 [B1]30000;	"Linear interpolation
END;	

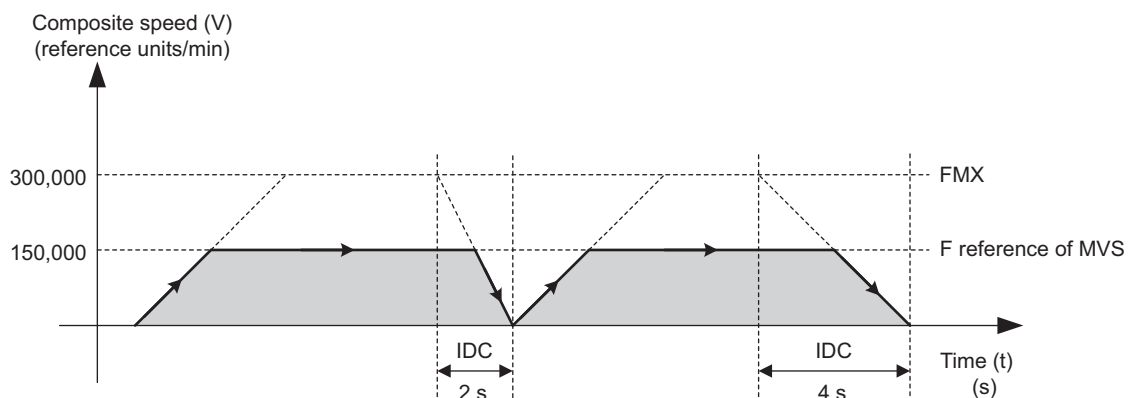


Fig. 6.24 Programming Example for IDC Instruction

Change Interpolation Deceleration Time for Temporary Stop (IDH)

The IDH instruction changes the interpolation deceleration time for temporary stop for the following axis movement instructions.

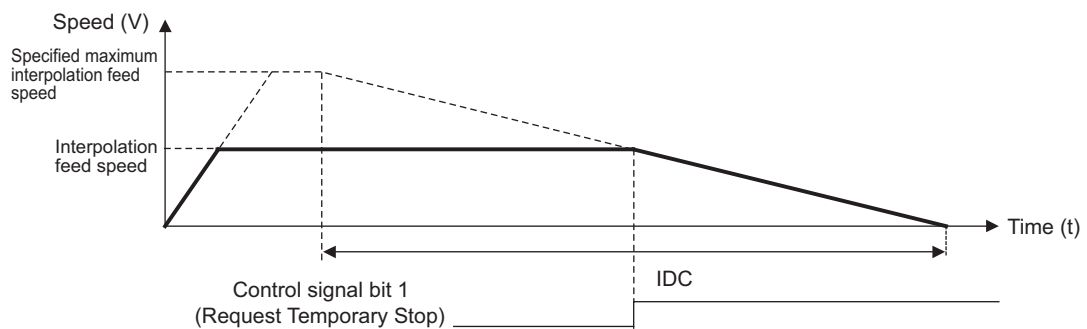
- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Linear Interpolation with Skip Function)

Use the IDH instruction when you want the axes to rapidly decelerate to a stop faster than the deceleration time specified by the IDC instruction.

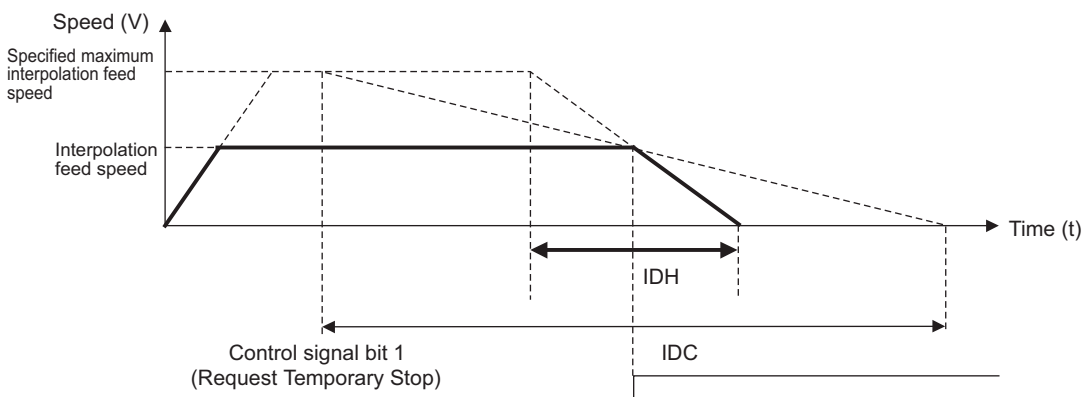
The FMX instruction must be used first to set the maximum interpolation feed speed before an IDH instruction is executed.

The deceleration time that is set by the IDH instruction remains in effect until it is changed by another IDH instruction.

If the IDH instruction is not used, the deceleration time set by the IDC instruction is used.



Before interpolation deceleration time for temporary stop is set



After interpolation deceleration time for temporary stop is set

Fig. 6.25 Change Interpolation Deceleration Time for Temporary Stop

Format

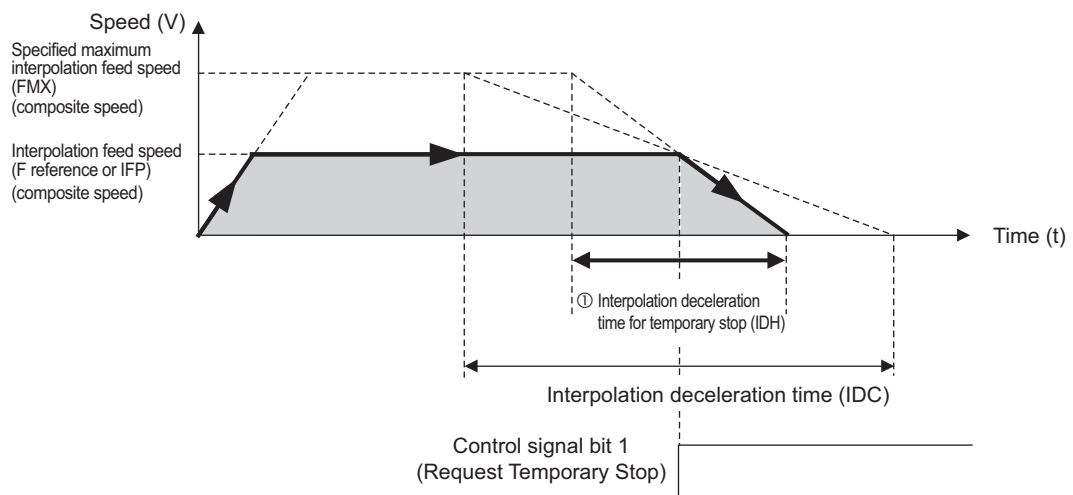
The format of the IDH instruction is as follows:

IDH T*interpolation_deceleration_time_for_temporary_stop*;

Item	Unit	Applicable Data
Interpolation deceleration time for temporary stop	ms or reference units/s ² (specified with IUT instruction)	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Settings for the IDH Instruction

This section describes the settings for the IDH instruction.



① Interpolation Deceleration Time for Temporary Stop

The interpolation deceleration time for temporary stop is set by specifying a register or a numerical value following the character “T” in the IDH instruction.

The time set with the IDH instruction designates the amount of time required to decelerate from the maximum interpolation feed speed to a speed of 0.

The valid range for the interpolation deceleration time for temporary stop is 0 to 32,767 ms.

Programming Example

A programming example that uses the IDH instruction is given below.

```

INC;                               "Incremental Mode
FMX T300000;                       "Set maximum interpolation feed speed (reference units/min).
IAC T2000;                          "Change interpolation acceleration time (ms).
IDC T4000;                          "Change interpolation deceleration time (ms).
IDH T100;                          "Set the interpolation deceleration time for temporary stop (ms).
MVS [A1]30000 [B1]30000 F150000;    "Linear interpolation (request temporary stop during axis operation)
END;

```

Change Interpolation Deceleration Time for Temporary Stop (IDH)

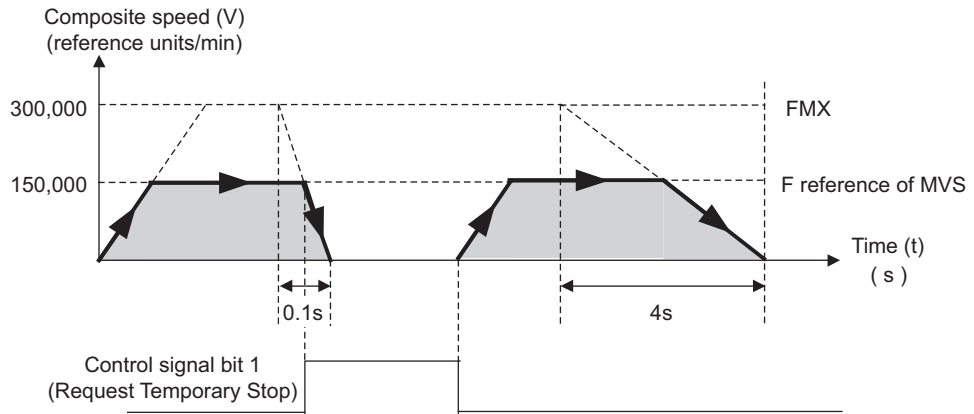


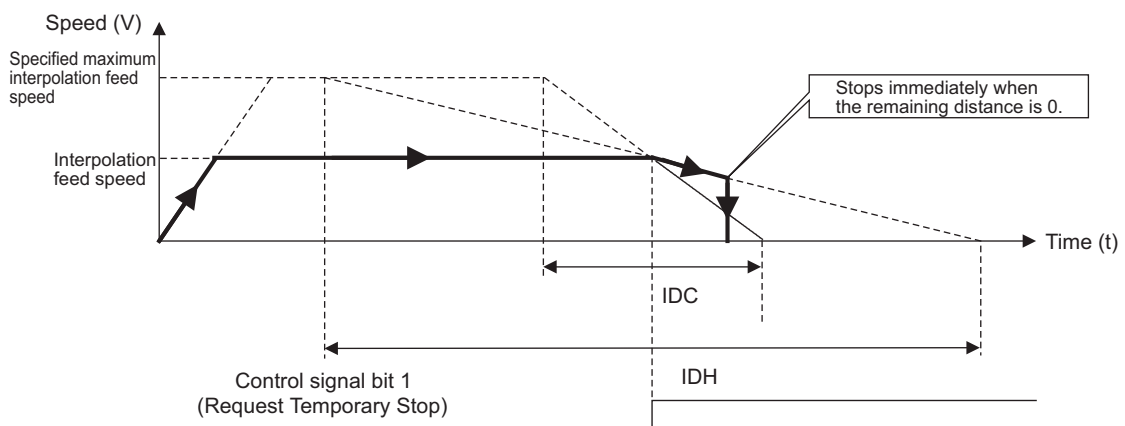
Fig. 6.26 Programming Example for IDH Instruction

Additional Information on the IDH Instruction

◆ Operation When the Deceleration Time Specified by the IDH Instruction Is Greater than the Deceleration Time Specified by the IDC Instruction

If the deceleration time specified by the IDH instruction is greater than the deceleration time specified by the IDC instruction, the remaining travel distance for the interpolation instructions may be less than the travel distance required to decelerate to a stop in the specified deceleration time.

If the remaining travel distance is less than the distance required to decelerate to a stop, the axis will stop immediately when the remaining distance equals 0.



◆ Operation When a Skip Signal Is Input

If a skip signal is input after the deceleration time is set by an IDH instruction while execution of an SKP instruction is in progress, the deceleration time set by the IDH instruction is used.

◆ Operation When the Acceleration/Deceleration Mode Is Set

The operation when a temporary stop request is made while execution of an interpolation instruction is in progress after the acceleration/deceleration mode was set with the ACCMODE instruction described below.

■ Temporary Stop Request before the Interpolation Distribution for the Next Block Begins

The axis decelerates to a stop in the deceleration time specified by the IDH instruction.

Even if the remaining travel distance for the previous block reaches 0, the distribution for the interpolation instructions in the next block has not started yet and therefore no interpolation between the blocks occurs.

■ Temporary Stop Request after the Interpolation Distribution for the Next Block Begins

Both the previous block and the next block will use the deceleration time set with the IDH instruction.

After the temporary stop request is removed, distribution of the remaining distance is performed for both the previous block and the next block.

Change Interpolation Acceleration/Deceleration Unit (IUT)

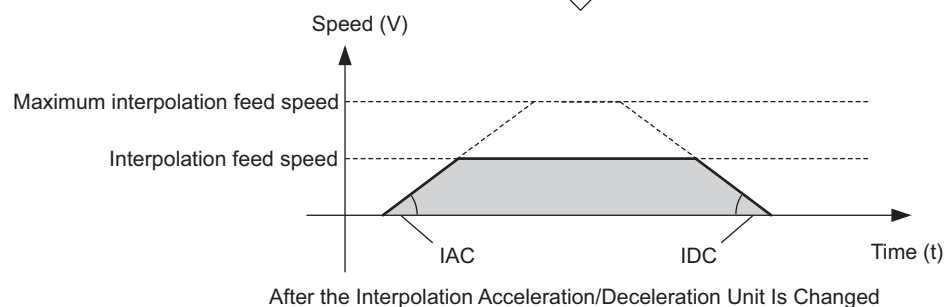
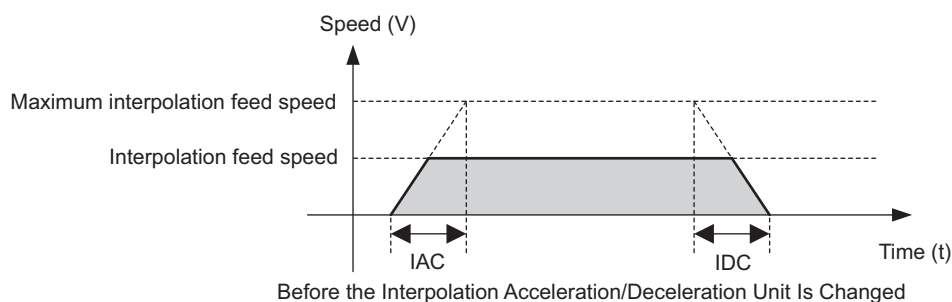
The IUT instruction can be used to change the acceleration/deceleration unit for interpolation instructions (MVS, SKP, MCW, and MCC).

The unit set with the IUT instruction is used for the following instructions.

- Change Interpolation Acceleration Time (IAC)
- Change Interpolation Deceleration Time (IDC)
- Change Interpolation Deceleration Time for Temporary Stop (IDH)

The interpolation acceleration/deceleration unit that is selected is retained until it is set again with the IUT instruction.

The interpolation acceleration/deceleration time unit is set to milliseconds when program operation starts.



Information

1. If the IUT instruction has not been executed, the interpolation acceleration/deceleration unit is milliseconds.
2. If the IUT instruction is set out of range, a compiler error will occur.
3. You can use the IUT instruction with the following versions.

Machine Controller or MPE720	Applicable Versions
MP3000-series Machine Controller	Ver. 1.08 or later
MPE720 Version 7	Version 7.23 or later

Format

The format of the IUT instruction is as follows:

IUT Interpolation acceleration/deceleration unit number:

Item	Unit	Applicable Data
Interpolation acceleration/deceleration unit number	–	Directly designated value 0: ms (default) 1: Reference units/s ²



Important

- When the IUT instruction is executed to change the interpolation acceleration/deceleration unit, the most gradual acceleration/deceleration is set to ensure safety. After you change the unit, set the interpolation acceleration/deceleration rates according to the new unit.

IUT Set Value	Set Value for IAC, IDC, or IDH
Changed from U0 to U1	1 (reference units/s ²)
Changed from U1 to U0	32,767 (ms)

- The setting ranges of the IAC, IDC, and IDH instructions depend on the acceleration/deceleration unit.

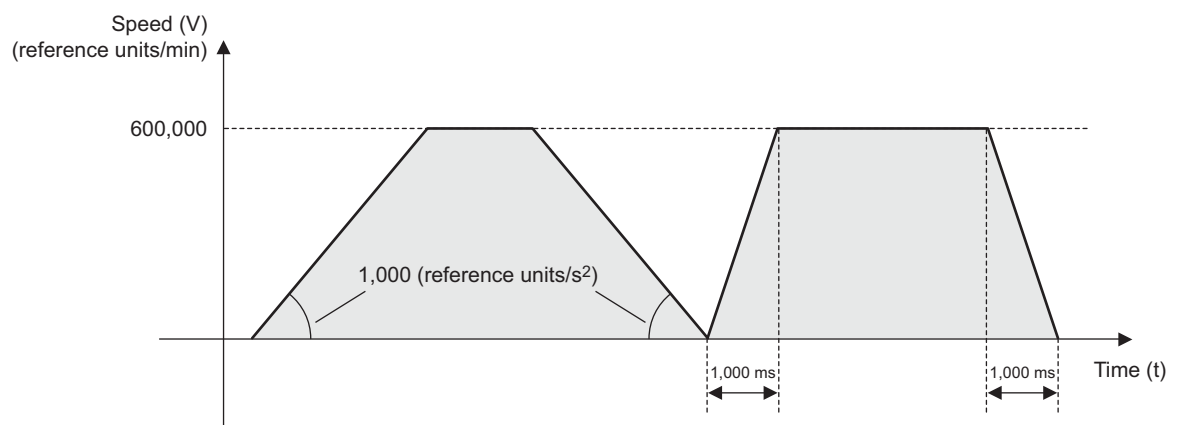
IUT Set Value	Set Value for IAC, IDC, or IDH
U0	0 to 32,767 (ms)
U1	1 to 2,147,483,647 (reference units/s ²)

Programming Example

A programming example that uses the IUT instruction is given below.

```

INC;                "Incremental Mode"
FMX T600000;       "Maximum interpolation feed speed"
IUT U1;            "Change interpolation acceleration/deceleration unit from ms to reference units/s2."
IAC T1000;         "Acceleration rate = 1,000 reference units/s2"
IDC T1000;         "Deceleration rate = 1,000 reference units/s2"
MVS [A1]1000000 F600000; "MVS ①"
IUT U0;            "Change interpolation acceleration/deceleration unit from reference units/s2 to ms."
IAC T1000;         "Acceleration time = 1,000 ms"
IDC T1000;         "Deceleration time = 1,000 ms"
MVS [A1]1000000 F600000; "MVS ②"
END;
  
```



Set Interpolation Feed Speed Axes (+ and -)

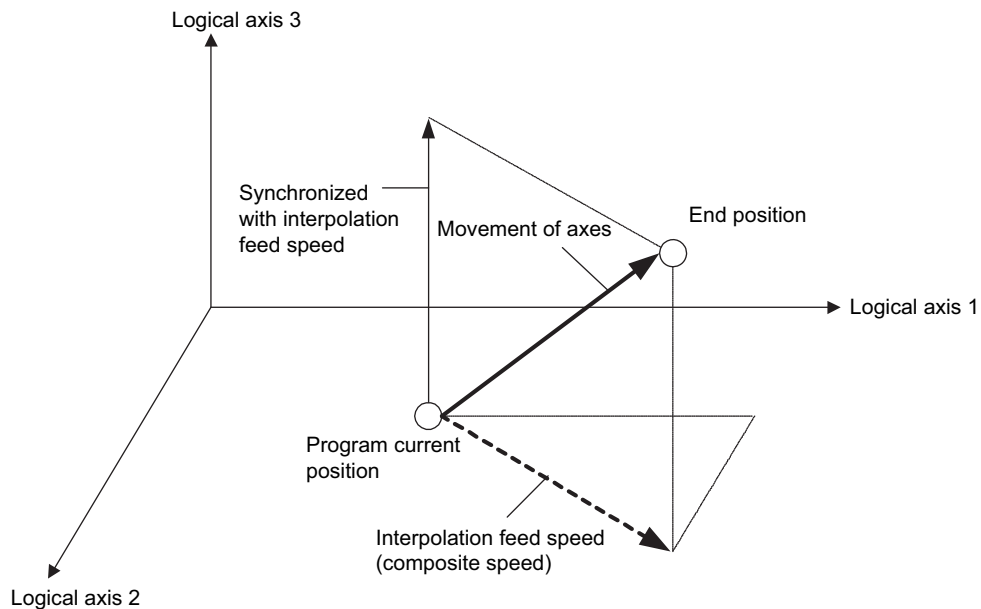
The Set Interpolation Feed Speed Axes (+ and -) instructions allow you to arbitrarily set the axes to use as component axes for the interpolation feed speed.

These instructions can be used for the MVS, SKP, MCW Helical, and MCC Helical interpolation instructions.

If “+” is given or nothing is given before the logical axis name, the axis is one of the component axes for the interpolation feed speed.

If “-” is given before the logical axis name, the axis operates at a speed that is synchronized with the interpolation feed speed.

The following figure shows the linear operation for three axes when logical axes 1 and 2 are set as interpolation feed speed axes (i.e., with “+”) and logical axis 3 is not set as an interpolation feed speed axis (i.e., with “-”).



Information

Normally when you need to maintain a constant speed for a specific axis when performing an interpolated movement for more than one axis, you must calculate the composite interpolation feed speed.

However, you can use these instructions to perform synchronized control for a constant speed of a specific axis without calculating the composite speed.

The master axes (+) operate at the specified interpolation feed speed and the slave axes (-) operate in synchronization with the master axes and at a speed that corresponds to the travel distances of the slave axes.

Format

The format of the Set Interpolation Feed Speed Axes instructions is as follows:

- Format in the MVS Instruction
MVS [(+)Logical_axis_name_1] Reference_position [(+)Logical_axis_name_2] Reference_position [-Logical_axis_name_3] Reference_position . . . Finterpolation_feed_speed;
- Format in the SKP Instruction
SKP [(+)Logical_axis_name_1] Reference_position [(+)Logical_axis_name_2] Reference_position [-Logical_axis_name_3] Reference_position . . . Finterpolation_feed_speed Sskip_input_signal_selection;
- Format in the MCW or MCC Helical Interpolation with Specified Center Point Instruction
MCW (or MCC) [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Ucenter_point_position Vcenter_point_position [-Logical_axis_name_3] End_position_for_linear_interpolation Tnumber_of_turns Finterpolation_feed_speed;
- Format in the MCW or MCC Helical Interpolation with Specified Radius Instruction
MCW (or MCC) [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Rradius [-Logical_axis_name_3] End_position_for_linear_interpolation Finterpolation_feed_speed;

Item	Unit	Applicable Data
Reference position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register
Interpolation feed speed	Reference units/min	
Skip input signal selection	–	
End position	Reference units	
Center point position	Reference units	
Linear interpolation end position	Reference units	
Number of turns	Number of turns	
Radius	Reference units	



Important

Precautions for the MVS and SKP Instructions

- If you add “+” before all of the logical axis names, the normal interpolation operation will be performed.
- If you add “-” before all of the logical axis names, a compiler error will occur.
- Depending on the settings, the speed of an axis that is not set as an interpolation feed speed axis may exceed the set value of the FMX instruction. To ensure safety, set the maximum speed for such an axis with the IFMX instruction before you use the axis.
- If the composite travel distance for the axes that are set as interpolation feed speed axes is 0, a motion program alarm will occur and the axes will not operate.



Important

Precautions for the MCW and MCC Helical Interpolation Instructions

- If you add “+” before all of the logical axis names of the linear interpolation axes for an MCW or MCC Helical Interpolation instruction, the normal helical interpolation operation will be performed.
- If you add “+” or “-” before the logical axis names of the circular interpolation axes, a compiler error will occur.
- The MCW and MCC Circular Interpolation instructions (with specified center points or specified radii) do not support this function.
- Depending on the settings, the speed of an axis that is not set as an interpolation feed speed axis may exceed the set value of the FMX instruction. To ensure safety, set the maximum speed for such an axis with the IFMX instruction before you use the axis.
- If the composite travel distance for the axes that are set as interpolation feed speed axes is 0, a motion program alarm will occur and the axes will not operate.

Programming Example

Programming examples that use the Set Interpolation Feed Speed Axes instructions are given below.

◆ Specification with the MVS Instruction

```

INC;
FMX T1000000;
IAC T100;
IDC T100;
MVS [+A1]10000 [-B1]20000 [-C1]30000 F1000000;
END;
    
```

"Incremental Mode"
 "Set maximum interpolation feed speed."
 "Interpolation acceleration time = 100 ms"
 "Interpolation deceleration time = 100 ms"
 "Linear interpolation with specification of interpolation feed speed axes"

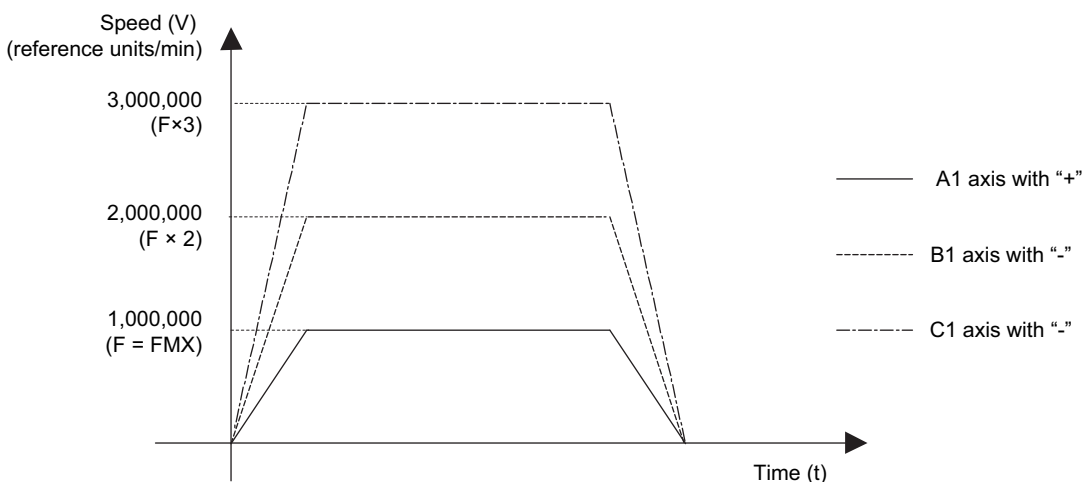


Fig. 6.27 Programming Example of MVS Instruction with Specification of Interpolation Feed Speed Axes

◆ Specification with the MCW and MCC Helical Interpolation Instructions

```

ABS;
FMX T30000000;
PLN [A1][B1];
MCC [A1]1000 [B1]0 R1000 [-C1]500 F2000;
END;
    
```

"Absolute Mode"
 "Set maximum interpolation feed speed."
 "Coordinate plane setting"
 "Helical interpolation with specification of interpolation feed speed axes"

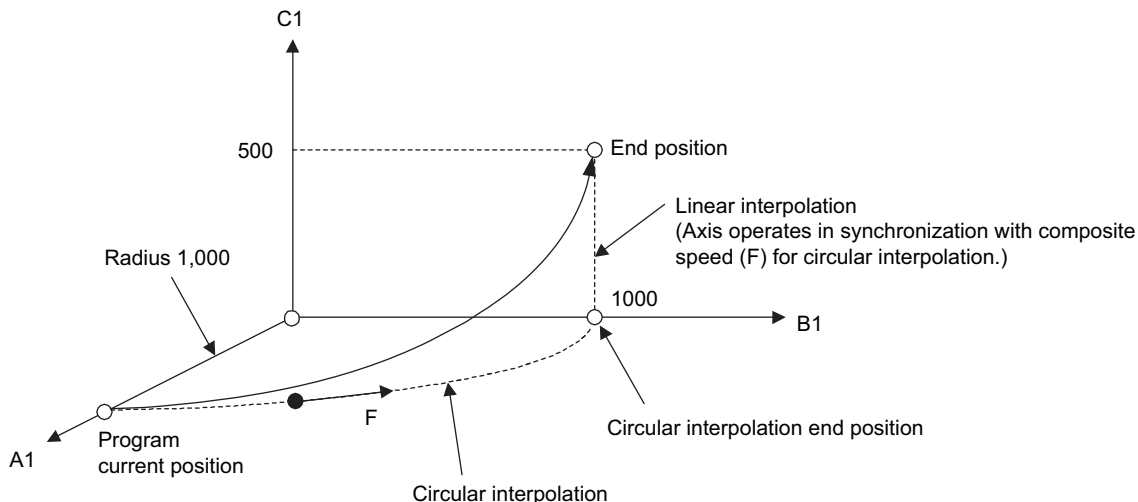


Fig. 6.28 Programming Example of MCW or MCC Helical Interpolation Instruction with Specification of Interpolation Feed Speed Axes

Set Interpolation Acceleration/Deceleration Mode (ACCMODE)

The ACCMODE instruction sets the acceleration/deceleration mode for the following interpolation instructions. You can use the ACCMODE instruction to connect the speeds between continuous interpolation instructions.

- MVS (Linear Interpolation)
- MCC or MCW (Circular Interpolation)
- MCC or MCW (Helical Interpolation)
- SKP (Linear Interpolation with Skip Function)

The interpolation acceleration/deceleration mode set by the ACCMODE instruction remains in effect until it is changed by another ACCMODE instruction.

The interpolation acceleration/deceleration mode is set to the default mode (interpolation acceleration/deceleration mode 0) when program operation starts.

Information

1. The interpolation acceleration/deceleration mode cannot be changed between continuous interpolation blocks.
Change the interpolation acceleration/deceleration mode only after the axes decelerate to a stop.
2. If the interpolation acceleration/deceleration mode is set out of range, the operation depends on the version of the CPU Unit/CPU Module.

Software Version	MPE720 Version 7.24 or Later	MPE720 Version 7.23 or Earlier
CPU Unit/Module Version 1.09 or Later	A motion program alarm (31 hex: Address M out of range) will occur when an interpolation instruction is executed.	A motion program alarm (31 hex: Not registered) will occur when an interpolation instruction is executed.
CPU Unit/Module Version 1.08 or Earlier	An alarm will not occur even when an interpolation instruction is executed. The current interpolation acceleration/deceleration mode will be retained.	

3. When the PFORK instruction is used, the interpolation acceleration/deceleration mode setting before branching to the forks is inherited by all of the forks. After branching, you can set the interpolation acceleration/deceleration mode for each fork independently.

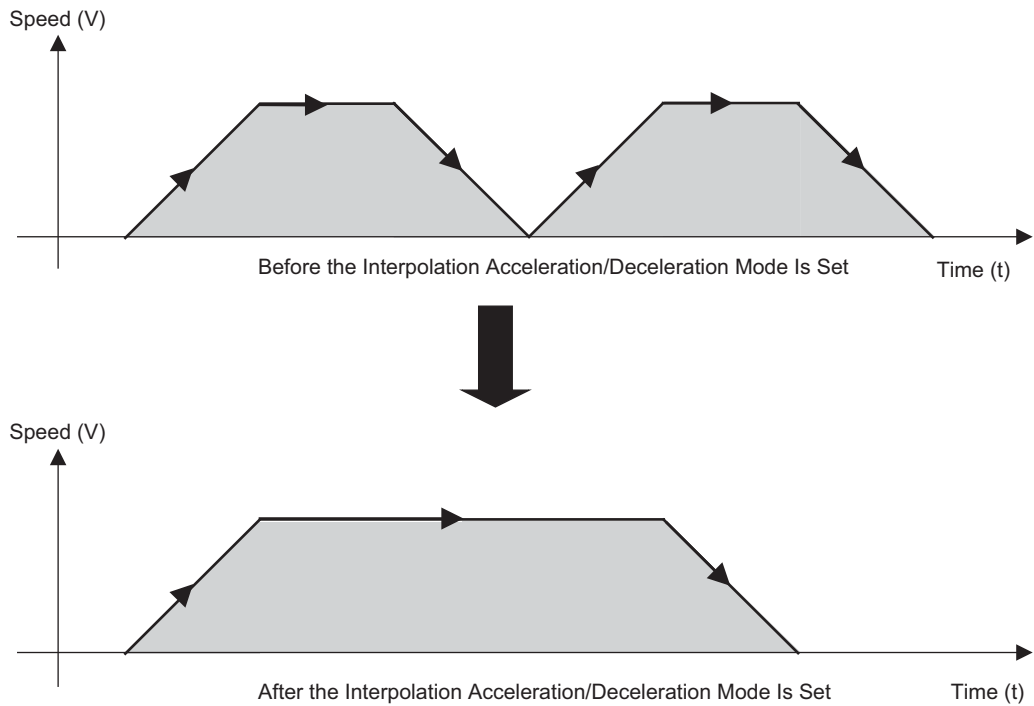


Fig. 6.29 Set Interpolation Acceleration/Deceleration Mode

Format

The format of the ACCMODE instruction is as follows:

ACCMODE *Minterpolation_acceleration_deceleration_mode*;

Item	Unit	Applicable Data
Interpolation acceleration/deceleration mode	–	Directly designated number (0 to 4)

Settings for the ACCMODE Instruction

This section describes the settings for the ACCMODE instruction.

The interpolation acceleration/deceleration mode is set by specifying a numerical value following the character “M” in the ACCMODE instruction.

There are five interpolation acceleration/deceleration modes.

- Interpolation acceleration/deceleration mode 0 (default mode)
- Interpolation acceleration/deceleration mode 1 (acceleration/deceleration mode with continuous process control signal monitoring)
- Interpolation acceleration/deceleration mode 2 (acceleration/deceleration mode with interpolation overlapping)
- Interpolation acceleration/deceleration mode 3 with continuous deceleration for minute blocks (acceleration/deceleration mode with continuous process control signal monitoring)
- Interpolation acceleration/deceleration mode 4 (acceleration/deceleration mode with next block speed specification)

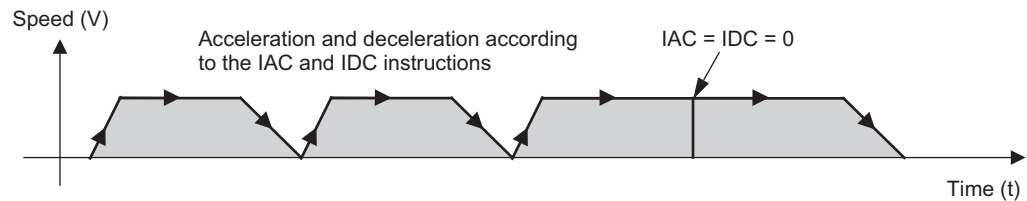
ACCMODE Details

This section describes the five interpolation acceleration/deceleration modes of the ACCMODE instruction.

◆ Interpolation Acceleration/Deceleration Mode 0 (Default Mode) Details

In this mode, acceleration and deceleration are performed according to the acceleration/deceleration times set with the IAC and IDC instructions.

This is the default mode when program operation starts.



■ Format

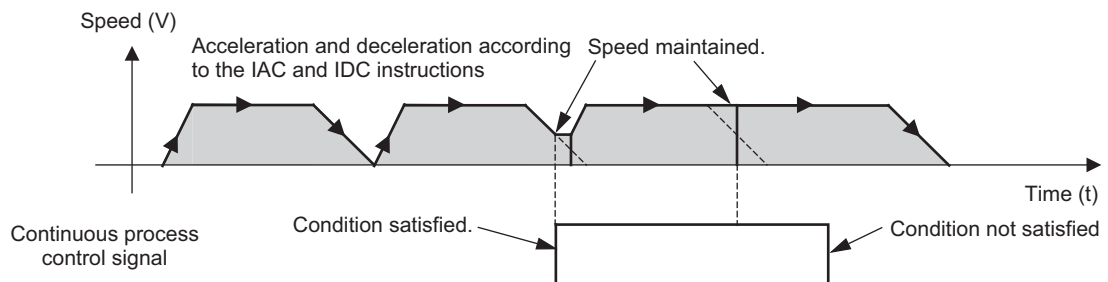
Use the following code format to select interpolation acceleration/deceleration mode 0.

```
ACCMODE M0;
```

◆ Interpolation Acceleration/Deceleration Mode 1 (Acceleration/Deceleration Mode with Continuous Process Control Signal Monitoring) Details

This mode monitors a continuous process control signal and performs continuous processing between continuous interpolation blocks when the specified conditions are satisfied.

This mode can be used only when the same axes are used for all continuous interpolation blocks.



■ Format

Use the following format to select interpolation acceleration/deceleration mode 1.

```
ACCMODE M1;
```

```
MVS [Logical_axis_name_1] Reference_position Finterpolation_feed_speed TWcontinuous_process_control_signal;
```

Or

```
ACCMODE M1;
```

```
MVS [Logical_axis_name_1] Reference_position Finterpolation_feed_speed FWcontinuous_process_control_signal;
```

Item	Unit	Applicable Data
Continuous process control signal	–	All bit data registers (excluding #, C, and D registers)

Note: The format is the same for the MCC, MCW, and SKP instructions.

If the characters “TW” or “FW” are added to the interpolation instruction, continuous process control signal monitoring is performed. The bit data register specified with the characters “TW” or “FW” is used as the continuous process control signal.

If the characters “TW” or “FW” are not added to the interpolation instruction, or if the conditions are not satisfied, the continuous process control signal is not monitored and acceleration/deceleration is performed according to the acceleration/deceleration times set with the IAC and IDC instructions.

Set Interpolation Acceleration/Deceleration Mode (ACCMODE)

Information

The characters "TW" and "FW" are valid only for interpolation acceleration/deceleration modes 1 and 3 (acceleration/deceleration modes with continuous process control signal monitoring). In other modes, the operation depends on the software version of the CPU Unit/CPU Module.

Software Version	MPE720 Version 7.24 or Later	MPE720 Version 7.23 or Earlier
CPU Unit/Module Version 1.09 or Later	A motion program alarm (32 hex: Specified address error) will occur when an interpolation instruction is executed.	A motion program alarm (32 hex: Not registered) will occur when an interpolation instruction is executed.
CPU Unit/Module Version 1.08 or Earlier	An alarm will not occur even when an interpolation instruction is executed. The TW or FW address is ignored and the interpolation instruction is executed.	

The characters "TW" designate monitoring the continuous process control signal with positive logic.

Continuous process control signal	Operation Summary
ON	The deceleration time specified with the IDC instruction is ignored. The current speed is maintained and pulse distribution is completed with a deceleration time of 0 ms.
OFF	The axis decelerates to a stop according to the deceleration time specified with the IDC instruction.

The characters "FW" designate monitoring the continuous process control signal with negative logic.

Continuous process control signal	Operation Summary
ON	The axis decelerates to a stop according to the deceleration time specified with the IDC instruction.
OFF	The deceleration time specified with the IDC instruction is ignored. The current speed is maintained and pulse distribution is completed with a deceleration time of 0 ms.



Important

If you specify a travel distance that is insufficient to perform continuous processing with the set deceleration time, unexpected operation may occur. Also specify a sufficient travel distance.

■ Programming Examples

The following example programming uses interpolation acceleration/deceleration mode 1 (acceleration/deceleration mode with continuous process control signal monitoring).

```
FMX T30000000;
```

```
ABS;
```

```
IAC T1000;
```

```
IDC T1000;
```

```
ACCMODE M1;
```

```
MVS [A1] 4000 F50000 TWMB000001;    "①"
```

```
IOW MB000001=1;                      "②"
```

```
MVS [A1] 8000;                         "③"
```

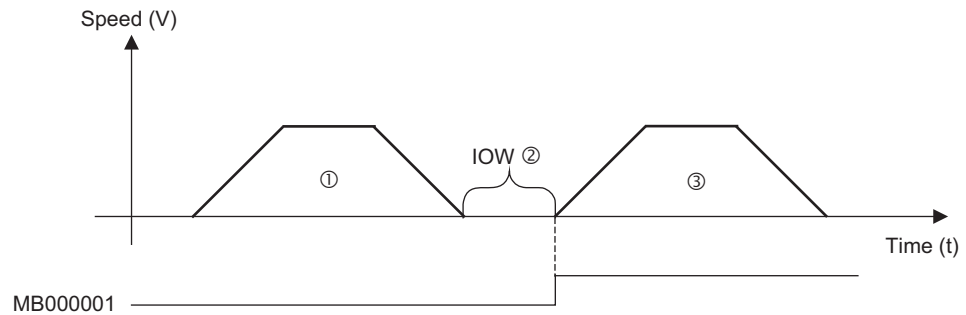
```
END;
```


The following examples show how to combine the MVS instruction and interpolation acceleration/deceleration mode 1.

- When the Continuous Process Control Signal Turns ON after Distribution for MVS Instruction ① Is Completed

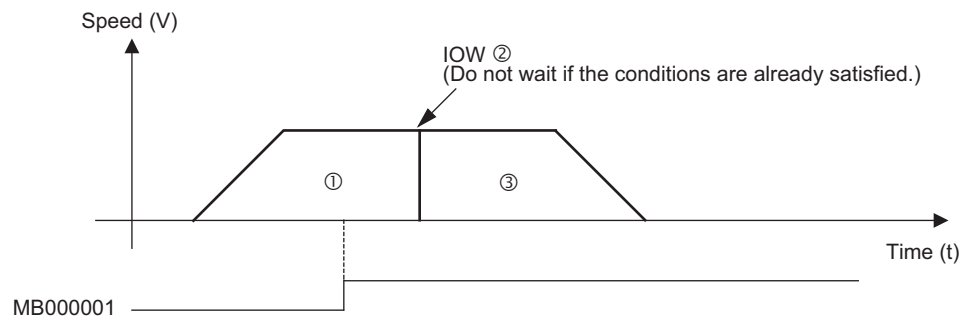
① Is Completed
The next block is executed after the axis decelerates to a stop for the MVS instruction ①.

For the MVS instruction ③, acceleration begins when the speed is 0 (reference units/min).



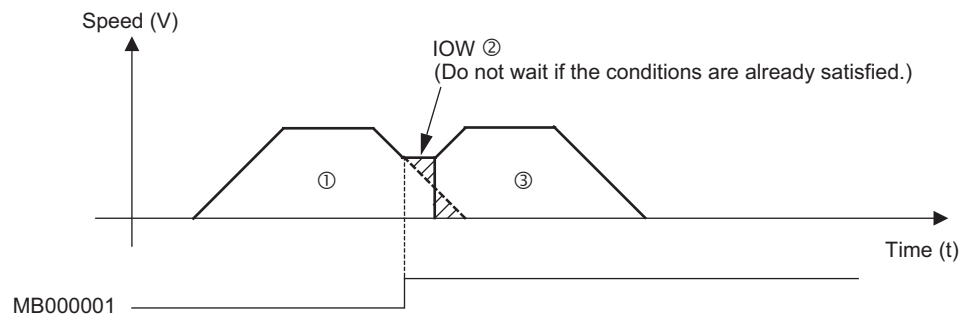
- When the Continuous Process Control Signal Turns ON during Distribution for MVS ① (before Deceleration)

MVS ③ is executed at the same speed from MVS ① without decelerating.



- When the Continuous Process Control Signal Turns ON during Distribution for MVS ① (during Deceleration)

MVS ③ is executed with the same speed as when the continuous process control signal turned ON.



Information

1. If the reference speed for MVS ③ is higher than for MVS ①, the end speed of ① is used for the start speed of ③. The axis then accelerates to the specified speed.
2. If the reference speed for MVS ③ is lower than for MVS ①, the end speed of ① is used for the start speed of ③. The axis then decelerates to the specified speed.
3. If the travel distance for MVS ③ is shorter than the deceleration distance, distribution is finished during the deceleration of ③.

■ Additional Information

Refer to the additional information below for details on operation in the acceleration/deceleration mode with continuous process control signal monitoring.

• Request Temporary Stop Operation

Temporary Stop Request before the Interpolation Distribution for the Next Block Begins

The axis decelerates according to the interpolation deceleration time specified with the IDH instruction. No continuous processing to the next interpolation block is performed.

Temporary Stop Request after the Interpolation Distribution for the Next Block Begins

The axis decelerates according to interpolation deceleration time specified with the IDH instruction for both the previous block and the next block.

After the temporary stop request is removed, distribution of the remaining distance is performed for both the previous block and the next block.

• Request Stop Operation

The interpolation block for the axis in motion stops immediately.

• Program Single-block Mode Operation

No continuous processing to the next interpolation block is performed.

• Debug Mode Operation

No continuous processing to the next interpolation block is performed.

• Operation When the Next Block Is Not an Interpolation Instruction Block

No continuous processing to the next block is performed.

Acceleration begins from a stopped state for the next block.

• Operation When the Interpolation Deceleration Time (IDC) Is Set to 0 ms

Continuous processing to the next interpolation block is performed, regardless of the status of the continuous process control signal.

• Continuous Operation during Parallel Execution (PFORK)

Continuous processing is not performed across a PFORK instruction.

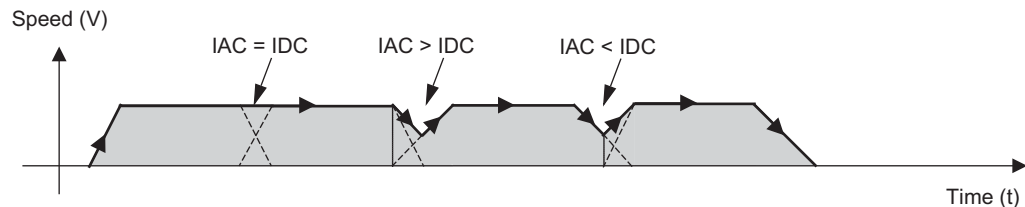
Set the instructions so that processing for this mode ends during each fork.

◆ Interpolation Acceleration/Deceleration Mode 2 (Acceleration/Deceleration Mode With Interpolation Overlapping) Details

In this mode, pulse distribution for each interpolation block is made to overlap by starting acceleration for the next interpolation block to perform continuous processing between consecutive interpolation blocks.

Each block accelerates and decelerates according to the acceleration times and deceleration times that are set with the IAC and IDC instructions.

This mode is valid for the MVS, MCW, and MCC instructions.



■ Format

ACCMODE M2;

MVS [*Logical_axis_name_1*] *Reference_position* *Finterpolation_feed_speed* *Dinterpolation_overlap_distance*;

Item	Unit	Applicable Data
Interpolation overlap distance	Reference units	<ul style="list-style-type: none"> Directly designated value Indirect designation with a double-length integer register

Note: The interpolation overlap distance can be omitted.

The format is the same for the MCC and MCW instructions.

In this mode, you can add the character "D" to an interpolation instruction to specify the maximum distance for the interpolation distribution to overlap.

When the character "D" is added to an interpolation instruction in this mode, distribution for the next interpolation block begins when the remaining travel distance for the current interpolation block falls below the interpolation overlap distance. If 0 (reference units) is specified for the interpolation overlap distance, distribution for the next interpolation block begins when the current interpolation block begins deceleration.

If the character "D" is not specified for the interpolation instruction, the last interpolation overlap distance that was specified in the motion program is used.

The interpolation overlap distance is set to 0 (reference units) when program operation starts.

Information

- The character "D" is valid only for this mode.
In other interpolation acceleration/deceleration modes, the operation depends on the version of the CPU Unit/CPU Module.

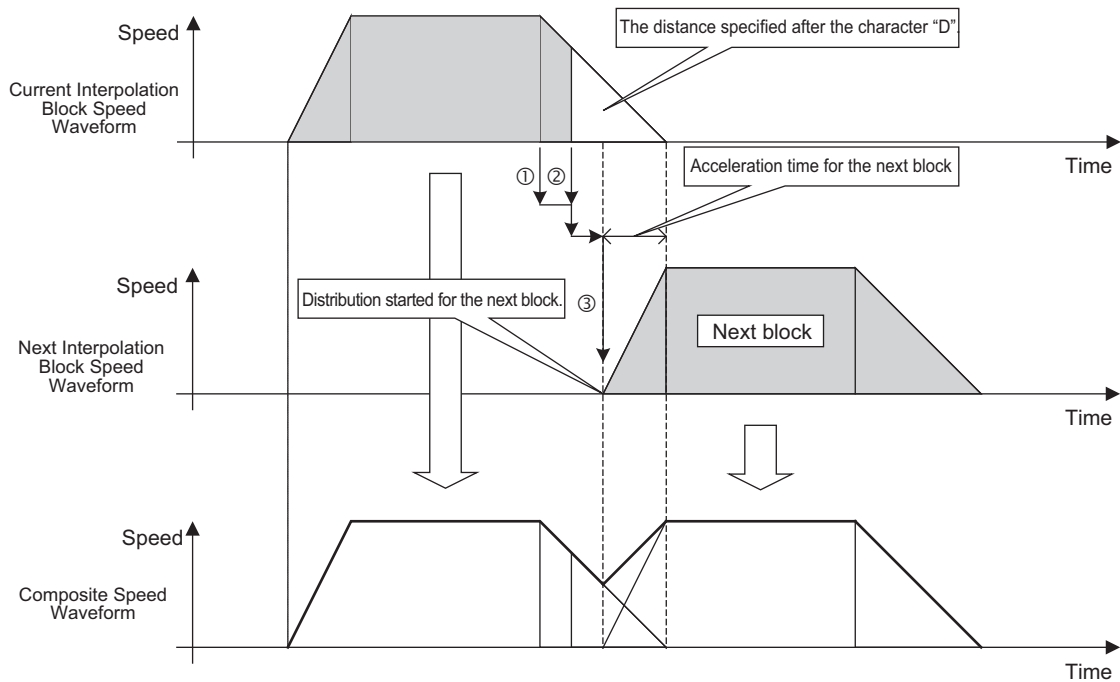
Software Version	MPE720 Version 7.24 or Later	MPE720 Version 7.23 or Earlier
CPU Unit/Module Version 1.09 or Later	A motion program alarm (32 hex: Specified address error) will occur when an interpolation instruction is executed.	A motion program alarm (32 hex: Not registered) will occur when an interpolation instruction is executed.
CPU Unit/Module Version 1.08 or Earlier	An alarm will not occur even when an interpolation instruction is executed. The D address is ignored and the interpolation instruction is executed.	

- The valid range for the interpolation overlap distance is 0 to 2,147,483,647 (reference units).
If a negative value is specified, the absolute value is used.

■ Conditions to Begin Distribution for the Next Interpolation Block

Distribution for the next interpolation block begins when all of the following conditions are satisfied.

No.	Condition
1	Not in Program Single-block Mode.
2	Control signal bit 1 (Request Temporary Stop) is OFF.
3	No PFN or PFP instructions have been added to the interpolation instructions.
4	The interpolation block must have started deceleration. (Refer to the timing of ① in the figure below.)
5	The remaining distance for the interpolation block is less than the interpolation overlap distance specified after the character "D". (Refer to the timing of ② in the figure below.)
6	The remaining deceleration time of the current interpolation block is less than the acceleration time of the next interpolation block. (Refer to the timing of ③ in the figure below.)



■ Programming Example

A programming example for the acceleration/deceleration mode with interpolation overlapping is given below.

```
FMX T300000;
```

```
INC;
```

```
IAC T1000;
```

```
IDC T2000;
```

```
ACCMODE M2;
```

```
MW00010 = 30;
```

```
MVS [A1] 20000 [B1] 10000 F200000; "Linear interpolation ①"
```

```
MW00010 = 20;
```

```
MVS [A1] 10000 [B1] -20000 D100; "Linear interpolation ②"
```

```
MW00010 = 10;
```

```
MVS [A1] 20000 [B1] 10000; "Linear interpolation ③"
```

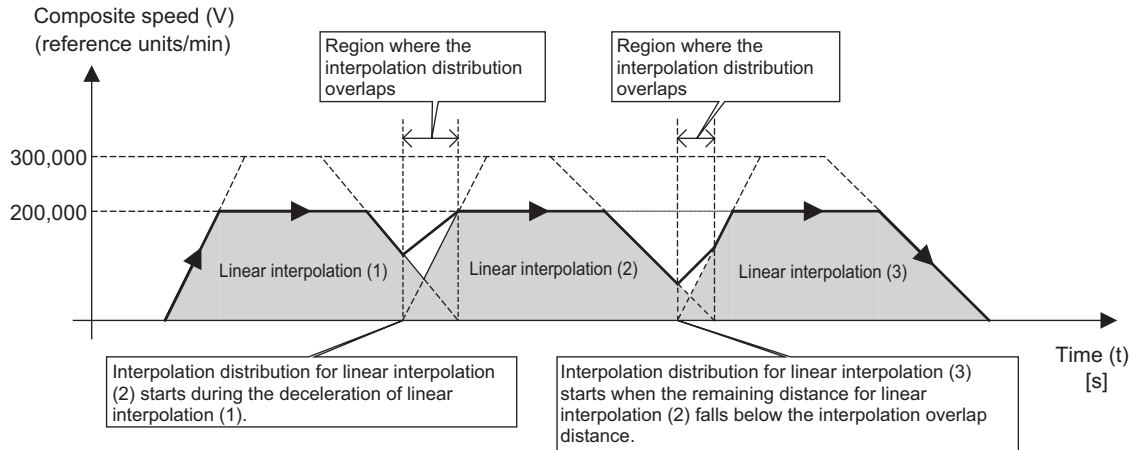
```
MW00010 = 0;
```

```
END;
```

In processing for interpolation acceleration/deceleration mode 2, execution moves to the next execution block in the program when deceleration occurs for the interpolation block or when the interpolation overlap distance becomes equal to or less than the specified interpolation overlap distance.

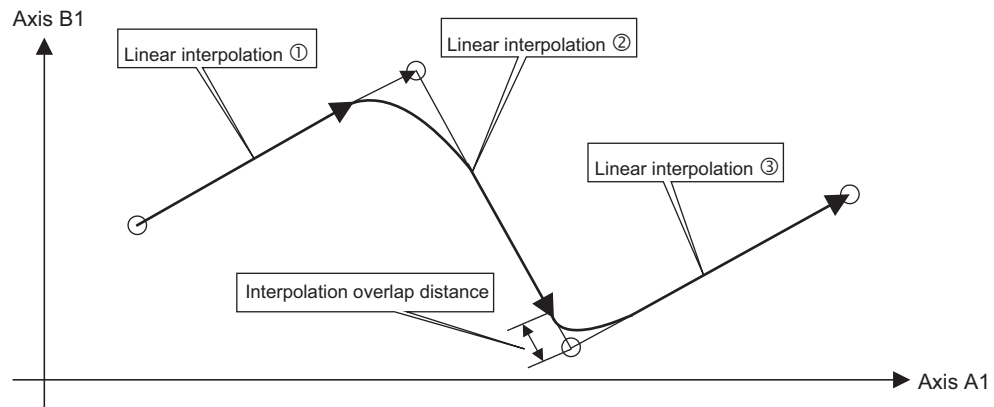
S-type instructions (e.g., operation instructions) that occur during continuous processing for interpolation blocks are executed when program execution moves to the next block.

The speed waveform for the programming example is given below.



The interpolated path for the above programming example is given below.

As shown in the figure below, some interpolation block end points (i.e., the start point for the next interpolation block) do not pass through the movement path because the distribution for the next interpolation block starts during deceleration of the current interpolation block.



■ Additional Information

Refer to the following additional information for details on operation in the acceleration/deceleration mode with interpolation overlapping.

• Request Temporary Stop Operation

Before distribution for the next interpolation block begins, the interpolation block for axes currently in motion decelerates in the deceleration time specified with the IDH instruction.

However, no continuous processing to the next interpolation block is performed.

After distribution for the next interpolation block begins, each interpolation block decelerates in the deceleration time specified with the IDH instruction.

• Request Stop Operation

Each interpolation block stops immediately.

• Program Single-block Mode Operation

No continuous processing to the next interpolation block is performed.

• Debug Mode Operation

No continuous processing to the next interpolation block is performed.

• Operation When the Next Block Is Not an Interpolation Instruction Block

No continuous processing to the next block is performed.

Acceleration begins from a speed of 0 for the next block.

• Continuous Operation during Parallel Execution (PFORK)

This mode cannot be used across a PFORK instruction.

Adjust the timing of pulse distribution for the interpolation block with the PFN or PFP instruction so that processing (i.e., pulse distribution) for this mode is completed within each fork.

• Operation for Execution of T-type Instructions

If a T-type instruction (e.g., a timer instruction) is executed in continuous processing for an interpolation block, the distribution timing in the next interpolation block will be changed.

◆ Interpolation Acceleration/Deceleration Mode 3 (Acceleration/Deceleration Mode with Continuous Process Control Signal Monitoring) Details


In the same way as in interpolation acceleration/deceleration mode 1 (acceleration/deceleration mode with continuous process control signal monitoring), interpolation acceleration/deceleration mode 3 (acceleration/deceleration mode with continuous process control signal monitoring) monitors a continuous process control signal and performs continuous processing between consecutive interpolation blocks when the specified conditions are satisfied.

However, opposed to interpolation acceleration/deceleration mode 1 (acceleration/deceleration mode with continuous process control signal monitoring), when continuous processing is performed for a minute block with a minute travel distance, deceleration is performed as much as possible to the specified speed in continuous processing between consecutive interpolation blocks.

Information A minute block is an interpolation block with a travel distance that is too small for the distance required to decelerate to a stop at the specified deceleration rate from the speed for continuous processing operation.

■ Format

Use the following format to select interpolation acceleration/deceleration mode 3 (acceleration/deceleration mode with continuous process control signal monitoring). Refer to the following section for details on continuous processing control signals.

 ◆ *Interpolation Acceleration/Deceleration Mode 1 (Acceleration/Deceleration Mode with Continuous Process Control Signal Monitoring) Details (page 6-65)*

```
ACCMODE 3;
MVS [Logical_axis_name_1] Reference_position Finterpolation_feed_speed TWcontinuous_process_control_signal;
```

Or

```
ACCMODE 3;
MVS [Logical_axis_name_1] Reference_position Finterpolation_feed_speed FWcontinuous_process_control_signal;
```

Note: The format is the same for the MCC, MCW, and SKP instructions.

Information The characters "TW" and "FW" are valid only for this mode and for interpolation acceleration/deceleration mode 1 (acceleration/deceleration mode with continuous process control signal monitoring). In other modes, the operation when the characters "TW" or "FW" are specified depends on the software version of the CPU Unit/CPU Module.

Software Version	MPE720 Version 7.24 or Later	MPE720 Version 7.23 or Earlier
CPU Unit/CPU Module Version 1.09 or Later	An alarm (32 hex: Specified address error) will occur when an interpolation instruction is executed.	An alarm (32 hex: No registered) will occur when an interpolation instruction is executed.
CPU Unit/CPU Module Version 1.08 or Earlier	An alarm will not occur even when an interpolation instruction is executed. The TW or FW address is ignored and the interpolation instruction is executed.	

■ Programming Example

The difference between interpolation acceleration/deceleration modes 1 and 2 for the MVS instruction is described below.

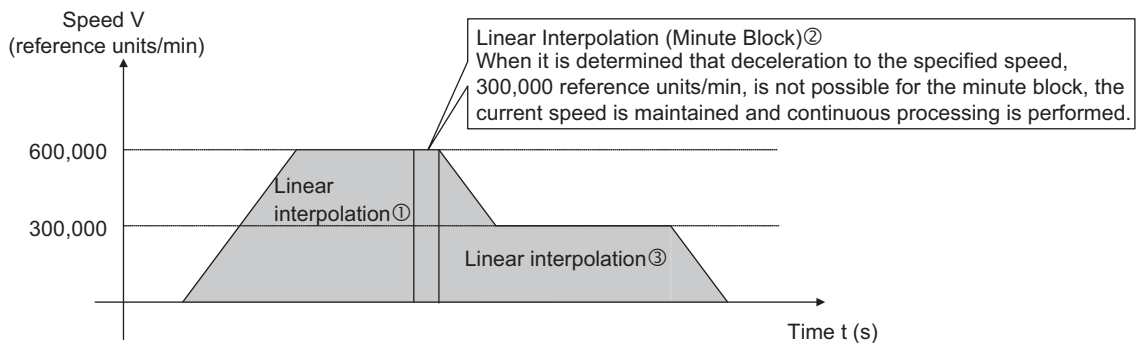
```

INC;
FMX T1000000;
IAC T5000;
IDC T5000;
MB1000=1; // Continuous control signal bit
//Interpolation acceleration/deceleration mode (ACCMODE M1 or ACCMODE M3 executed.)
ACCMODE M1; // ACCMODE M1 or ACCMODE M3
MVS [A1]100000 F600000 TWMB1000; // Linear interpolation ①
MVS [A1]5000 F300000 TWMB1000; // Linear interpolation (minute block) ②
MVS [A1]100000 F300000 FWMB1000; // Linear interpolation ③

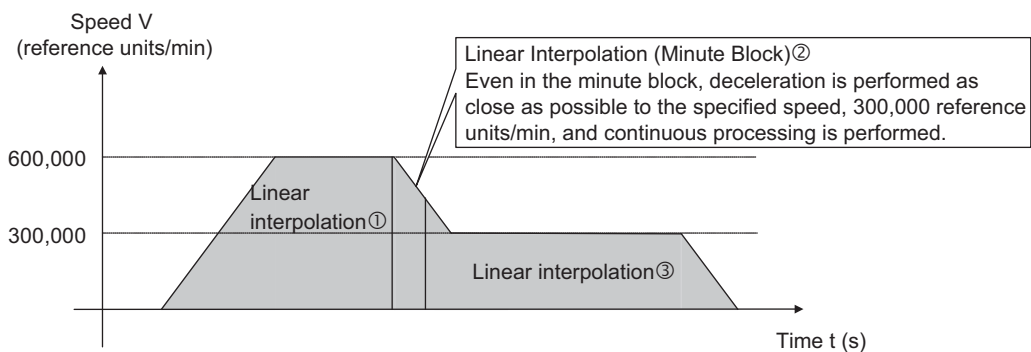
END;
```

The speed waveform for the programming is given below.

• Operation in Interpolation Acceleration/Deceleration Mode 1



• Operation in Interpolation Acceleration/Deceleration Mode 3



■ Additional Information

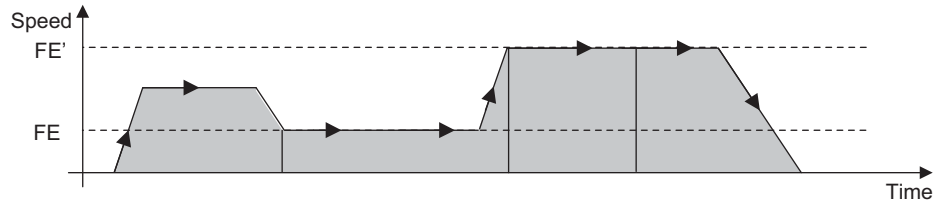
Refer to the following section for additional information on interpolation acceleration/deceleration mode 3 (acceleration/deceleration mode with continuous process control signal monitoring).

 ■ Additional Information (page 6-68)

◆ Interpolation Acceleration/Deceleration Mode 4 (Acceleration/Deceleration Mode with Next Block Speed Specification) Details

Interpolation Acceleration/Deceleration Mode 4 (Acceleration/Deceleration Mode with Next Block Speed Specification) Details

This mode can be used only when the same axes are used for all consecutive interpolation blocks.



■ Format

Use the following format to select interpolation acceleration/deceleration mode 4.

```
ACCMODE 4;
MVS [Logical_axis_name_1] Reference_position Finterpolation_feed_speed Ffinal_interpolation_
feed_speed;
```

Item	Unit	Applicable Data
Final interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	<ul style="list-style-type: none"> Indirect designation with a double-length integer register Directly designated value

Note: The final interpolation feed speed can be omitted. The format is the same for the MCC, MCW, and SKP instructions.

In this mode, you can add the characters "FE" to an interpolation instruction to specify the final speed for the interpolation block.

If you add the characters "FE" to an interpolation instruction, pulse distribution is adjusted so that the interpolation block ends at the specified final interpolation feed speed.

If the specified final interpolation feed speed is 0 (speed units), continuous processing is not performed and the axes decelerate to a stop.

If the characters "FE" are not specified for the interpolation instruction, the final interpolation feed speed that was last specified in the motion program is used.

The final interpolation feed speed is 0 (speed units) when program operation starts.



Important

- The characters "FE" are valid only for this mode. In other interpolation acceleration/deceleration modes, a motion program alarm will occur.
- The valid range for the final interpolation feed speed is 0 to 2,147,483,647 (speed units). A compiler error will occur if a negative number is specified.

Information

This mode can be used with the following or later versions.

- CPU Unit/CPU Module: Version 1.09
- MPE720 version: 7.24

■ Programming Example

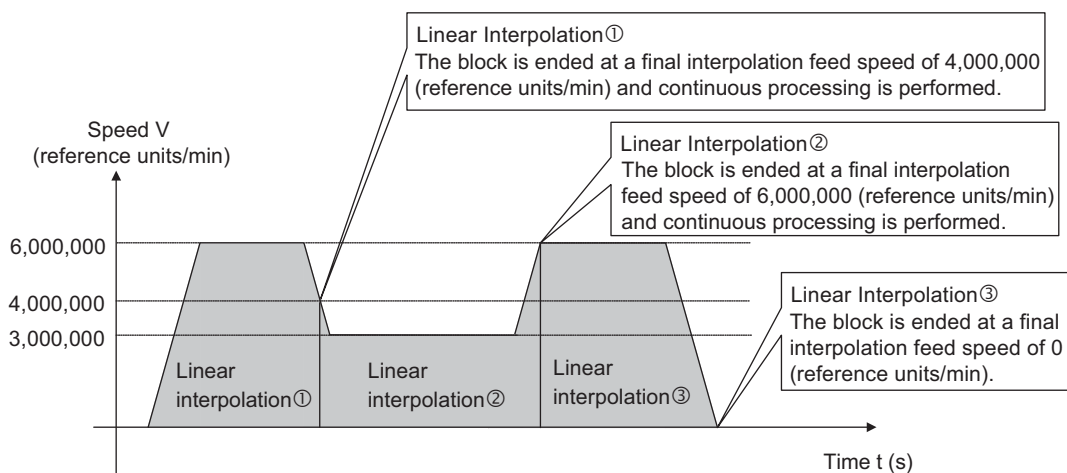
```

FMX T6000000;
IAC T1000;
IDC T1000;
INC;
ACCMODE M4;
MVS [A1]300000 F6000000 FE4000000; "Linear interpolation ①"
MVS [A1]300000 F3000000 FE6000000; "Linear interpolation ②"
MVS [A1]300000 F6000000 FE0;      "Linear interpolation ③"

END;

```

The speed waveform for the programming is given below.



■ Additional Information

Refer to the following additional information for details on operation in the acceleration/deceleration mode with next block speed specification.

- **Request Temporary Stop Operation**
If a temporary stop is requested, the axes decelerate to a stop at the set deceleration rates. If the travel distance is insufficient, a quick stop is performed at the target position.
- **Request Stop Operation**
The interpolation block stops immediately.
- **Operation When Final Interpolation Feed Speed Is Not Reached**
Acceleration or deceleration to the final interpolation feed speed is continued and an immediate stop is performed when the travel distance is reached.
If the next block is an interpolation instruction, continuous processing is performed when the immediate stop occurs.
- **Program Single-block Mode Operation**
No continuous processing to the next interpolation block is performed.
- **Debug Mode Operation**
No continuous processing to the next interpolation block is performed.
- **Operation When the Next Block Is Not an Interpolation Instruction Block**
No continuous processing to the next block is performed.
Acceleration begins from a speed of 0 for the next block.
- **Continuous Operation during Parallel Execution (PFORK)**
Continuous processing is not performed across a PFORK instruction.
Set the instructions so that processing for this mode ends during each fork.

6.2 Axis Movement Instructions

Axis movement instructions are used to move axes that are connected to a Motion Control Function Module.

There are 11 axis movement instructions. You can use these instructions only in motion programs.

The following table lists the axis movement instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
MOV	Positioning	MOV [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logical_axis_name_3] Reference_position ...;	Performs positioning at the positioning speed for up to 32 axes.	○	×
MVS	Linear Interpolation	MVS [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logical_axis_name_3] Reference_position ... Finterpolation_feed_speed;	Performs linear movement at interpolation feed speed F for up to 32 axes.	○	×

Continued on next page.

Continued from previous page.

Instruction	Name	Format		Description	Motion Programs	Sequence Programs
MCW	Clockwise Circular Interpolation	Center Position Designation	MCW [<i>Logical_axis_name_1</i>] <i>End_position</i> [<i>Logical_axis_name_2</i>] <i>End_position</i> U <i>center_point_position</i> V <i>center_point_position</i> T <i>number_of_turns</i> F <i>interpolation_feed_speed</i> ;	Executes circular interpolation at tangential speed F for two axes simultaneously following radius R or designated center point coordinates.	○	×
		Radius Designation	MCW [<i>Logical_axis_name_1</i>] <i>End_position</i> [<i>Logical_axis_name_2</i>] <i>End_position</i> R <i>radius</i> F <i>interpolation_feed_speed</i> ;			
MCC	Counterclockwise Circular Interpolation	Center Position Designation	MCC [<i>Logical_axis_name_1</i>] <i>End_position</i> [<i>Logical_axis_name_2</i>] <i>End_position</i> U <i>center_point_position</i> V <i>center_point_position</i> T <i>number_of_turns</i> F <i>interpolation_feed_speed</i> ;	Multiple circles can be specified after “T” if the center point coordinate is specified. (“T” can be omitted.)	○	×
		Radius Designation	MCC [<i>Logical_axis_name_1</i>] <i>End_position</i> [<i>Logical_axis_name_2</i>] <i>End_position</i> R <i>radius</i> F <i>interpolation_feed_speed</i> ;			

Continued on next page.

Continued from previous page.

Instruction	Name	Format		Description	Motion Programs	Sequence Programs
MCW	Clockwise Helical Interpolation	Center Position Designation	MCW [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Ucenter_point_position Vcenter_point_position [Logical_axis_name_3] End_position_for_linear_interpolation Tnumber_of_turns Finterpolation_feed_speed;	Moves three axes simultaneously with a combination of circular interpolation and linear interpolation outside the circular interpolation plane. Speed F is the circular interpolation tangential speed. The number of turns can be specified after "T" if the center point coordinate is specified. ("T" can be omitted.)	○	×
		Radius Designation	MCW [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Radius [Logical_axis_name_3] End_position_for_linear_interpolation Finterpolation_feed_speed;			
MCC	Counterclockwise Helical Interpolation	Center Position Designation	MCC [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Ucenter_point_position Vcenter_point_position [Logical_axis_name_3] End_position_for_linear_interpolation Tnumber_of_turns Finterpolation_feed_speed;		○	×
		Radius Designation	MCC [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Radius [Logical_axis_name_3] End_position_for_linear_interpolation Finterpolation_feed_speed;			
ZRN	Zero Point Return	ZRN [Logical_axis_name_1]0 [Logical_axis_name_2]0 [Logical_axis_name_3]0 ...;		Returns each axis to its zero point.	○	×

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
DEN	Position after Distribution	MOV [<i>Logical_axis_name_1</i>] <i>Reference_position</i> [<i>Logical_axis_name_2</i>] <i>Reference_position</i> [<i>Logical_axis_name_3</i>] <i>Reference_position ... DEN</i> ;	Performs positioning to the next block after distribution is completed without waiting for a Positioning Completed signal.	○	×
SKP	Skip Function	SKP [<i>Logical_axis_name_1</i>] <i>Reference_position</i> [<i>Logical_axis_name_2</i>] <i>Reference_position</i> [<i>Logical_axis_name_3</i>] <i>Reference_position ...</i> <i>Finterpolation_feed_speed</i> <i>SSskip_input_signal_selection</i> ;	If the SKIP signal turns ON during a linear interpolation operation, the remaining movement is skipped and operation proceeds to the next block.	○	×
MVT	Set-time Positioning	MVT [<i>Logical_axis_name_1</i>] <i>Reference_position</i> [<i>Logical_axis_name_2</i>] <i>Reference_position</i> [<i>Logical_axis_name_3</i>] <i>Reference_position ...</i> <i>Tpositioning_time(ms)</i> ;	Executes positioning by adjusting the feed speed so that travel can be completed at the designated time.	○	×
EXM	External Positioning	EXM [<i>Logical_axis_name_1</i>] <i>Reference_position</i> <i>Dtravel_distance_from_external_positioning_signal_input</i> ;	If an external positioning signal is input during external positioning, the axis is moved only by the travel distance designated after "D" as an incremental value, and then the next instruction is executed.	○	×

Positioning (MOV)

The MOV instruction independently moves each axis from the program current position to the end position at the positioning speed.

Up to 32 axes can be moved with one instruction. Any axis that is not specified in the instruction will not be moved.

The movement path for the MOV instruction will not necessarily be linear like the one that occurs for linear interpolation.

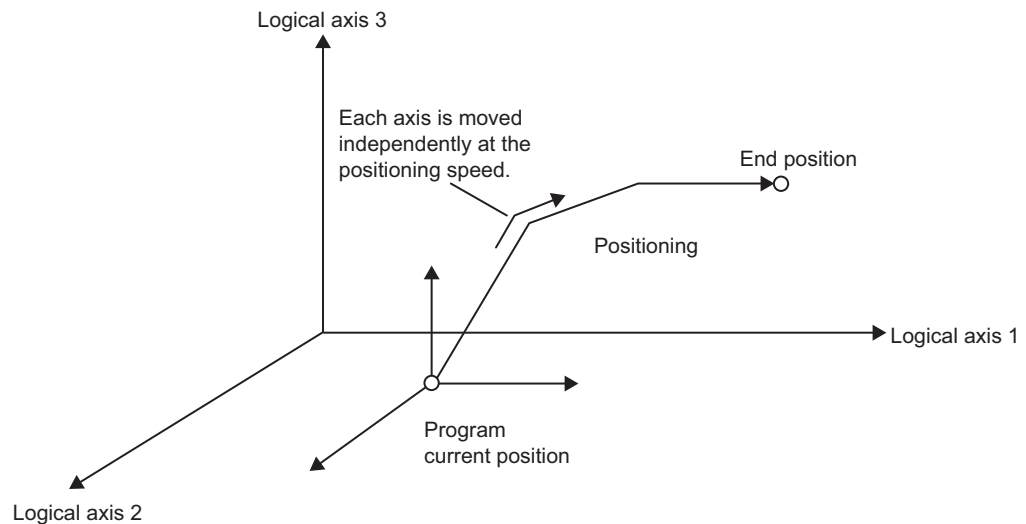


Fig. 6.30 Movement Path for the MOV Instruction

⚠ CAUTION

- The travel path for the Positioning (MOV) instructions will not necessarily be a straight line. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely. There is a risk of injury or device damage.



Important

If an alarm occurs for any axis that is specified in an MOV instruction, a motion program alarm will occur and the axis will stop.

Format

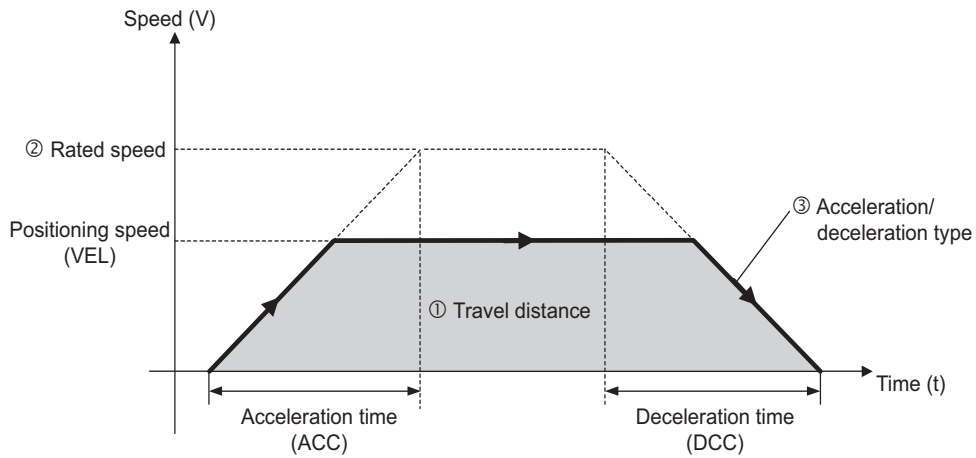
The format of the MOV instruction is as follows:

```
MOV [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logical_axis_name_3] Reference_position. . . ;
```

Item	Unit	Applicable Data
Reference position	Reference units	<ul style="list-style-type: none"> Directly designated value Indirect designation with a double-length integer register

Settings for the MOV Instruction

This section describes the settings for the MOV instruction.



① Travel distance

The travel amount of each axis depends on the movement mode (Absolute or Incremental Mode).

•Absolute Mode Travel Distance

In Absolute Mode, the difference between the program current position and the reference position is the travel position.

•Incremental Mode Travel Distance

In Incremental Mode, the reference position is the travel distance.

② Rated Speed

The rated speed for each axis is set in fixed parameter No. 34 (Rated Motor Speed).

③ Acceleration/Deceleration Type

There are three acceleration/deceleration types for the MOV instruction.

The acceleration/deceleration type is set by a combination of the ACC, DCC, and SCC instructions and bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) or bits 8 to B (Filter Type Selection) in the OW□□□03 setting parameter.

• No Acceleration/Deceleration

This method moves the axes with an acceleration time and deceleration time of 0.

Setting Method	Operation Example
<ul style="list-style-type: none"> • Set bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of OW□□□03 to 1 (ms). • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 0 (No filter). • Set 0 for the ACC instruction. • Set 0 for the DCC instruction. 	

• 1Single-step Linear Acceleration/Deceleration

This method moves the axes with fixed acceleration and deceleration rates.

Setting Method	Operation Example
<ul style="list-style-type: none"> • Set bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of OW□□□03 to 1 (ms). • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 0 (No filter). • Set any value other than 0 for the ACC instruction. • Set any value other than 0 for the DCC instruction. 	

• S-curve Acceleration/Deceleration

This method moves the axes with the S-curve acceleration and deceleration rates.

Setting Method	Operation Example
<ul style="list-style-type: none"> • Set bits 4 to 7 (Acceleration/Deceleration Rate Unit Selection) of OW□□□03 to 1 (ms). • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 2 (Moving average filter). • Set any value other than 0 for the ACC instruction. • Set any value other than 0 for the DCC instruction. • Set any value other than 0 for the SCC instruction. 	

Information A PFN (in-position check) is performed to check if an axis that was moved with a MOV instruction is in the positioning completed range. After the in-position check, the next movement instruction block is executed.

The following figure shows the operation of the PFN instruction.

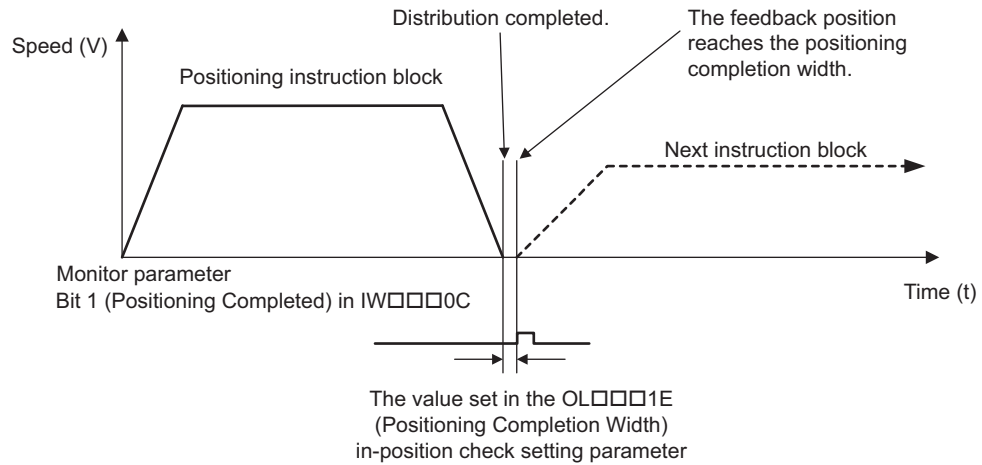


Fig. 6.31 In-Position Check Operation

Programming Example

A programming example that uses the MOV instruction in Absolute Mode is given below.

```
ABS;  
ACC [A1]1000 [B1]1000 [C1]1000;  
DCC [A1]1000 [B1]1000 [C1]1000;  
VEL [A1]2000 [B1]2000 [C1]2000;  
MOV [A1]4000 [B1]3000 [C1]2000;  
END;
```

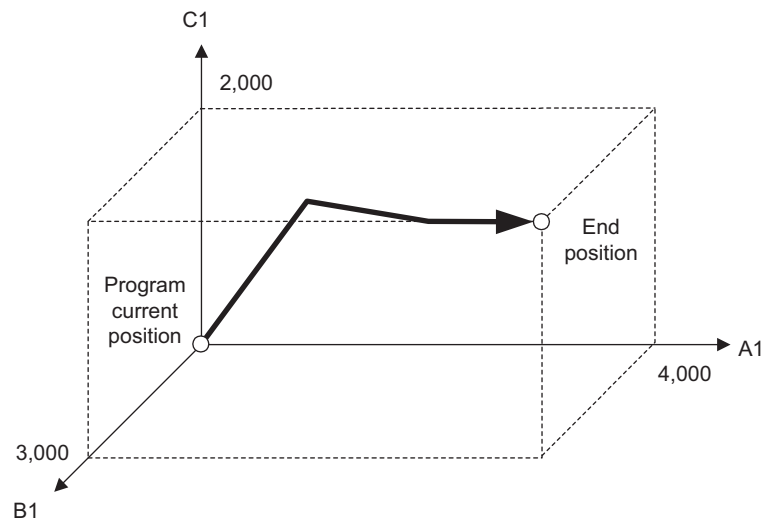


Fig. 6.32 Programming Example for the MOV Instruction

Linear Interpolation (MVS)

The MVS instruction moves each axis linearly at the interpolation feed speed from the program current position to the end position.

Up to 32 axes can be moved with one instruction. Any logical axis that is not specified in the instruction will not be moved.

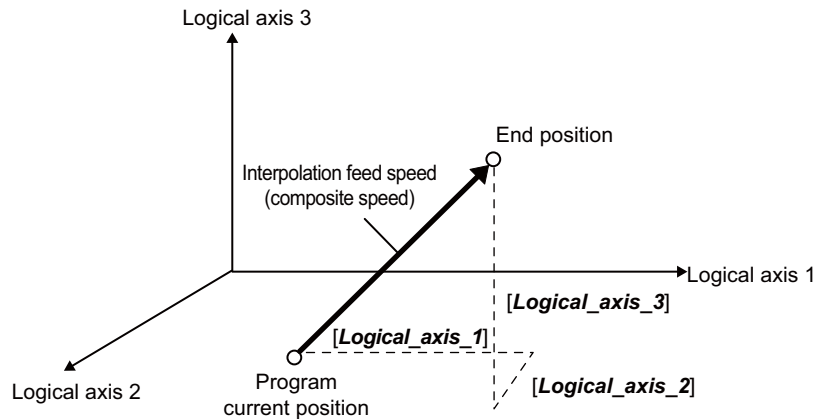


Fig. 6.33 Movement Path for the MVS Instruction

⚠ CAUTION

- The Linear Interpolation (MVS) instruction can be used on both linear axes and rotary axes. However, if a rotary axis is included, the linear interpolation path will not necessarily be a straight line. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely. There is a risk of injury or device damage.



Important

If an alarm occurs for any axis that is specified in an MVS instruction, a motion program alarm will occur and the axis will stop.

Information

A PFN (in-position check) is not performed to check if an axis that was moved with an MVS instruction is in the positioning completed range. Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Format

The format of the MVS instruction is as follows:

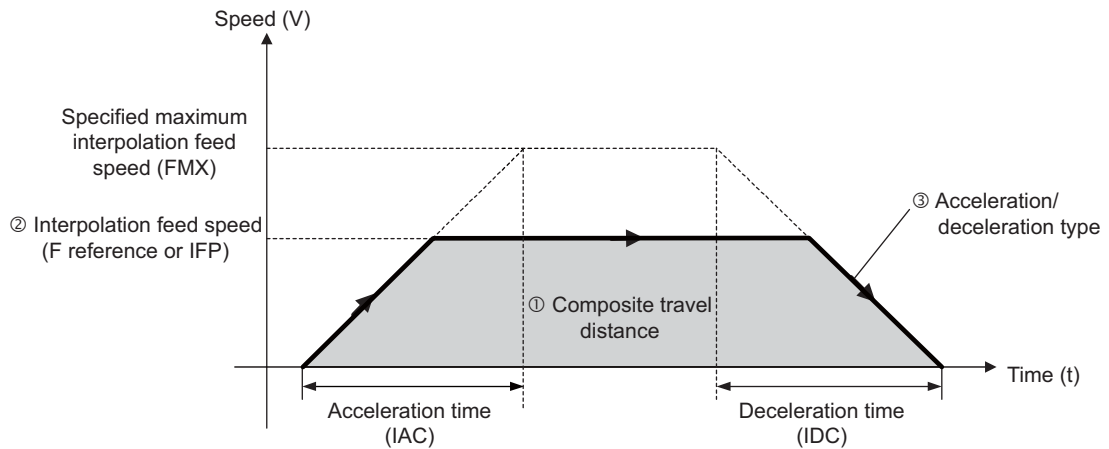
MVS [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logical_axis_name_3] Reference_position . . . Finterpolation_feed_speed;

Item	Unit	Applicable Data
Reference position	Reference units	
Interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	<ul style="list-style-type: none"> Indirect designation with a double-length integer register Directly designated value

Note: You can omit the interpolation feed speed.

Settings for the MVS Instruction

This section describes the settings for the MVS instruction.



① Composite travel distance

The composite travel distance depends on the movement mode: Absolute Mode or Incremental Mode.

•Absolute Mode Composite Travel Distance

In Absolute Mode, the difference between the program current position and the reference position is the composite travel distance.

•Incremental Mode Composite Travel Distance

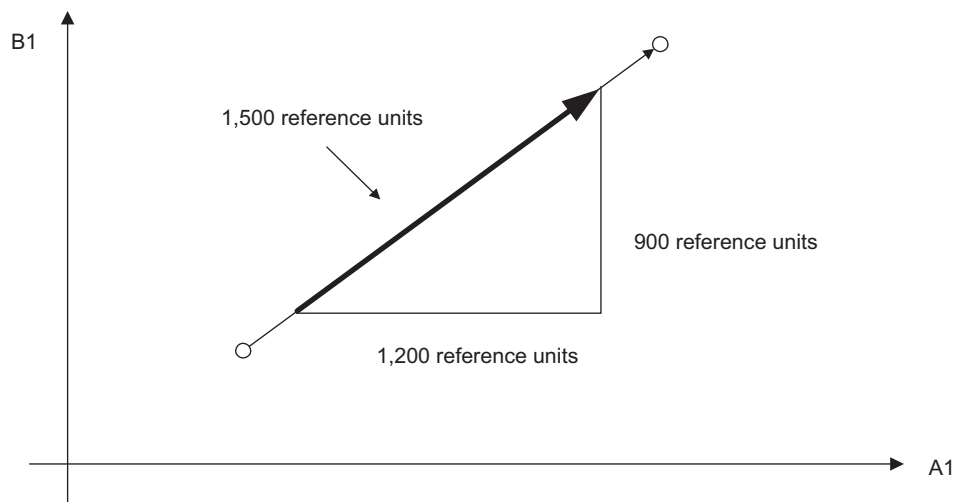
In Incremental Mode, the reference position is the composite travel distance.

Example

INC MVS[A1]1200 [B1]900;

For the above instruction block, the composite travel distance is calculated as follows:

The composite travel distance $\sqrt{1200^2 + 900^2} = 1500$

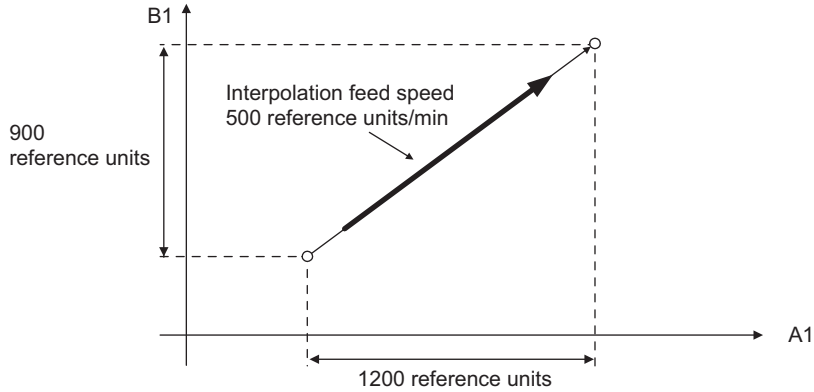


② Interpolation feed speed (F reference or IFP)

You can set the interpolation feed speed by specifying a register or a numerical value after the character “F” in the MVS instruction (F reference). The interpolation feed speed is the composite speed of all of the specified axes.

The valid range is 1 to the maximum interpolation feed speed (FMX) (reference units/min).

Example INC MVS[A1]1200 [B1]900 F500;



The feed speed of each axis is calculated using the following formula.

$$\text{The feed speed of each axis [reference units/min]} = \frac{\text{moving amount of each axis [reference units]}}{\text{composite moving amount [reference units]}} \times \text{interpolation feed speed [reference units/min]}$$

For example, the feed speed of each axis in above condition is calculated as following.

Interpolation feed speed (the value of F) = 500 [reference units/min]

Composite moving amount = $\sqrt{1200^2 + 900^2} = 1500$ [reference units]

- The feed speed of A1 axis = $\frac{1200}{1500} \times 500 = 400$ [reference units/min]
- The feed speed of B1 axis = $\frac{900}{1500} \times 500 = 300$ [reference units/min]

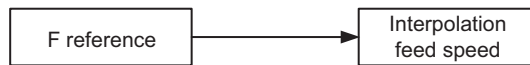
You can select whether to apply an interpolation override with an F reference.

Refer to the following section for how to use interpolation overrides.

Work Registers (page 1-23)

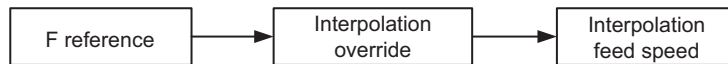
• When Not Specifying an Interpolation Override

F reference = Interpolation feed speed



• When Specifying an Interpolation Override

F reference × Interpolation override (0% to 327.67%) = Interpolation feed speed



The interpolation feed speed can also be specified as a percentage of the maximum interpolation feed speed (FMX).

Refer to the IFP instruction for how to specify the interpolation feed speed as a percentage.



Important

A motion program alarm occurs if a value is specified with an F reference (reference units/min) that exceeds the FMX reference value (reference units/min).

Information

1. If the interpolation feed speed with the interpolation override applied exceeds the FMX reference value, the output value of the interpolation feed speed will be reset to the FMX reference value.
2. When the interpolation feed speed is not specified in the instruction block, the interpolation feed speed that was specified in the previous instruction block is applied.

The interpolation override can be changed during axis movement.

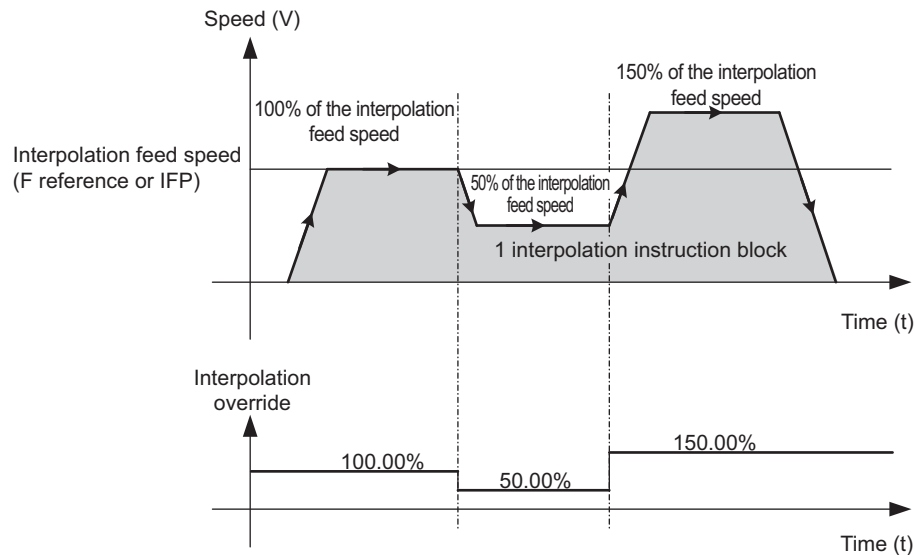


Fig. 6.34 Interpolation Override and Interpolation Instructions

③ Acceleration/Deceleration Type

The acceleration/deceleration type is set by a combination of the IAC, IDC, and SCC instructions and bits 8 to B (Filter Type Selection) in the OW□□□03 setting parameter.

There are three acceleration/deceleration types for the MVS instruction.

• No Acceleration/Deceleration

This method moves the axes with an acceleration time and deceleration time of 0.

Setting Method	Operation
<ul style="list-style-type: none"> • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 0 (No filter). • Set 0 for the IAC instruction. • Set 0 for the IDC instruction. 	

• Single-step Linear Acceleration/Deceleration

This method moves the axes with fixed acceleration and deceleration rates.

Setting Method	Operation
<ul style="list-style-type: none"> • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 0 (No filter). • Set any value other than 0 for the IAC instruction. • Set any value other than 0 for the IDC instruction. 	

• S-curve Acceleration/Deceleration

This method moves the axes with the S-curve acceleration and deceleration rates.

Setting Method	Operation
<ul style="list-style-type: none"> • Set bits 8 to B (Filter Type Selection) of OW□□□03 to 2 (Moving average filter). • Set any value other than 0 for the IAC instruction. • Set any value other than 0 for the IDC instruction. • Set any value other than 0 for the SCC instruction. 	



Important

You must specify the maximum interpolation feed speed (FMX) at the beginning of the motion program.

Otherwise, a motion program alarm will occur when the MVS instruction is executed.

Information

1. If the acceleration/deceleration time is not specified, the default time of 0 ms is applied.
2. An in-position check is not performed to check if an axis that was moved with an MVS instruction is in the positioning completed range. Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Programming Example

A programming example that uses the MVS instruction in Absolute Mode is given below.

```

FMX T30000000;
ABS;
IAC T1000;
IDC T1000;
MVS [A1]4000 [B1]3000 [C1]2000 F50000;
END;

```

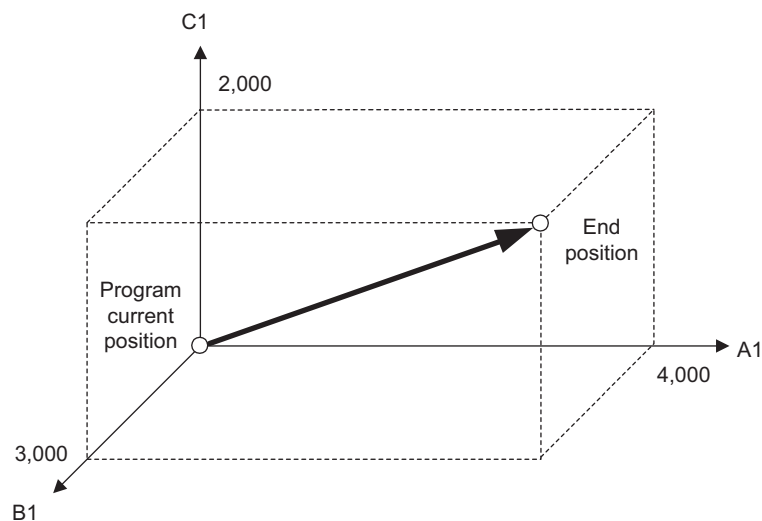
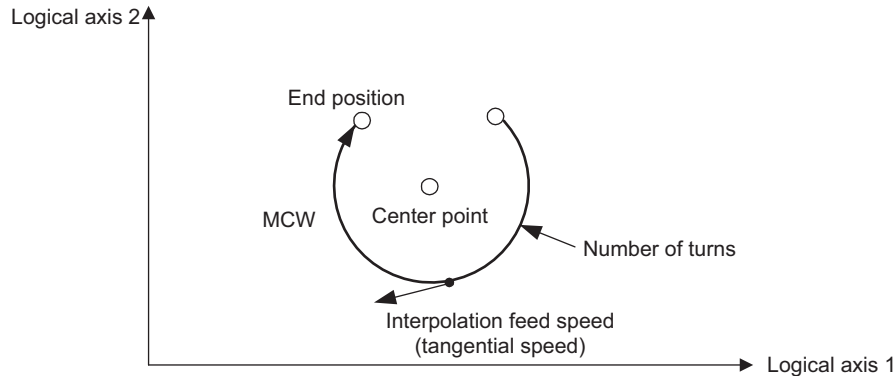


Fig. 6.35 Programming Example for MVS Instruction

Circular Interpolation with Specified Center Point (MCW and MCC)

When used with specified center points, the MCW and MCC instructions move two axes simultaneously from the program current position to the end position on the specified plane at the interpolation feed speed along the circle determined by the center point position.

- MCW: Clockwise
- MCC: Counterclockwise



Important

1. Always specify the plane for circular interpolation with the PLN instruction before you execute a circular interpolation instruction (MCW or MCC) for specified center points. A motion program alarm will occur if an MCW or MCC Circular Interpolation instruction with a specified center point is executed before the PLN instruction.
2. Specify the axes for the end position and center point position in the same order as the axes were specified in the PLN instruction.
3. You must specify the maximum interpolation feed speed (FMX) at the beginning of the program. A motion program alarm will occur if an MCW or MCC Circular Interpolation instruction with a specified center point is executed before the FMX instruction.
4. If an alarm occurs for any axis that is specified in an MCW or MCC Circular Interpolation instruction with a specified center point, a motion program alarm will occur and the axes will stop.

Information

1. If the acceleration/deceleration time is not specified, the default time of 0 ms is applied.
2. An in-position check is not performed to check if an axis that was moved with an MCW or MCC instruction is in the positioning completed range. Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Format

The format of the MCW instruction is as follows:

MCW [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Ucenter_point_position Vcenter_point_position Tnumber_of_turns Finterpolation_feed_speed;

Item	Unit	Applicable Data
End position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register
Center point position	Reference units	
Number of turns	Number of turns	
Interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	

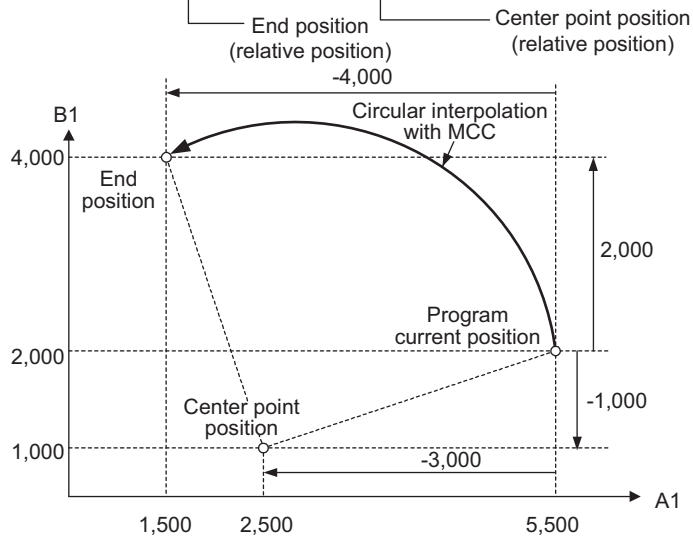
Note: You can omit the number of turns and the interpolation feed speed.

Example Incremental Mode

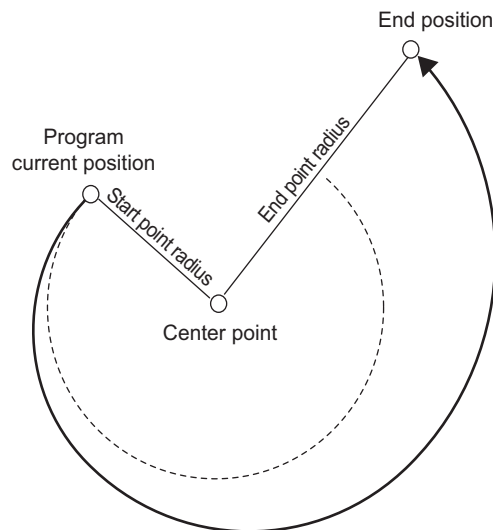
In Incremental Mode, the center point position and end position are treated as relative positions from the program current position.

```

FMX T30000000;
INC;
PLN[A1][B1];
MCC [A1]-4000 [B1]2000 U-3000 V-1000 F50000;
    
```



Set the start point radius and end point radius carefully. The circular interpolation path will become as shown below if the start point radius is not equal to the end point radius.



② Number of turns

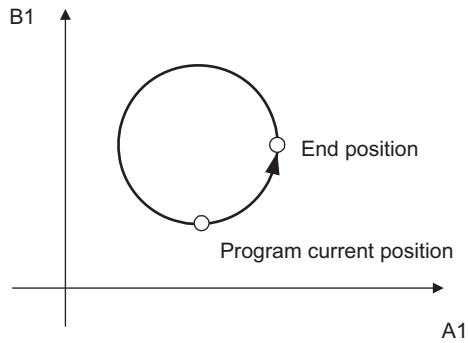
The number of turns is set by specifying a register or a numerical value after the character “T” to the MCW or MCC Circular Interpolation instruction with a specified center point.

You can specify the number of turns to implement multiple circular operations. A motion program alarm occurs if a negative value is set for the number of turns. The number of circular movements that will be performed for the specified number of turns depends on the relationship between the program current position and end position as shown below.

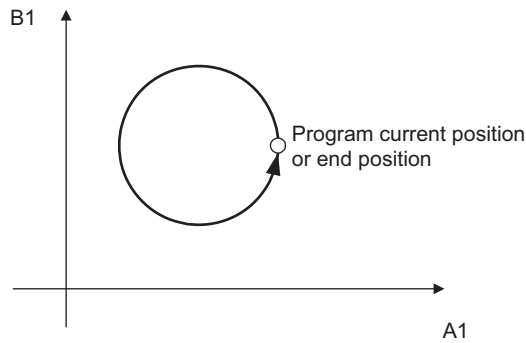
Example

When the Number of Turns Is Set to 2

- If program current position ≠ end position, the circular path consists of 2 circles + 1/4 circle.



- If program current position = end position, the circular path consists of 3 circles.



③ Interpolation feed speed

The interpolation feed speed for an MCW or MCC Circular Interpolation instruction with a specified center point is the speed in the tangential direction.

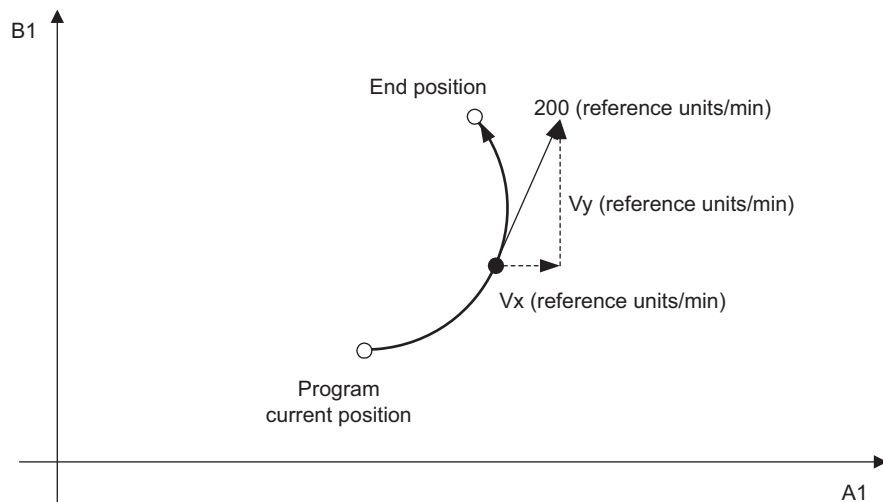
The valid range is 1 to the maximum interpolation feed speed (reference units/min).

Example

For MCC[A1]- [B1]- F200;

The interpolation feed speed for the above instruction block is calculated as follows:

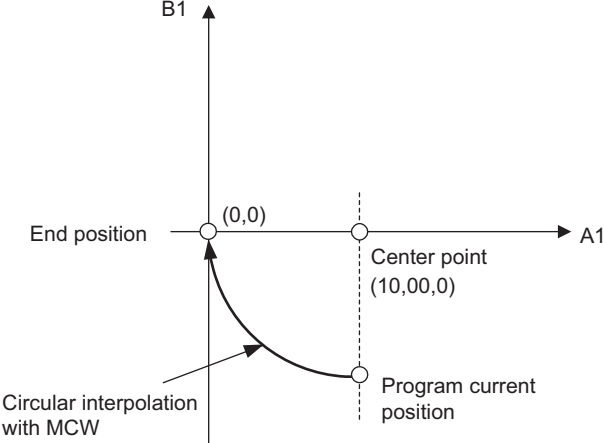
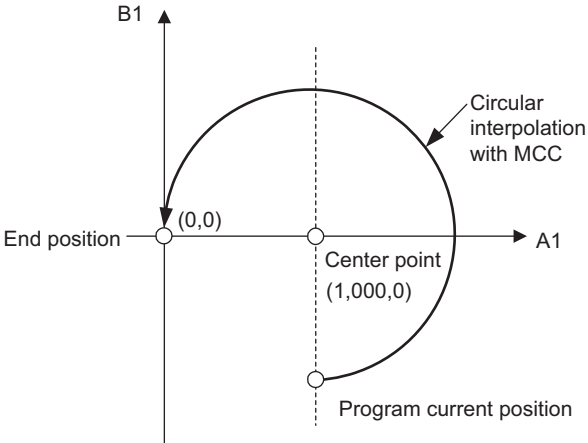
$$\sqrt{V_x^2 + V_y^2} = 200 \text{ (reference units/min).}$$



Programming Examples

Programming examples that use the MCW and MCC Circular Interpolation instructions with specified center points in Absolute Mode are given below.

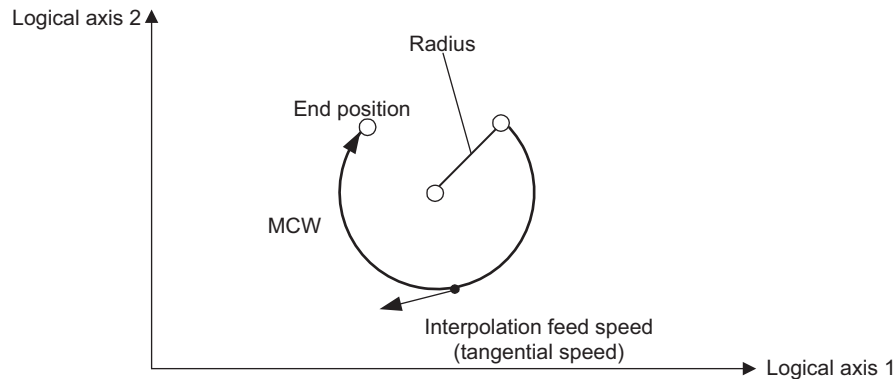
The MCW instruction turns axes clockwise, while the MCC instruction turns axes counterclockwise.

Rotational Direction	Programming Example
<p>Clockwise (MCW)</p>	<pre data-bbox="531 443 1187 589"> ABS; FMX T30000000; PLN [A1][B1]; MCW [A1]0 [B1]0 U1000 V0 F2000; "MCW (Clockwise)" END; </pre>  <p data-bbox="509 1068 1342 1124">Fig. 6.36 Programming Example for the MCW Instruction with a Specified Center Point</p>
<p>Counterclockwise (MCC)</p>	<pre data-bbox="531 1137 1259 1283"> ABS; FMX T30000000; PLN [A1][B1]; MCC [A1]0 [B1]0 U1000 V0 F2000; "MCC (Counterclockwise)" END; </pre>  <p data-bbox="501 1762 1350 1818">Fig. 6.37 Programming Example for MCC Instruction with Specified Center Point</p>

Circular Interpolation with Specified Radius (MCW and MCC)

When used with a specified radius, the MCW or MCC instruction moves two axes simultaneously from the program current position to the end position on the specified plane at the interpolation feed speed along the circle determined by the radius.

- MCW: Clockwise
- MCC: Counterclockwise



Important

1. Always specify the plane for circular interpolation with the PLN instruction before you execute the circular interpolation instruction.
A motion program alarm will occur if an MCW or MCC Circular Interpolation instruction with a specified radius is executed before the PLN instruction.
2. Specify the axes for the end position and center point position in the same order as the axes were specified in the PLN instruction.
3. You must specify the maximum interpolation feed speed (FMX) at the beginning of the program.
A motion program alarm will occur if an MCW or MCC Circular Interpolation instruction with a specified radius is executed before the FMX instruction.
4. If an alarm occurs for any axis that is specified in an MCW or MCC Circular Interpolation instruction with a specified radius, a motion program alarm will occur and the axes will stop.

Information

1. If the acceleration/deceleration time is not specified, the default time of 0 ms is applied.
2. A PFN (in-position check) is not performed to check if an axis that was moved with an MCW or MCC Circular Interpolation instruction with a specified radius is in the positioning completed range. Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Format

The format of the MCW instruction with a specified radius is as follows:

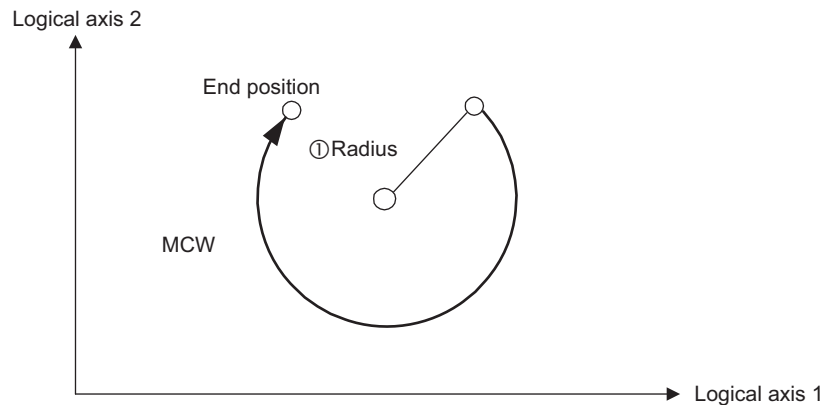
MCW [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Radius Finterpolation_feed_speed;

Item	Unit	Applicable Data
End position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register
Radius	Reference units	
Interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	

- Note: 1. You cannot specify the number of turns if you specify a radius.
2. You can omit the interpolation feed speed.

Settings for the MCW and MCC Instructions with Specified Radii

This section describes the settings for the MCW and MCC instructions with specified radii.



① Radius

The radius is set by specifying a register or a numerical value after the character “R” to the MCW or MCC Circular Interpolation instruction with a specified radius.

The circular interpolation path depends on the sign of the radius reference value as shown below.

Example

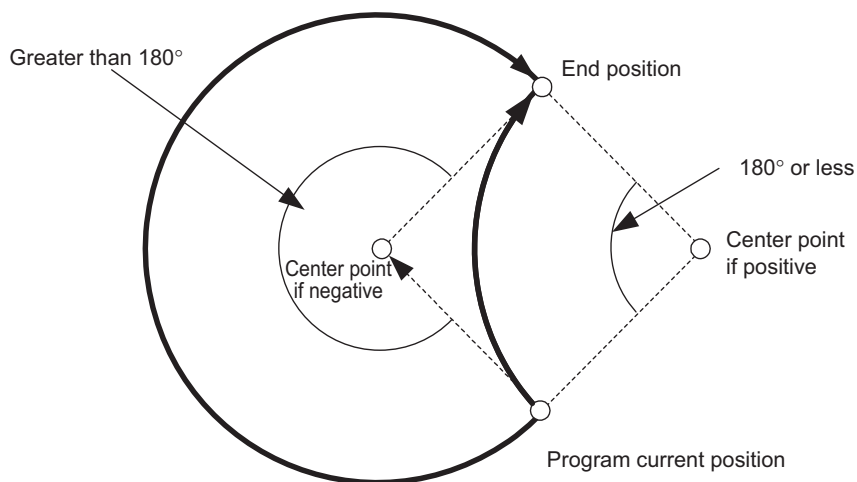
Interpolation Path for the MCW and MCC Instructions with a Specified Radius

For the instruction block: *MCW [A1] - [B1] - R - ;*

If $R > 0$: Circular interpolation with an arc angle of 180° or less

If $R < 0$: Circular interpolation with an arc angle of greater than 180°

If $R = 0$: A motion program alarm occurs.



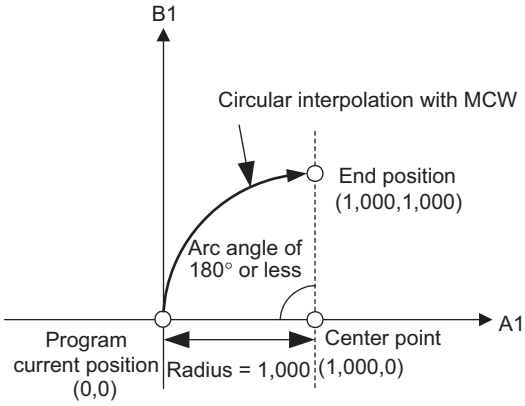
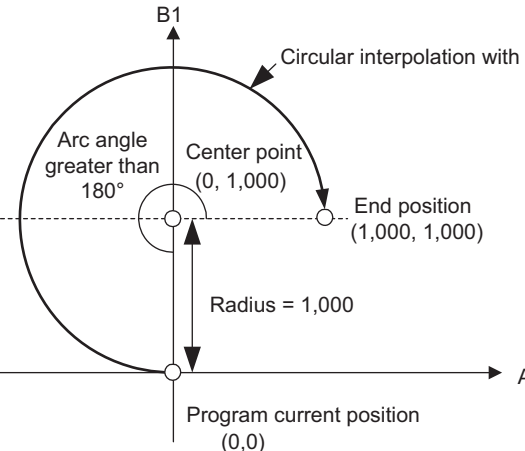
Information

If you specify a radius for circular interpolation, you cannot specify the number of turns.

Programming Examples

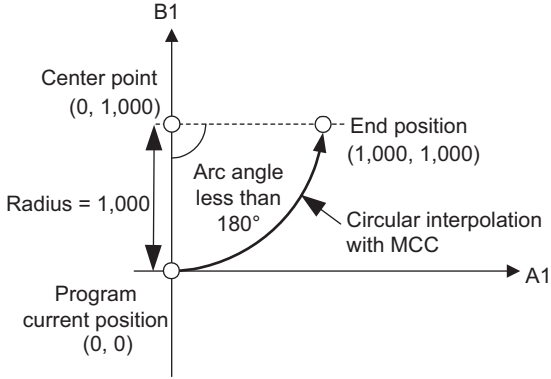
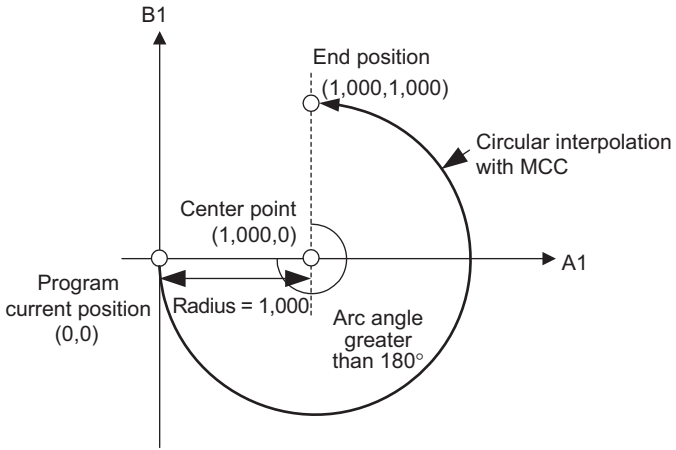
Programming examples that use the MCW and MCC Circular Interpolation instructions with specified radii in Absolute Mode are given below.

The MCW instruction turns axes clockwise, while the MCC instruction turns axes counterclockwise. The sign of the arc angle radius reference value also determines the rotational direction.

Rotational Direction	Arc Angle	Programming Example
Clockwise (MCW)	180° or less (Radius reference value > 0)	<p>ABS; FMX T30000000; PLN [A1][B1]; MCW [A1]1000 [B1]1000 R1000 F2000; "MCW (Clockwise)" END;</p>  <p>Fig. 6.38 Programming Example for the MCW Instruction with a Specified Radius</p>
	Greater than 180° (Radius reference value < 0)	<p>ABS; FMX T30000000; PLN [A1][B1]; MCW [A1]1000 [B1]1000 R-1000 F2000; "MCW (Clockwise)" END;</p>  <p>Fig. 6.39 Programming Example for the MCW Instruction with a Specified Radius</p>

Continued on next page.

Continued from previous page.

Rotational Direction	Arc Angle	Programming Example
Counter-clockwise (MCC)	180° or less (Radius reference value > 0)	<p>ABS; FMX T30000000; PLN [A1][B1]; MCC [A1]1000 [B1]1000 R1000 F2000; "MCC (Counterclockwise)" END;</p>  <p>Fig. 6.40 Programming Example for the MCC Instruction with a Specified Radius</p>
	Greater than 180° (Radius reference value < 0)	<p>ABS; FMX T30000000; PLN [A1][B1]; MCC [A1]1000 [B1]1000 R-1000 F2000; "MCC (Counterclockwise)" END;</p>  <p>Fig. 6.41 Programming Example for the MCC Instruction with a Specified Radius</p>

Helical Interpolation with Specified Center Point (MCW and MCC)

When used with a specified center point, the MCW and MCC instructions simultaneously perform a linear interpolation movement while moving along the circle that is determined by circular interpolation around the specified center point position.

The interpolation feed speed is the composite of the circular interpolation tangential speed and linear interpolation.

- MCW: Clockwise
- MCC: Counterclockwise

CAUTION

- The linear interpolation for the Helical Interpolation (MCW and MCC) instructions can be used for both linear axes and rotary axes. However, depending on how the linear axis is taken, the path of helical interpolation will not be a helix. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely. There is a risk of injury or device damage.



Important

1. Always specify the plane for circular interpolation with the PLN instruction before you execute the helical interpolation instruction.
Use logical axis 1 and logical axis 2 to specify the end positions and center points of circle of the horizontal and vertical axes of the designated plane.
2. Specify the axes for the end position and center point position in the same order as the axes were specified in the PLN instruction.
3. Any axis that has not been specified in the plane designation can be specified as a linear interpolation axis. The axis does not need to be at right angles to the interpolation plane.
4. If an alarm occurs for any axis that is specified in an MCW or MCC Helical Interpolation instruction with a specified center point, a motion program alarm will occur and the axes will stop.

Information

An in-position check (PFN) is not performed to check if an axis that was moved with an MCW or MCC Helical Interpolation instruction with a specified center point is in the positioning completed range.

Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Format

The format of the MCW instruction with a specified center point is as follows:

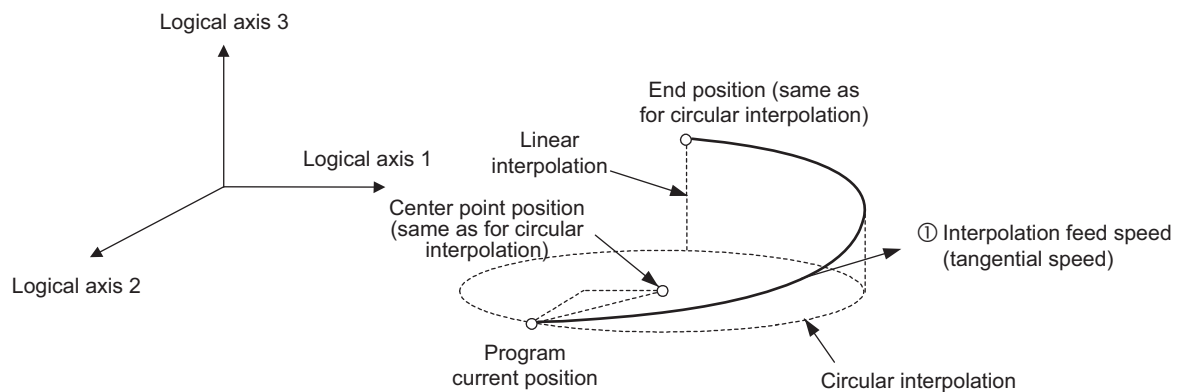
```
MCW [Logical_axis_name_1 End_position] [Logical_axis_name_2] End_position Ucenter_point_posi-
tion Vcenter_point_position
    [Logical_axis_name_3] End_position_for_linear_interpolation Tnumber_of_turns Finterpol-
ation_feed_speed;
```

Item	Unit	Applicable Data
End position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register
Center point position	Reference units	
Number of turns	Number of turns	
Interpolation feed speed	Reference units/min or refer- ence units/s (specified with FUT instruction)	

Note: 1. You cannot specify the number of turns if you specify a radius.
2. You can omit the interpolation feed speed.

Settings for the MCW and MCC Instructions with Specified Center Points

This section describes the settings for an MCW or MCC Helical Interpolation instruction with a specified center point.



① Interpolation feed speed

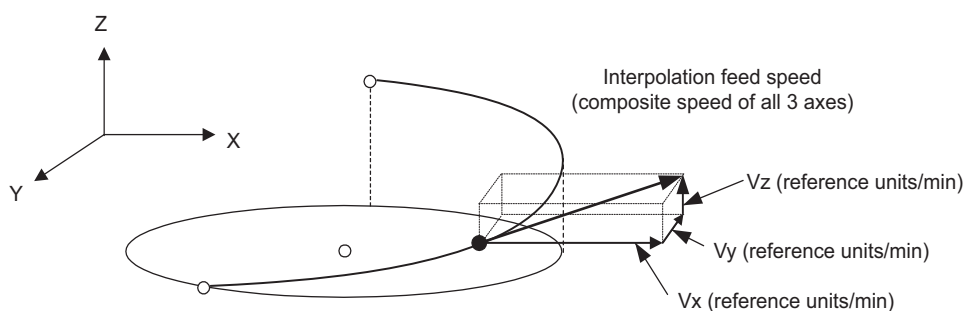
The interpolation feed speed for the MCW or MCC instruction is the composite of the speed of the linear interpolation axis and the speed in the tangential direction of the circular interpolation.

Example

For MCC[X]- [Y]- U- V- [Z]- F300;

The interpolation feed speed for the above instruction block is calculated as follows:

$$\sqrt{V_x^2 + V_y^2 + V_z^2} = 300 \text{ (reference units/min).}$$



Programming Example

A programming example that uses the MCC instruction in Absolute Mode is given below.

```

ABS;
FMX T30000000;
PLN [A1][B1];
MCC [A1]1000 [B1]0 U0 V0 [C1]500 F2000;
END;

```

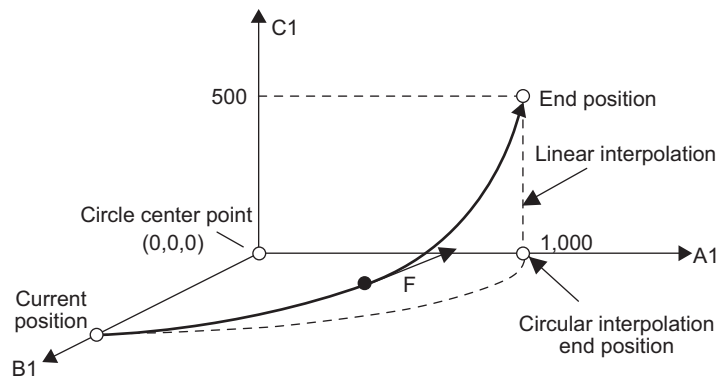


Fig. 6.42 Programming Example for the MCC Instruction with a Specified Center Point

Helical Interpolation with Specified Radius (MCW and MCC)

When used with a specified radius, the MCW and MCC instructions simultaneously perform a linear interpolation movement while moving along the circle that is determined by circular interpolation for the specified radius.

The interpolation feed speed is the composite of the circular interpolation tangential speed and linear interpolation.

- MCW: Clockwise
- MCC: Counterclockwise

CAUTION

- The linear interpolation for the Helical Interpolation (MCW and MCC) instructions can be used for both linear axes and rotary axes. However, depending on how the linear axis is taken, the path of helical interpolation will not be a helix. Check to confirm the paths of the axis when this instruction is used in programs to ensure that the system operates safely. There is a risk of injury or device damage.



Important

1. Always specify the plane for circular interpolation with the PLN instruction before you execute the helical interpolation instruction.
Use logical axis 1 and logical axis 2 to specify the end positions and center points of circle of the horizontal and vertical axes of the designated plane.
2. Specify the axes for the end position and center point position in the same order as the axes were specified in the PLN instruction.
3. Any axis that has not been specified in the plane designation can be specified as a linear interpolation axis. The axis does not need to be at right angles to the interpolation plane.
4. If an alarm occurs for any axis that is specified in an MCW or MCC Helical Interpolation instruction with a specified radius, a motion program alarm will occur and the axes will stop.

Information

An in-position check is not performed to check if an axis that was moved with an MCW or MCC Helical Interpolation instruction with a specified radius is in the positioning completed range. Use the PFN instruction when it is necessary to check if the axis is in the positioning completed range.

Format

The format of the MCW instruction with a specified radius is as follows:

```
MCW [Logical_axis_name_1] End_position [Logical_axis_name_2] End_position Rradius
    [Logical_axis_name_3] End_position_for_linear_interpolation Finterpolation_feed_speed;
```

Item	Unit	Applicable Data
End position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register
Center point position	Reference units	
Radius	Reference units	
Interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	

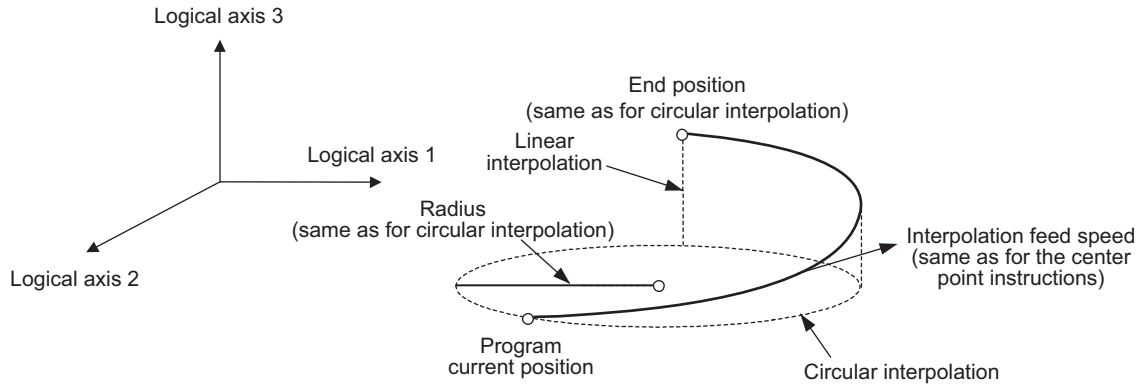
- Note: 1. You cannot specify the number of turns if you specify a radius.
2. You can omit the interpolation feed speed.

Settings for the MCW or MCC Instruction with a Specified Radius

This section describes the settings for the MCW or MCC instruction with a specified radius.

The method used to specify the radius and the end position for the helical interpolation with specified radius instructions are the same as for the circular interpolation with specified radius instructions.

Additionally, the method used to specify the interpolation feed speed is the same as for the helical interpolation with specified center point instructions.



Programming Example

A programming example that uses the MCC Helical Interpolation instruction with specified radius in Absolute Mode is given below.

```

ABS;
FMX T30000000;
PLN [A1][B1];
MCC [A1]1000 [B1]0 R1000 [C1]500 F2000;
END;

```

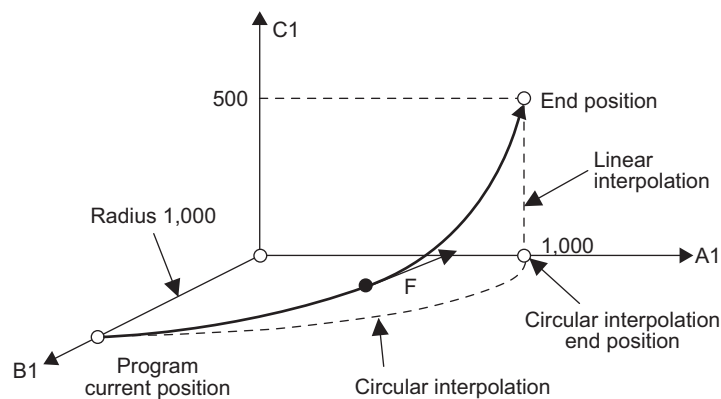


Fig. 6.43 Programming Example for the MCC Instruction with a Specified Radius

Zero Point Return (ZRN)

The ZRN instruction performs a zero point return.

Up to 32 axes can be moved with one instruction. Any axis that is not specified in the instruction will not be moved. Execution moves to the next block only after the zero point return operation has been completed for all specified axes.

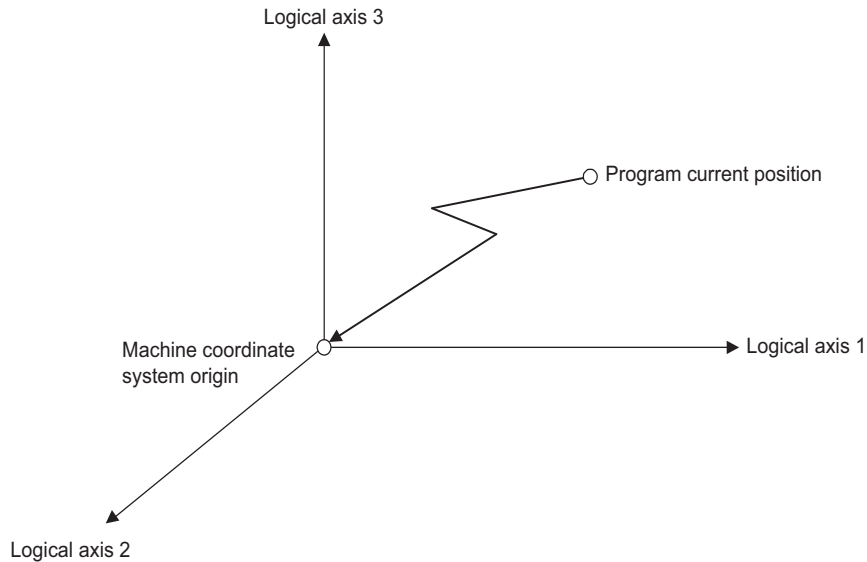


Fig. 6.44 Movement Path for the ZRN Instruction

When the ZRN instruction is executed, the position that the axis returns to is set as the machine coordinate origin. The working coordinate system previously set by the POS instruction is canceled at this time. After the ZRN instruction is executed, the machine coordinate system will be the same as the working coordinate system. The MVM instruction is then invalid until the POS instruction is executed again. Refer to the following section for details on the machine coordinate system and the working coordinate system.

 *Current Position Set (POS) (page 6-117)*




Important

If an alarm occurs for any axis that is specified in an ZRN instruction, a motion program alarm will occur and the axis will stop.



Note

The Request for Pause of Program control signal is invalid while execution of a ZRN instruction is in progress. To stop an operation, use a Request for Stop of Program control signal instead. Refer to the following section for details on Request for Pause of Program and Request for Stop of Program control signals.

 *Work Registers (page 1-23)*

Format

The format of the ZRN instruction is as follows:

```
ZRN [Logical_axis_name_1] 0 [Logical_axis_name_2] 0 [Logical_axis_name_3] 0 ...;
```

Note: Never omit the 0's after the logical axis names.


Settings for the ZRN Instruction

This section describes the settings for the ZRN instruction.

◆ Zero Point Return Method

The zero point return method for each axis is set in the OW□□□3C (Zero Point Return Method) setting parameter. The following table lists the usable zero point return methods.

Refer to the following manual for details on each method.

 *MP3000 Series Motion Control User's Manual* (Manual No.: SIEP C880725 11)

Name	Zero Point Return Method Setting (OW□□□3C)	SVA-01	SVB-01, SVC-01, SVC, or SVC32	PO-01
DEC1 + phase-C pulse	0	Yes	Yes	No
ZERO signal	1	Yes	Yes	No
DEC1 + ZERO signal	2	Yes	Yes	Yes
Phase-C pulse	3	Yes	Yes	No
DEC2 + ZERO signal	4	Yes	No	Yes
DEC1 + LMT + ZERO signal	5	Yes	No	Yes
DEC2 + phase-C signal	6	Yes	No	No
DEC1 + LMT + phase-C signal	7	Yes	No	No
C pulse only	11	Yes	Yes	No
P-OT + phase-C pulse	12	Yes	Yes	No
P-OT	13	Yes	Yes	No
HOME LS & phase-C pulse	14	Yes	Yes	No
HOME LS	15	Yes	Yes	No
N-OT & phase-C pulse	16	Yes	Yes	No
N-OT	17	Yes	Yes	No
INPUT + phase-C pulse	18	Yes	Yes	No
INPUT	19	Yes	Yes	No

Yes: Usable, No: Not usable

◆ Zero Point Return Speed

The zero point return speed depends on the zero point return method that is used.

Programming Example

A programming example that uses the ZRN instruction in Absolute Mode is given below.

The stop position is set at the machine coordinate system origin of (0, 0).

```
ZRN [A1]0 [B1]0;  
END;
```

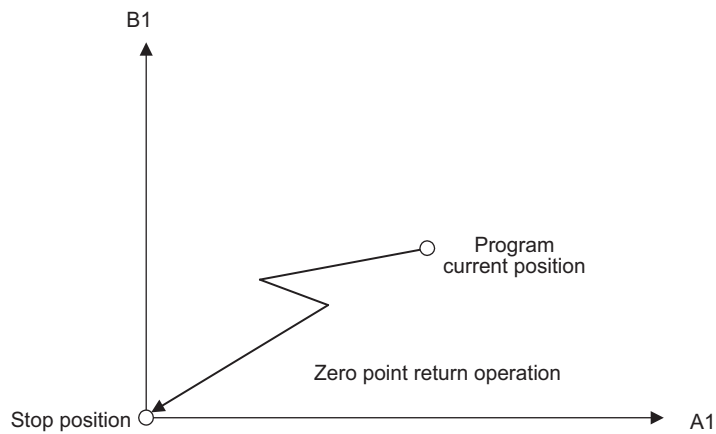


Fig. 6.45 Programming Example for the ZRN Instruction

Position after Distribution (DEN)

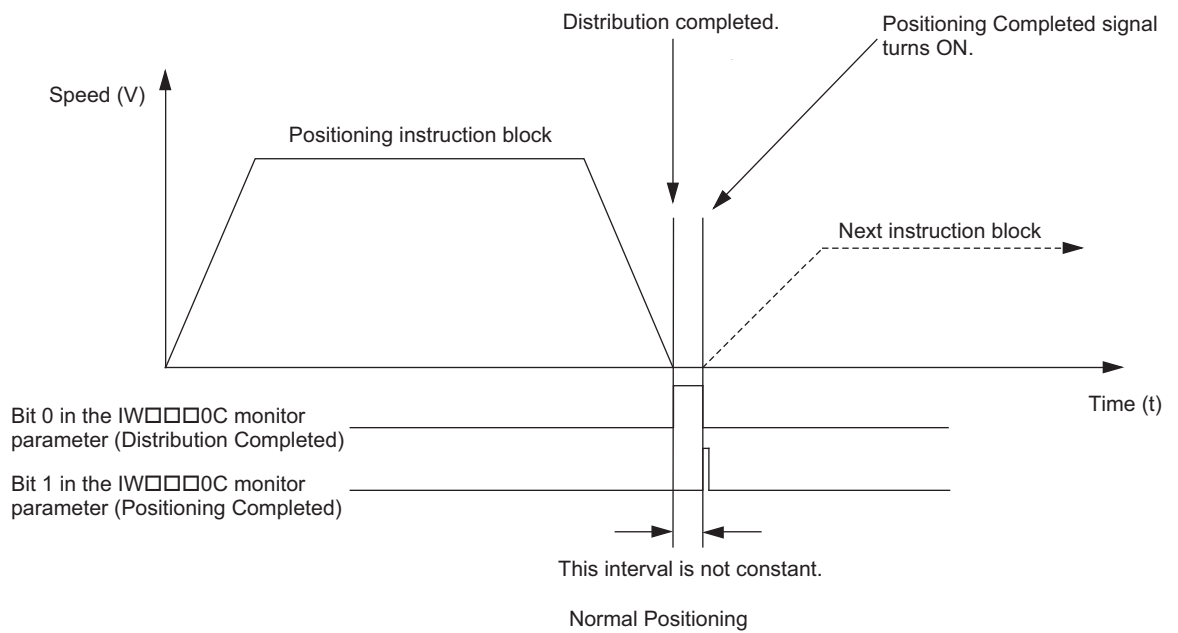
The DEN instruction is an extended version of the MOV instruction.

Up to 32 axes can be moved with one instruction. Any axis that is not specified in the instruction will not be moved.

The DEN instruction is executed in the next instruction block immediately after bit 1 (Distribution Completed) in $IW□□□0C$ turns ON without waiting for bit 0 (Positioning Completed) in $IW□□□0C$.

The operation of the DEN instruction is not the same as a normal positioning operation.

The following figure shows a normal positioning operation.



The following figure shows the positioning operation for the DEN instruction.

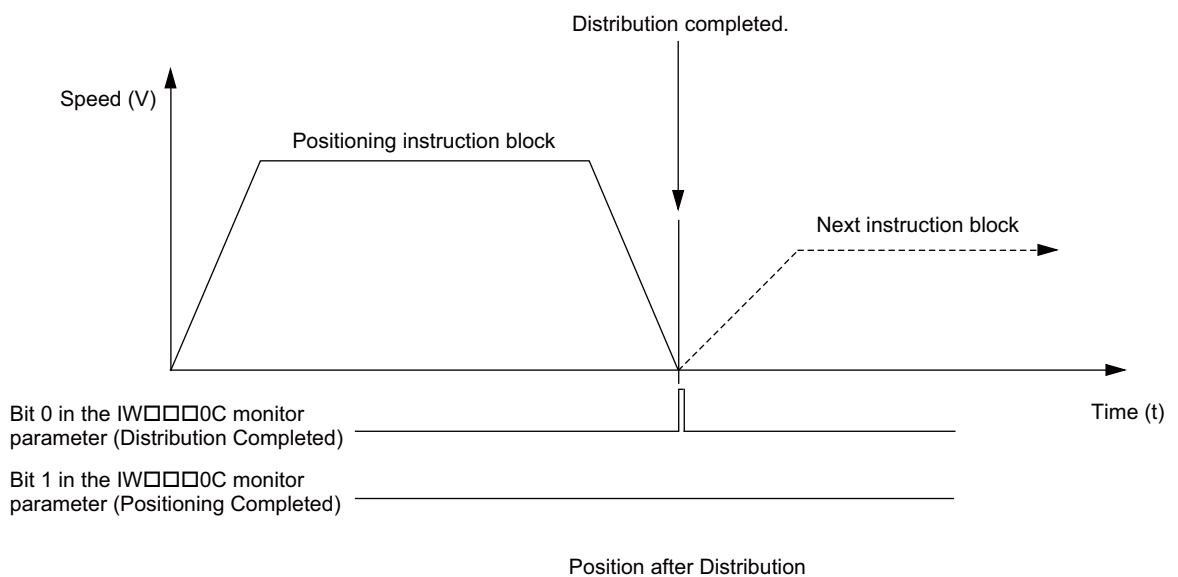


Fig. 6.46 Position after Distribution

Format

The format of the DEN instruction is as follows:

```
MOV [Logical_axis_name_1] - [Logical_axis_name_2] - [Logical_axis_name_3] ..... DEN;
```

Item	Unit	Applicable Data
Reference position	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Programming Example

A programming example that uses the DEN instruction and its positioning path are given below.

```
ABS;
MOV [A1]10000 DEN;
MOV [B1]10000 DEN;
MOV [A1]20000 DEN;
END;
```

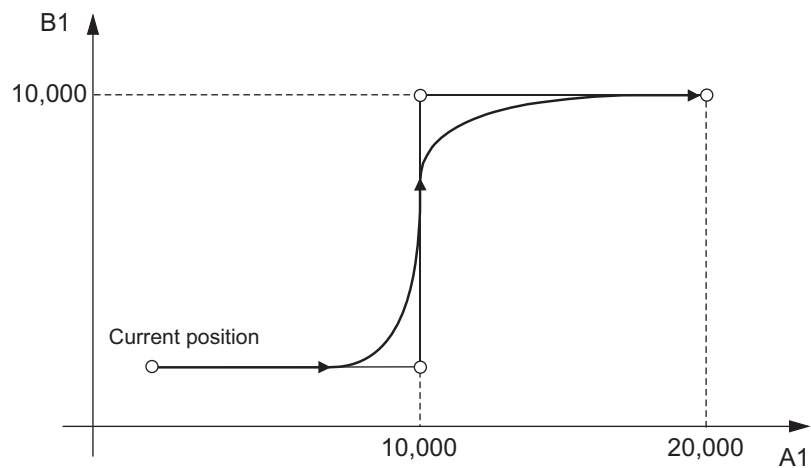


Fig. 6.47 Programming Example for the DEN Instruction

Linear Interpolation with Skip Function (SKP)

The SKP instruction is an extended version of the MVS instruction. If the skip input signal is turned ON during axis movement for a SKP instruction, the axis decelerates to a stop and the remaining travel distance is canceled.

You can use the SKP instruction to program motion control operations that respond to external status changes. The skip signal is input to the control signal for the MSEE instruction or the control register of M-EXECUTOR.

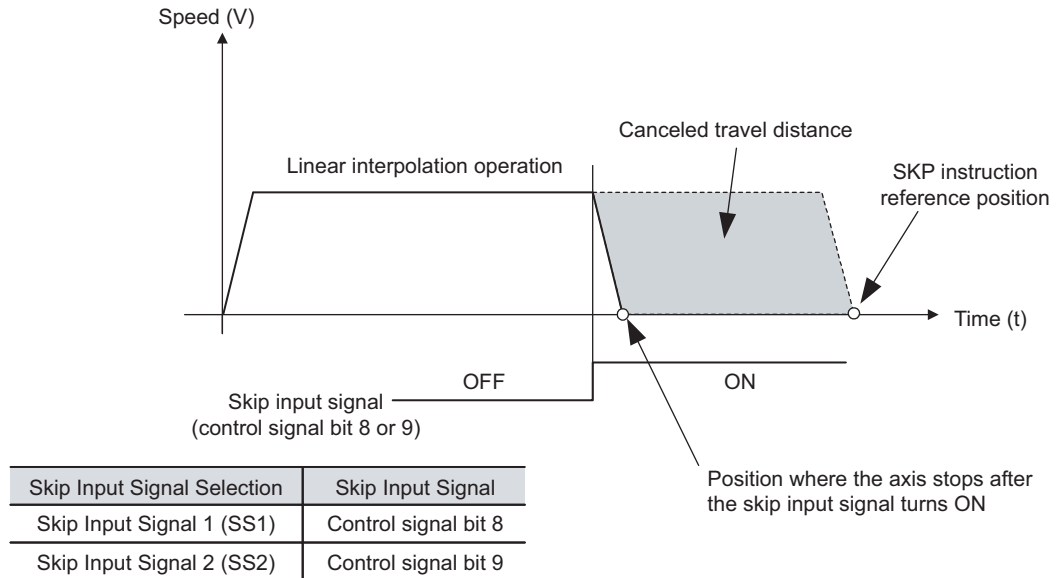


Fig. 6.48 Operation Example for SKP Instruction

Important If an alarm occurs for any axis that is specified in an SKP instruction, a motion program alarm will occur and the axis will stop.

Note The moving axis decelerates to a stop when the skip input signal turns ON. However, the SKP instruction remains active until the Positioning Completed signal turns ON.

Format

The format of the SKP instruction is as follows:

```
SKP [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logical_axis_name_3] Reference_position ... ;
    Finterpolation_feed_speed SSskip_input_signal_selection;
```

Item	Unit	Applicable Data
Reference position	Reference units	
Interpolation feed speed	Reference units/min or reference units/s (specified with FUT instruction)	<ul style="list-style-type: none"> Directly designated value Indirect designation with a double-length integer register
Skip Input Signal Selection	-	<ul style="list-style-type: none"> Directly designated number (1 or 2) Indirect designation with a double-length integer register

Note: You can omit the interpolation feed speed.

Programming Example

A programming example that uses the SKP instruction in Absolute Mode is given below.

```
FMX T30000000;  
ABS;  
IAC T1000;  
IDC T1000;  
SKP [A1]4000 [B1]3000 [C1]2000 F50000 SS1;  
END;
```

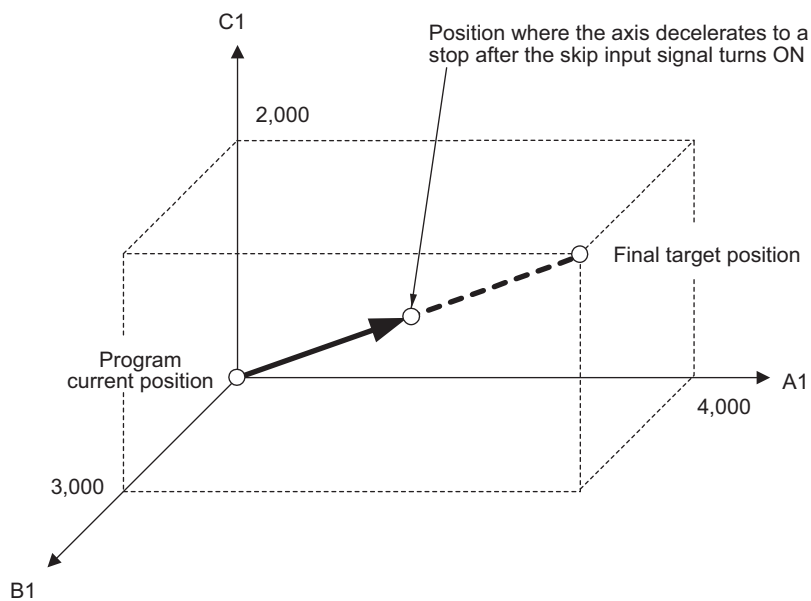


Fig. 6.49 Programming Example for SKP Instruction

Set-time Positioning (MVT)

The MVT instruction is an extended version of the MOV instruction.

Up to 32 axes can be moved with one instruction. Any axis that is not specified in the instruction will not be moved.

When the MVT instruction is used, the feed speed of each axis is adjusted to complete positioning in the specified time. The MVT instruction does not use an interpolation operation, and there is no restriction on completing the positioning for all the specified axes simultaneously.

There is a time lag for the acceleration/deceleration time setting.

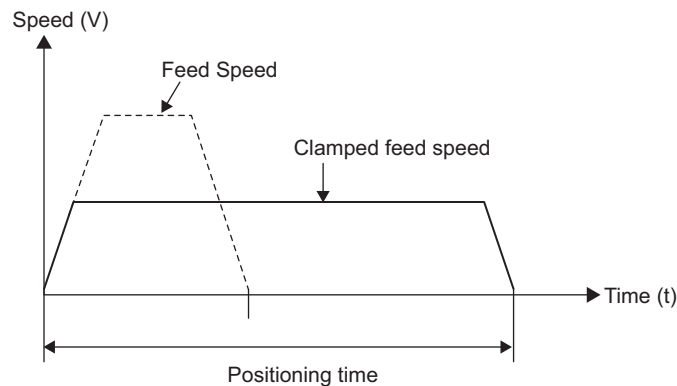


Fig. 6.50 Operation Example for MVT Instruction

Positioning cannot be completed in the specified time if an interpolation override is used.

If a filter is used, the positioning time will be delayed by the filter time constant.

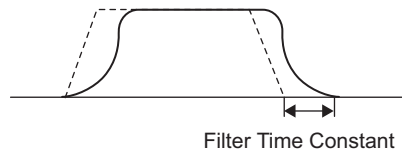


Fig. 6.51 Positioning Time Delay When a Filter Is used



Important

1. The values set by the VEL instruction are overwritten for all axes specified in the MVT instruction. Be sure to set the feed speeds again with the VEL instruction after the MVT instruction is executed.
2. A motion program alarm occurs if 0 is set for the positioning time.
3. A motion program alarm occurs if 0 is set for the travel distance of any axis.
4. If an alarm occurs for any axis that is specified in an MVT instruction, a motion program alarm will occur and the axis will stop.

Format

The format of the MVT instruction is as follows:

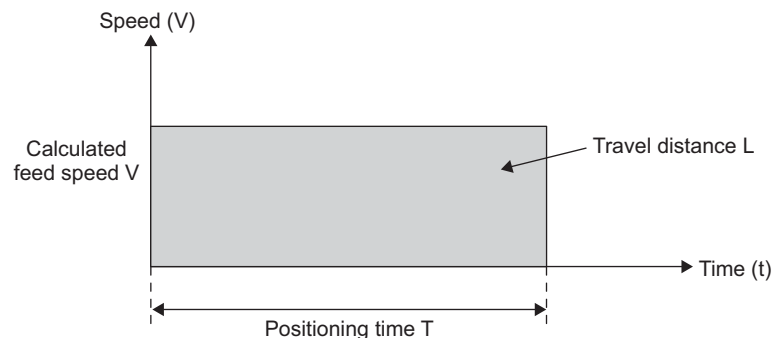
```
MVT [Logical_axis_name_1] Reference_position [Logical_axis_name_2] Reference_position [Logi-
cal_axis_name_3] Reference_position ... ;
    Tpositioning_time;
```

Item	Unit	Applicable Data
Reference position	Reference units	• Directly designated value
Positioning time	ms	• Indirect designation with a double-length integer register

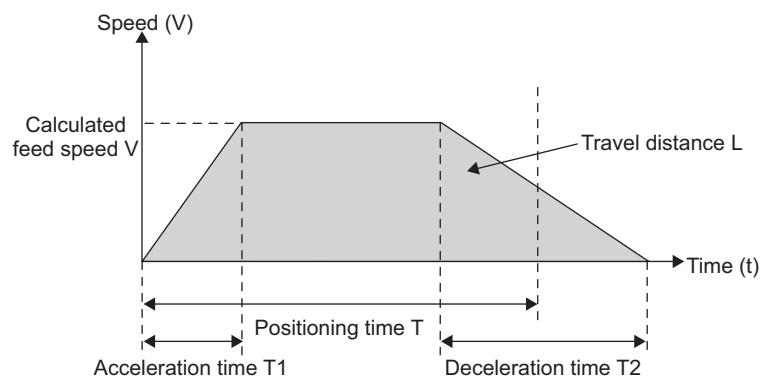
The valid range for the positioning time is 1 to 2,147,483,647 ms.

The feed speed while execution of the MVT instruction is in progress is calculated internally by the Machine Controller based on the positioning time and the travel distance.

This calculation is performed with an acceleration rate of 0, as shown below.



The actual operation when the acceleration time T_1 is less than the deceleration time T_2 is as shown below.



The values set for the VEL instruction are overwritten for all axes specified in the MVT instruction. Be sure to set the feed speeds again with the VEL instruction after the MVT instruction is executed.

Information A PFN (in-position check) is performed to check if an axis that was moved with an MVT instruction is in the positioning completed range, just like for the MOV instruction.

Programming Example

A programming example that uses the MVT instruction in Absolute Mode is given below.

```
ABS;
ACC [A1]1000;
DCC [A1]1000;
MVT [A1]4000 T1000;
END;
```

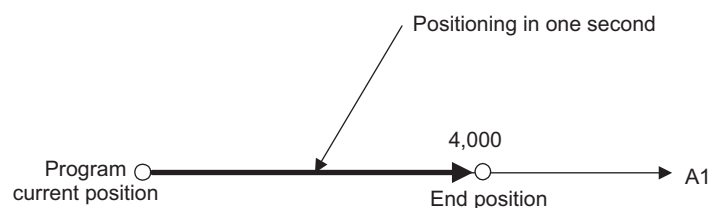


Fig. 6.52 Programming Example for MVT Instruction

External Positioning (EXM)

The EXM instruction is an extended version of the MOV instruction.

The EXM instruction incrementally moves the axis by the specified travel distance to perform positioning when the external positioning signal is turned ON. If the external positioning signal did not turn ON, positioning is performed to the reference position of the EXM instruction.

Only one axis can be specified for the EXM instruction.

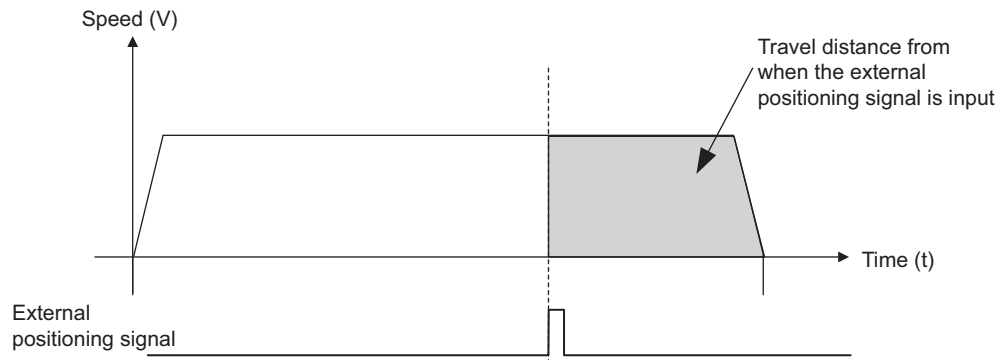


Fig. 6.53 Operation Example for EXM Instruction

If a negative value is specified for the travel distance, the axis decelerates to a stop and then moves in the negative direction.



Important

1. The EXM instruction cannot be used with the PO-01 Function Module.
A motion program alarm will occur if the EMX instruction is executed for a PO-01 Module.
2. Be careful if you use the external latch input signal, because it is also used for the zero point return operation.
3. If an alarm occurs for any axis that is specified in an EXM instruction, a motion program alarm will occur and the axis will stop.

Format

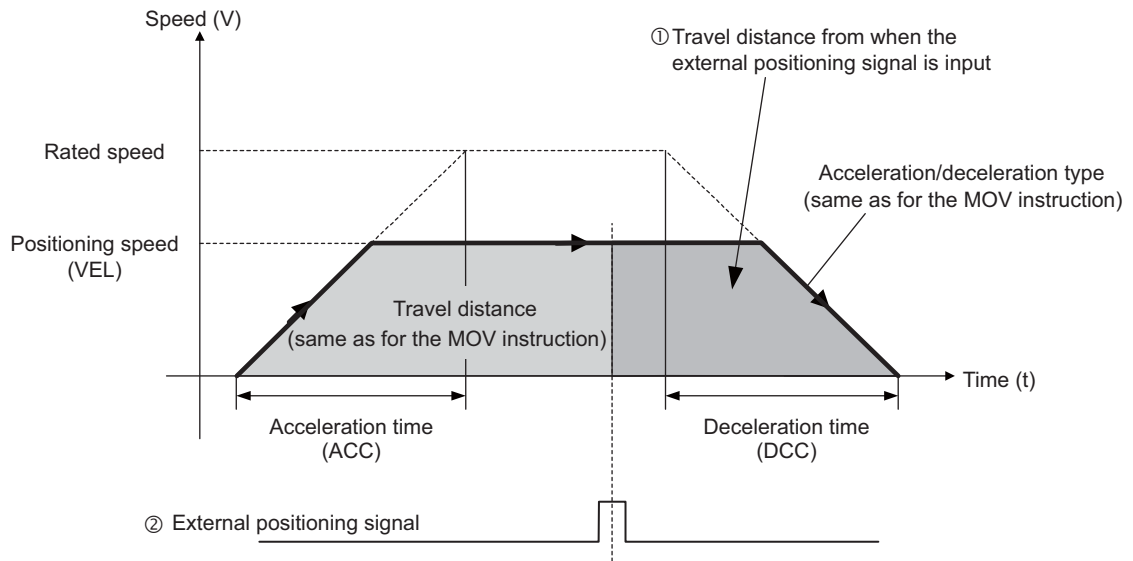
The format of the EXM instruction is as follows:

EXM [*Logical_axis_name_1*] *Reference_position* *Dtravel_distance_from_external_positioning_signal_input*;

Item	Unit	Applicable Data
Reference position	Reference units	<ul style="list-style-type: none"> • Directly designated value
Travel distance from when the external positioning signal is input	Reference units	<ul style="list-style-type: none"> • Indirect designation with a double-length integer register

Settings for the EXM Instruction

This section describes the settings for the EXM instruction.



① Travel distance from when the external positioning signal is input

The travel distance after the external positioning signal is input is set as an incremental value. The valid range is -2,147,483,648 to 2,147,483,647 reference units.

② External positioning signal

The external positioning signal is set in bits 4 to 7 (Function Settings 2) of the OW□□□04 setting parameter.

Programming Example

A programming example that uses the EXM instruction in Absolute Mode is given below.

```

ABS;
ACC [A1]1000;
DCC [A1]1000;
VEL [A1]2000;
DL00000 = 1000;
EXM [A1]4000 DDL00000;
END;

```


6.3

Axis Control Instructions

Axis control instructions control details such as the positions or coordinates of assigned axes. There are seven axis control instructions. You can use these instructions only in motion programs.

The following table lists the axis control instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
POS	Set Current Position	POS [<i>Logical_axis_name_1</i>] <i>New_coordinate_values</i> [<i>Logical_axis_name_2</i>] <i>New_coordinate_values</i> ...;	Changes the current values to the desired coordinate values for up to 32 axes. Subsequent movement instructions use this new coordinate system.	○	×
MVM	Move on Machine Coordinates	MVM MOV [<i>Logical_axis_name_1</i>] <i>Reference_position</i> [<i>Logical_axis_name_2</i>] <i>Reference_position</i> [<i>Logical_axis_name_3</i>] <i>Reference_position</i> ...;	Moves to the target position in the machine coordinate system. The coordinate system that is set automatically on completion of the zero point return is called the machine coordinate system. This coordinate system is not affected by the POS instruction.	○	×
PLD	Update Program Current Position	PLD [<i>Logical_axis_name_1</i>] [<i>Logical_axis_name_2</i>] ... ;	Updates the program current position for axes that were moved manually. Up to 32 axes can be specified with one instruction.	○	×
PFN	In-Position Check	MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ... PFN; Or MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ...; PFN [<i>Logical_axis_name_1</i>] [<i>Logical_axis_name_2</i>] MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ...;	Causes interpolation movement instructions in the same block or in the previous block to proceed to the next block only after the in-position range has been entered.	○	×
INP	In-Position Range	INP [<i>Logical_axis_name_1</i>] <i>NEAR_signal_output_width</i> [<i>Logical_axis_name_2</i>] <i>NEAR_signal_output_width</i> ...;	Sets the NEAR signal output widths (i.e., the in-position ranges). The execution of subsequent interpolation movement instructions that are used with a PFN instruction proceed to the next block only after the NEAR signal output width is entered.	○	×

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
PFP	Positioning Completed Check	MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ... PFP; Or MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ...; PFP [<i>Logical_axis_name_1</i>] [<i>Logical_axis_name_2</i>] MVS [<i>Logical_axis_name_1</i>] - [<i>Logical_axis_name_2</i>] - ...;	Causes interpolation movement instructions in the same block or in the previous block to proceed to the next block only after positioning has been completed.	○	×
PLN	Coordinate Plane Setting	PLN [<i>Logical_axis_name_1</i> (<i>vertical axis</i>)] [<i>Logical_axis_name_2</i> (<i>horizontal axis</i>)];	Designates the coordinate plane to be used for an instruction that requires a plane designation.	○	×

Current Position Set (POS)

The POS instruction changes the current positions of the specified axes to the desired coordinate values and creates new coordinate systems for those axes.

In this manual, the newly set coordinate system is called the working coordinate system, while the original coordinate system of the machine is called the machine coordinate system.

Movement instructions executed after a POS instruction operate in the working coordinate system.

Coordinate System	Description	Remarks
Machine coordinate system	The original coordinate system of the machine	The position for a zero point return is the origin (0).
Working coordinate system	A coordinate system that is constructed with user-defined positions	Create a new coordinate system with the POS instruction.

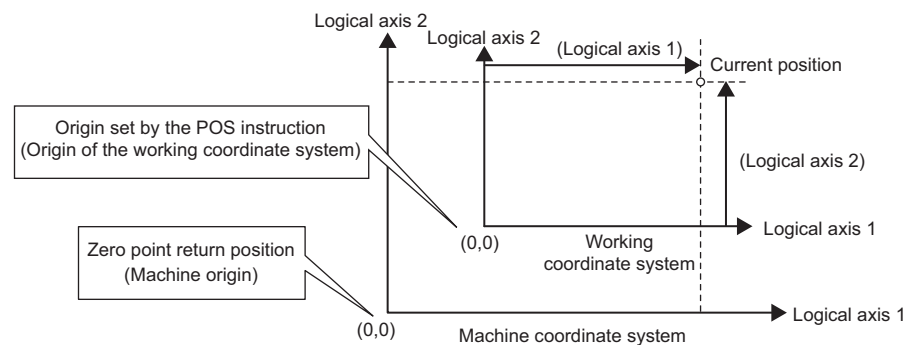


Fig. 6.54 Working Coordinate System Set with POS Instruction

⚠ CAUTION

- The Set Current Position (POS) Instruction creates a new working coordinate system. Therefore, unexpected operation may occur if the POS instruction is specified incorrectly. When you use the POS instruction, always confirm that the working coordinate system is in the correct position before you begin operation. There is a risk of injury or device damage.

The working coordinate system can be changed as often as desired by using the POS instruction.

Always set the machine coordinate system first.

The machine coordinate system is not affected by the POS instruction.

Up to 32 axes can be specified in one POS instruction. The working coordinate system for any unspecified axis is not changed.

Movement instructions in a working coordinate system cannot exceed the maximum programmable value when converted to coordinates in the machine coordinate system.

The following table shows the setting status of the machine coordinate system and the working coordinate system.

Table 6.1 Coordinate System Setting Timing

Coordinate System Setting Timing	Fixed Parameter No. 30 (Encoder Selection)	
	Incremental Encoder/Absolute Encoder Used as Incremental Encoder	Absolute Encoder
After the power supply is turned ON	Machine coordinate system: Temporary* ¹ Working coordinate system: Canceled.* ³	Machine coordinate system: Defined.* ² Working coordinate system: Canceled.
After a ZRN instruction is executed	Machine coordinate system: Set. Working coordinate system: Canceled.	Working coordinate system: Canceled.
After a POS instruction is executed	Working coordinate system: Set.	Working coordinate system: Set.
After the zero point is set	Machine coordinate system: Set.	Machine coordinate system: Set.

*1. Temporary: The origin of the machine coordinate system is set as the current position when the power supply is turned ON.

If a zero point return operation is not performed afterwards, software limit switches cannot be used.

*2. Defined: The origin of the machine coordinate system is created based on the position information from the absolute encoder.

*3. Canceled: The previously set working coordinate system is canceled, and the working coordinate system is set to equal the machine coordinate system.



Important

1. For an infinite-length axis, set a value that is between 0 and POSMAX.
A motion program alarm occurs if a value is set that is outside of this range.
2. When the zero point return operation is executed without using a ZRN instruction, such as a zero point return operation that is executed from a ladder program, the working coordinate system will not be canceled.

Format

The format of the POS instruction is as follows:

POS [*Logical_axis_name_1*] *Coordinate_axis* [*Logical_axis_name_2*] *Coordinate_axis* ...;

Item	Unit	Applicable Data
Coordinate axis	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Programming Example

A programming example that uses the POS instruction is given below.

```
ABS; "Absolute Mode
```

```
MOV [A1]1000 [B1]2000;"Positioning
```

```
POS [A1]0 [B1]0;"Update the working coordinate system.
```

```
MOV [A1]3000 [B1]4000;"Positioning
```

```
DL00000 = IL8010;"Obtain the CPOS (Machine Coordinate System Calculated Position) for axis A1.
```

```
DL00002 = IL8090;"Obtain the CPOS (Machine Coordinate System Calculated Position) for axis B1.
```

```
POS [A1]DL00000 [B1]DL00002;"Cancel the working coordinate system.
```

```
END;
```

Move on Machine Coordinates (MVM)

The MVM instruction is used to temporarily move axes in the machine coordinate system after a working coordinate system that is different from the machine coordinate system has been set with the POS instruction.

Specify MVM for an axis movement instruction to temporarily move the axis to the absolute coordinate position in the machine coordinate system. During execution of an MVM instruction, the axis moves in Absolute Mode regardless of the setting of the movement mode.

The result of the MVM instruction is valid only in the block that contains the MVM instruction. For example, the axes will move in the working coordinate system for the linear interpolation starting from the next block after the MVM instruction.

⚠ CAUTION

- The Move on Machine Coordinates (MVM) instruction temporarily performs positioning to a coordinate position in the machine coordinate system. Therefore, unexpected operation may occur if the instruction is executed without confirming the zero point position in the machine coordinate system first. When you use the MVM instruction, always confirm that the machine zero point is in the correct position before you begin operation. There is a risk of injury or device damage.

Format

The format of the MVM instruction is as follows:

```
MVM MOV .....;
Or
MVM MVS .....;
```

Programming Example

A programming example that uses the MVM instruction is given below.

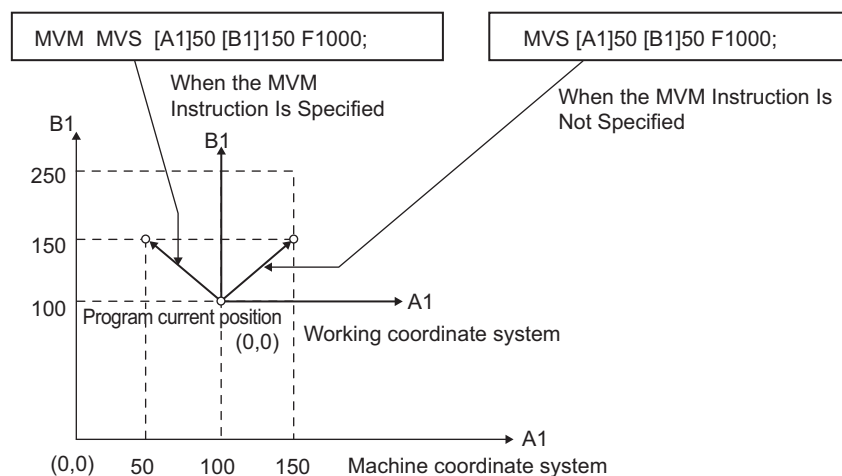


Fig. 6.55 Programming Example for MVM Instruction

Update Program Current Position (PLD)

The PLD instruction updates program current positions that have been changed manually (i.e., manual intervention) or for some other cause. Up to 32 axes can be specified in one instruction.

If an axis is moved from another program (i.e., a ladder program or another motion program) while the motion program is running, the program current position for that axis will not be updated. If the motion program is executed in this status, the axis will move to a position that is offset by the travel distance that occurred for manual intervention.

To solve this problem, the PLD instruction is used to update the program current positions.

Information

1. The PLD instruction is executed by the user when necessary. The PLD instruction is not used in some applications where manual intervention is required while the motion program is running.
2. The program current positions will not be updated for axes that are not specified in the PLD instruction.
3. Use the PLD instruction while the axis is stopped.

Format

The format of the PLD instruction is as follows:

```
PLD [Logical_axis_name_1] [Logical_axis_name_2] [Logical_axis_name_3] ...;
```

Programming Example

A programming example that uses the PLD instruction is given below.

◆ Manual Intervention during Motion Program Operation

```
MOV [A1]1000;
    "Axis A1 was jogged during execution of this instruction block.
    PLD [A1];"Update the program current position.
MOV [A1]2000;
```

◆ Axis Is Moved in a Motion Program User Function

```
MOV [A1]1000;
UFC FNC10 MB000000 IW0100 MB000020;"Axis A1 was moved by a user function.
    PLD [A1];"Update the program current position.
MOV [A1]2000;
```

◆ Precautions

If you execute a PLD instruction immediately after an interpolation instruction (a MVS, SKP, MCW, or MCC instruction) for an axis specified by a Motion Module (SVA-01, SVB-01, SVC-01, or PO-01), always execute the EOX instruction (One Scan Wait) before the PLD instruction.

If you do not execute the EOX instruction, a delay in updating the data in the scan may prevent updating the current position of the program correctly.

Example

Example of Executing the EOX Instruction before the PLD Instruction

MVS [A1]1000;	"Execute interpolation instruction for axis allocated to Optional Module.
EOX;	"One Scan Wait
PLD [A1];	"Update the program current position.
MOV [A1]1000;	

In-position Check (PFN)

The PFN instruction checks to see whether the axes have entered the positioning proximity during an interpolation operation.

An in-position check is not normally performed to check if an axis that was moved with an MVS, MCW, MCC, or SKP interpolation instruction is in the positioning completed range. Use the PFN instruction when it is necessary to check if an axis is in the positioning completed range.

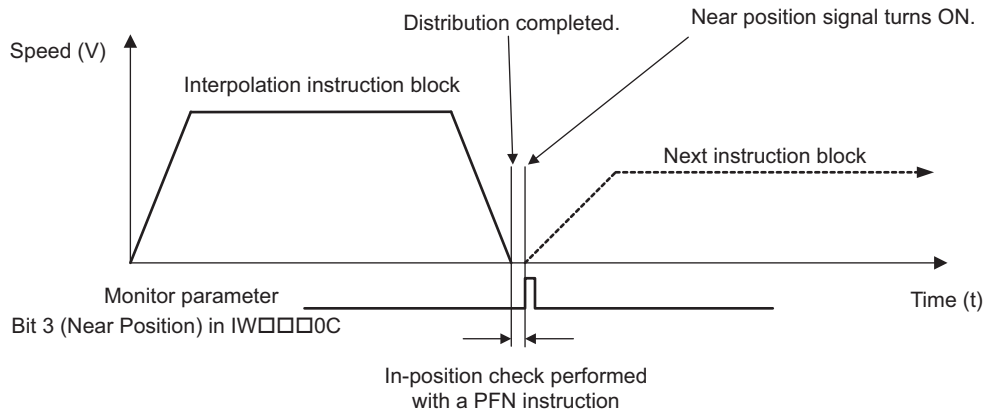


Fig. 6.56 Operation of PFN Instruction

Bit 3 (Near Position Signal) in the IW□□□0C monitor parameter turns ON when $|MPOS - APOS| \leq$ NEAR Signal Output Width.

Set the NEAR signal output width with the INP instruction.

Information If the NEAR signal output width is set to 0, bit 3 in the IW□□□0C monitor parameter turns ON when reference pulse distribution, including the filter, is completed.

Format

The format of the PFN instruction is as follows:

- When Specified in the Same Block as an Interpolation Instruction
MVS [Logical_axis_name_1] - [Logical_axis_name_2] - [Logical_axis_name_3] ... PFN;
- When Specified Independently
PFN [Logical_axis_name_1] [Logical_axis_name_2] [Logical_axis_name_3] ...;

Programming Example

A programming example that uses the PFN instruction is given below.

◆ When Specified in the Same Block as an Interpolation Instruction

```
MVS [A1]1000 F20000 PFN;
MOV [A1]3000;
END;
```

◆ When Specified Independently

```
MVS [A1]1000 F20000;
PFN [A1];
MOV [A1]3000;
END;
```

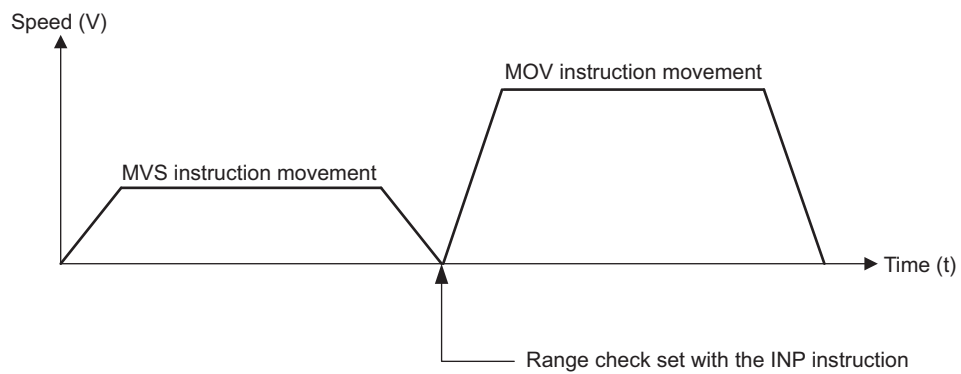


Fig. 6.57 Programming Example for the PFN Instruction

In-Position Range (INP)

The INP instruction sets the in-position range.

Up to 32 axes can be specified in one instruction. The OL□□□20 (NEAR Signal Output Width) setting parameter is updated for each specified axis.

The valid range is 1 to 65,535 reference units.

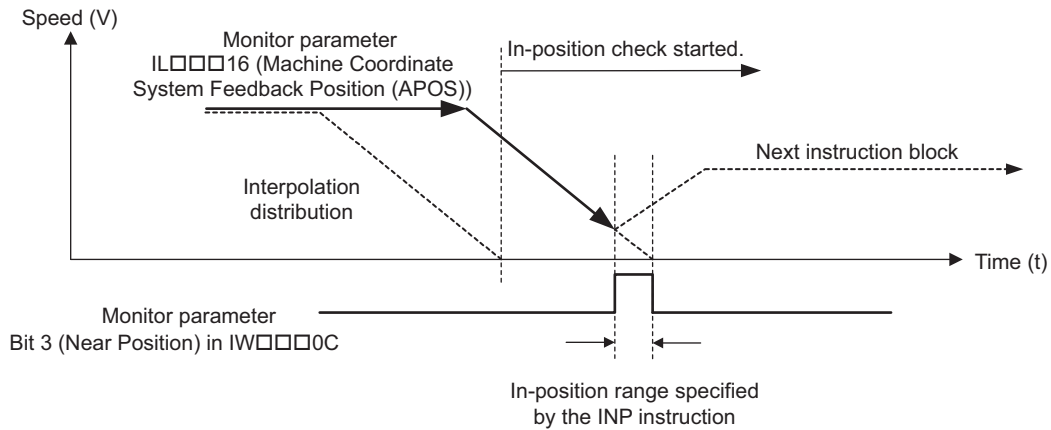


Fig. 6.58 How to Specify the INP Instruction

Information

The SVR or SVR32 Function Module does not support the OL□□□20 (NEAR Signal Output Width) setting parameter.

The in-position range is always 0 for the SVR or SVR32 Function Module.

Format

The format of the INP instruction is as follows:

```
INP [Logical_axis_name_1] NEAR_signal_output_width [Logical_axis_name_2] NEAR_signal_output_width ...;
```

Item	Unit	Applicable Data
NEAR Signal Output Width	Reference units	<ul style="list-style-type: none"> • Directly designated value • Indirect designation with a double-length integer register

Programming Example

A programming example that uses the INP instruction is given below.

```

ABS;
MOV [A1]0 [B1]0;"Positioning to origin
INP [A1]100 [B1]200;"Set the in-position range
MVS [A1]1000 PFN;
MVS [B1]1000 PFN;
MVS [A1]-1000 ;
END;

```

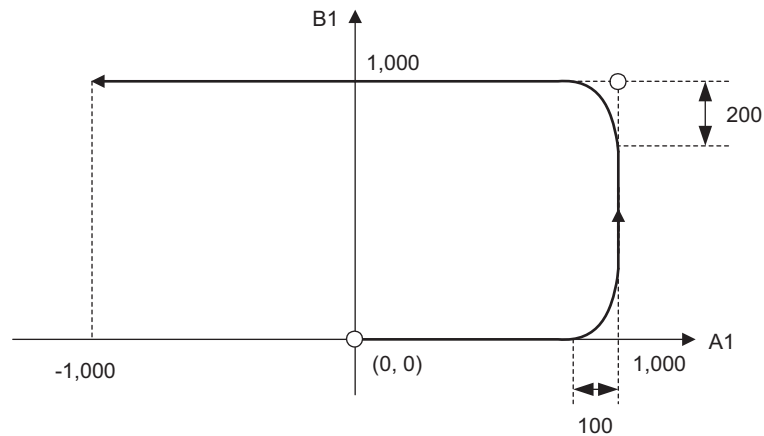


Fig. 6.59 Programming Example for INP Instruction

Positioning Completed Check (PFP)

The PFP instruction checks to see whether positioning has been completed for the specified axes moved by an interpolation instruction.

A positioning completed check is not performed for axes moved by an MVS, MCW, MCC, or SKP interpolation instruction and execution moves to the next instruction block.

Use the PFP instruction when it is necessary to check if positioning has been completed for an axis.

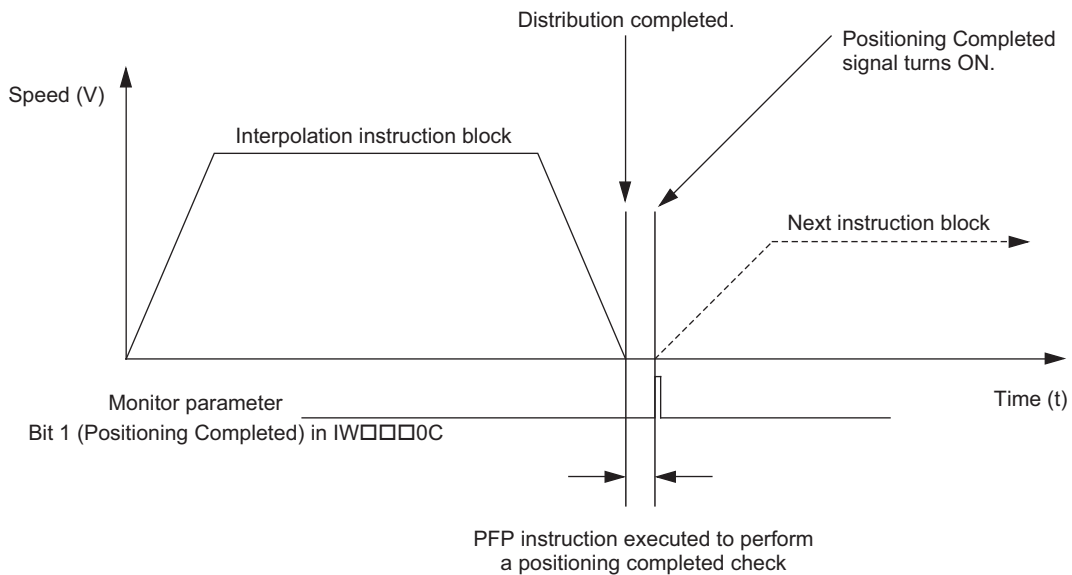


Fig. 6.60 Positioning Completed Check

Bit 1 (Positioning Completed Signal) in the IW□□□□0C monitor parameter turns ON when distribution has been completed and the current position is in the positioning completed range.

Set the positioning completion width in OW□□□□1E.

Format

The format of the PFP instruction is as follows:

- When Specified in the Same Block as an Interpolation Instruction
MVS [Logical_axis_name_1] - [Logical_axis_name_2] - [Logical_axis_name_3] ... PFP;
- When Specified Independently
PFP [Logical_axis_name_1] [Logical_axis_name_2] [Logical_axis_name_3] ...;

Programming Example

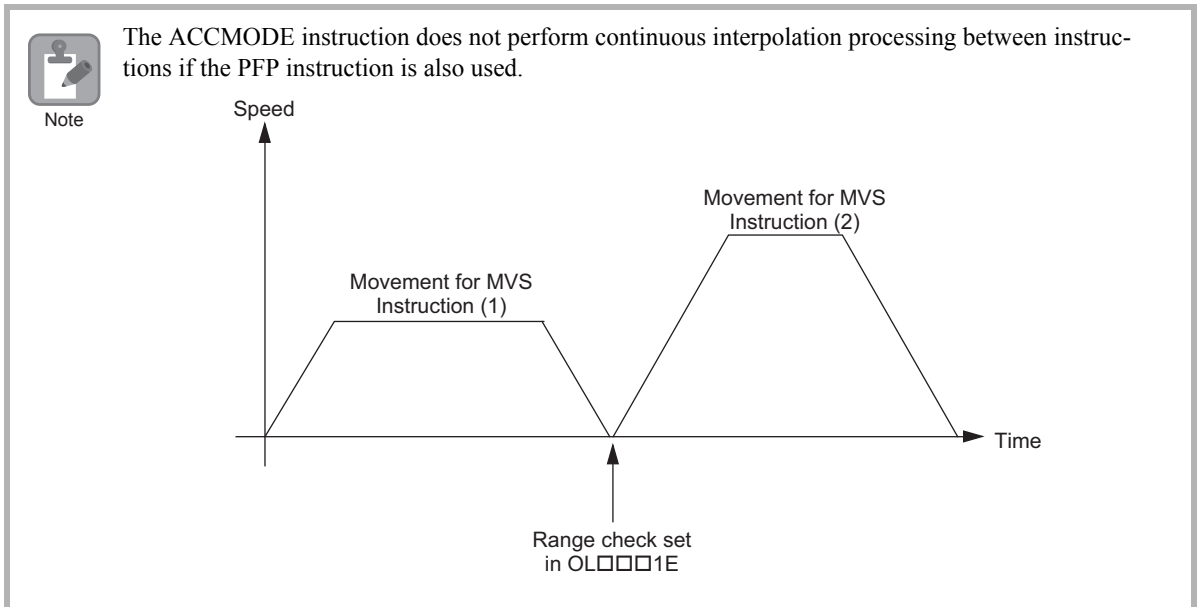
A programming example that uses the PFP instruction is given below.

◆ When Specified in the Same Block as an Interpolation Instruction

```
MVS [A1]1000 F20000 PFP;"MVS instruction (1)
MVS [A1]3000 F50000;"MVS instruction (2)
END;
```

◆ When Specified Independently

```
MVS [A1]1000 F20000;"MVS instruction (1)
PFP [A1];
MVS [A1]3000 F50000;"MVS instruction (2)
END;
```



Coordinate Plane Setting (PLN)

The PLN instruction specifies two logical axes in the parameters to define a coordinate plane. Always execute this instruction before you execute an MCW or MCC circular or helical interpolation instruction. The designated coordinate plane remains in effect until it is reset by another PLN instruction or until the END instruction.

Format

The format of the PLN instruction is as follows:

```
Horizontal axis name  Vertical axis name
PLN [Logical_axis_name_1] [Logical_axis_name_2];
```

Specify the two axes to define the coordinate plane.

Programming Example

A programming example that uses the PLN instruction is given below.

```
PLN[A1][B1];"Specify axis A1 and axis B1 to make up the plane.
MCW [A1]50 [B1]50 R50 F1000;
```

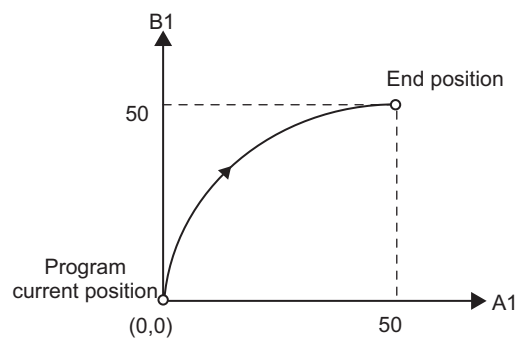


Fig. 6.61 Programming Example for PLN Instruction



Note

Specify the end position and center point position for circular interpolation or helical interpolation in the same order used to specify the axes for the PLN instruction.

```
PLN [Logical_axis_name_1] [Logical_axis_name_2];
MCC [A1]1500  [B1]4000  U2500  V1000  F150;
```

6.4

Program Control Instructions

Program control instructions control the execution sequence of a program.

There are 16 program control instructions.

The following table lists the program control instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
IF ELSE IEND	Branching	IF <i>(Conditional_expression)</i> ; <i>(Process_1)</i> ; ELSE; <i>(Process_2)</i> ; IEND;	Executes process 1 if the conditional expression is satisfied, or executes process 2 if the conditional expression is not satisfied.	○	○
WHILE WEND	Repetition	WHILE <i>(Conditional_expression)</i> ; ...; WEND;	Repeatedly executes the processes between WHILE and WEND as long as the conditional expression is satisfied.	○	○
WHILE WENDX	Repetition with One Scan Wait	WHILE <i>(Conditional_expression)</i> ; ...; WENDX;	Repeatedly executes the processes between WHILE and WENDX as long as the conditional expression is satisfied. Executes one loop process per scan.	○	○
PFORK JOINTO PJOINT	Parallel Execution	PFORK <i>Label_1, Label_2,</i> <i>Label_3...</i> ; <i>Label_1: Process_1;</i> JOINTO <i>Label_X;</i> <i>Label_2: Process_2;</i> JOINTO <i>Label_X;</i> <i>Label_3: Process_3;</i> JOINTO <i>Label_X;</i> <i>Label_X: PJOINT;</i>	Executes the blocks (forks) that are designated by the labels in parallel. The END and RET instructions cannot be used in parallel execution processing.	○	×
SFORK JOINTO SJOINT	Selective Execution	SFORK <i>Conditional_expression_1?</i> <i>Label_1,</i> <i>Conditional_expression_2?</i> <i>Label_2,</i> <i>Conditional_expression_3?</i> <i>Label_3,</i> <i>Conditional_expression_4?</i> <i>Label_4;</i> <i>Label_1: Process_1;</i> JOINTO <i>Label_X;</i> <i>Label_2: Process_2;</i> JOINTO <i>Label_X;</i> <i>Label_3: Process_3;</i> JOINTO <i>Label_X;</i> <i>Label_4: Process_4;</i> JOINTO <i>Label_X;</i> ...; <i>Label_X: SJOINT;</i>	Executes process 1 if conditional expression 1 is satisfied, or executes process 2 if conditional expression 2 is satisfied.	○	○

Continued on next page.

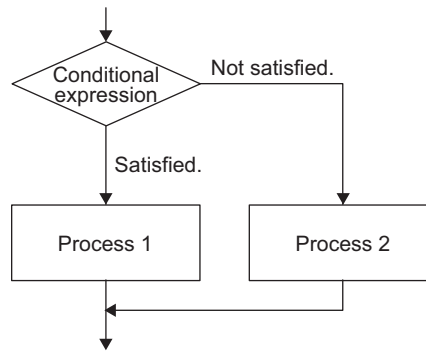
Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
MSEE	Call Subprogram	MSEE MPS□□□;	Executes the MPS□□□ subprogram.	○	×
SSEE	Call Sequence Subprogram	SSEE SPS□□□;	Executes the SPS□□□ subprogram.	×	○
UFC	Call User Function	UFC <i>User_function_name</i> <i>Input_data</i> , <i>Input_address</i> , <i>Output_data</i> ;	Calls a user-created function from the motion program.	○	×
FUNC	User Function	FUNC <i>User_function_name</i> <i>Input_data</i> , <i>Input_address</i> , <i>Output_data</i> ;	Calls a user-created function from the sequence program.	×	○
END	Program End	END;	Ends the program.	○	○
RET	Subprogram Return	RET;	Ends the subprogram.	○	○
TIM	Dwell Time	TIM T – ;	Waits for the period of time specified by T, and then proceeds to the next block.	○	×
TIM1MS	One-ms Dwell Time	TIM1MS T – ;	Waits for the period of time specified by T, and then proceeds to the next block.	○	×
IOW	I/O Variable Wait	IOW MB – = = ...;	Stops execution of the motion program until the conditional expression is satisfied.	○	×
EOX	One Scan Wait	EOX;	Divides the execution of consecutive sequence instructions. The instruction block after EOX is executed in the next scan.	○	×
SNGD/ SNGE	Disable Single-block Signal (SNGD) and Enable Single-block Signal (SNGE)	SNGD; ...; SNGE;	Specifies whether to enable or disable single step operation during debugging.	○	×

Branching Instructions (IF, ELSE, and IEND)

The IF, ELSE, and IEND instructions execute the blocks between IF and ELSE when a conditional expression is satisfied. If the conditional expression is not satisfied, the blocks between ELSE and IEND are executed.

ELSE can be omitted. If ELSE is omitted and the conditional expression is not satisfied, execution will continue from the block after IEND.



Information The IF, ELSE, and IEND instructions can be nested to up to 8 levels.

Format

The format of the IF, ELSE, and IEND instructions is as follows:

```
IF (Conditional_expression);
... (Process_1)
ELSE;
... (Process_2)
IEND;
```

The conditional expressions that can be used in branching instructions are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
IF MB000000 == 0; "MB000000 = 0
IF MB000000 == 1; "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
IF (MB000000 & MB000001) == 1; "MB000000 = 1 and MB000001 = 1
IF (MB000000 & !MB000001) == 1; "MB000000 = 1 and MB000001 = 0
IF (MB000000 | MB000001) == 1; "MB000000 = 1 or MB000001 = 1
IF (MB000000 | !MB000001) == 1; "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
IF MB000000 <> 0;           ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
IF 1 == MB000000;          ⇒ Syntax error
IF MB000000 == MB000001;  ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
IF MB000000;              ⇒ Syntax error
IF (0);                   ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
IF (MB000000 == 0) & (MB000001 == 1); ⇒ Syntax error
```

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (`==`, `<>`, `>`, `<`, `>=`, `<=`) can be used for these data types.

Specify a register on either the left or the right side.

```
IF MW00000 == 3;           "MW00000 = 3
IF ML00000 <> ML00002;    "ML00000 ≠ ML00002
IF 1.23456 >= MF00000;    "1.23456 ≥ MF00000
```

■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

```
IF MW00000 == (MW00001/3);      "MW00000 = (MW00001 ÷ 3)
IF (ML00000 & F0000000H) <> ML00002; "(ML00000 ∧ F0000000H) ≠ ML00002
IF 1.23456 >= (MF00000 * MF00002); "1.23456 ≥ (MF00000 × MF00002)
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

```
IF 0 == 3;                    ⇒ Syntax error
IF (3.14 * 2 * 1000) > 9000.0; ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
IF MW000000;                 ⇒ Syntax error
IF (-1);                      ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
IF (MW00000 < 0) & (MW000001 > 0); ⇒ Syntax error
```

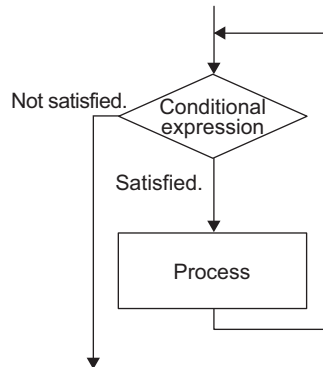
Programming Example

A programming example that uses the IF, ELSE, and IEND instructions is given below.

```
IF MB000000 == 1;
MOV [A1] 10000;           "If MB000000 is ON, A1 starts positioning.
ELSE;
MOV [B1] 10000;           "If MB000000 is OFF, B1 starts positioning.
IEND;
```

Repetition Instructions (WHILE, WEND)

Use the WHILE and WEND instructions to repeatedly execute the instruction blocks between the WHILE and WEND instructions as long as the conditional expression is satisfied. When the conditional expression is no longer satisfied, execution jumps to the next block after WEND.




Important

If the repeated program section is created using only instructions for which processing is completed in one scan, the Machine Controller may be overloaded by the scan processing, resulting in exceeding the scan time or a watchdog timer error.

For instructions that are executed in one scan, use the WHILE and WENDX instructions instead or insert an EOX or TIM instruction inside the repeated program section.

Refer to the following section for details on instructions that are executed in one scan.

 5.4 *Instruction Types and Execution Scans* (page 5-13)

Information The WHILE and WEND instructions can be nested to up to 8 levels.

Format

The format for WHILE and WEND instructions is as follows:

```

WHILE (Conditional_expression);
...;
(Process);
...;
WEND ;    "End of repetition instructions
  
```

The conditional expressions that can be used in repetition instructions are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
WHILE MB000000 == 0; "MB000000 = 0
WHILE MB000000 == 1; "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
WHILE (MB000000 & MB000001) == 1; "MB000000 = 1 and MB000001 = 1
WHILE (MB000000 & !MB000001) == 1; "MB000000 = 1 and MB000001 = 0
WHILE (MB000000 | MB000001) == 1; "MB000000 = 1 or MB000001 = 1
WHILE (MB000000 | !MB000001) == 1; "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
WHILE MB000000 <> 0; ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
WHILE 1 == MB000000; ⇒ Syntax error
WHILE MB000000 == MB000001; ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
WHILE MB000000; ⇒ Syntax error
WHILE (0); ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
WHILE (MB000000 == 0) & (MB000001 == 1); ⇒ Syntax error
```

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (==, <>, >, <, >=, <=) can be used for these data types.

Specify a register on either the left or the right side.

```
WHILE MW000000 == 3; "MW000000 = 3
WHILE ML000000 <> ML000002; "ML000000 ≠ ML000002
WHILE 1.23456 >= MF000000; "1.23456 ≥ MF000000
```

■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

WHILE MW00000 = = (MW00001/3);	"MW00000 = (MW00001 ÷ 3)
WHILE (ML00000 & F0000000H) <> ML00002;	"(ML00000 \wedge F0000000H) \neq ML00002
WHILE 1.23456 >= (MF00000 * MF00002);	"1.23456 \geq (MF00000 \times MF00002)

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

WHILE 0 = = 3;	\Rightarrow Syntax error
WHILE (3.14 * 2 * 1000) > 9000.0;	\Rightarrow Syntax error

- When there is no numeric comparison instruction:

WHILE MW000000;	\Rightarrow Syntax error
WHILE (-1);	\Rightarrow Syntax error

- When more than one numeric comparison instruction is used:

WHILE (MW00000 < 0) & (MW000001 > 0);	\Rightarrow Syntax error
---------------------------------------	----------------------------

Programming Example

The following programming example uses the WHILE and WEND instruction to draw a circle ten times.

```

MOV [A1] 0 [B1] 0;"Positioning
MW00100 = 1;"Preset counter.
INC; "Specify Incremental Mode.
PLN [A1] [B1];"Set coordinate plane.
WHILE MW00100 <= 10 ;"Repetition instructions
  MCW [A1]0 [B1]0 U50. V50. F8000 ; "Circular interpolation
  MOV [A1]50. [B1]50.; "Positioning
  MW00100 = MW00100 + 1; "Increment counter
WEND ;"End of repetition instructions

```

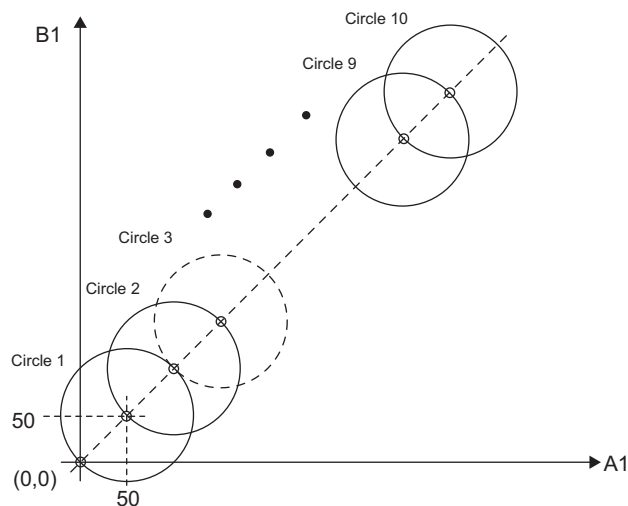
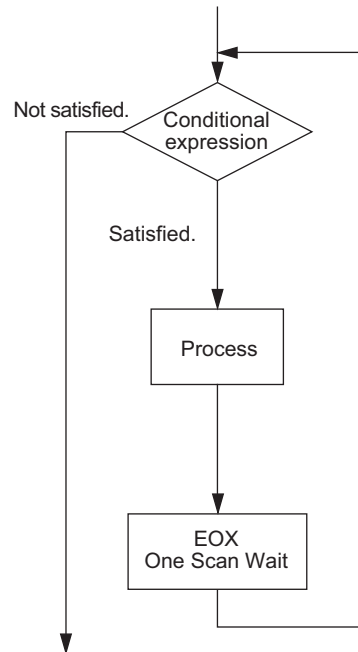


Fig. 6.62 Programming Example for the WHILE and WEND Instructions

Repetition with One Scan Wait (WHILE and WENDX)

The WHILE and WENDX instructions are effectively a combination of the WHILE, WEND, and EOX instructions. Use the WHILE and WENDX instructions to repeatedly execute the instruction blocks between the WHILE and WENDX instructions as long as the conditional expression is satisfied. When the conditional expression is no longer satisfied, execution jumps to the next block after the WENDX instruction.

Execution waits for one scan at the block before the WENDX instruction, then the processing for one scan and one loop is executed.



Format

The format for the WHILE and WENDX instructions is as follows:

```

WHILE (Conditional_expression);
  ...;
  (Process);
  ...;
WENDX;
  
```

"Wait for one scan, then end the repetition instruction.

The conditional expressions that can be used in repetition instructions are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
WHILE MB000000 == 0; "MB000000 = 0
WHILE MB000000 == 1; "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
WHILE (MB000000 & MB000001) == 1; "MB000000 = 1 and MB000001 = 1
WHILE (MB000000 & !MB000001) == 1; "MB000000 = 1 and MB000001 = 0
WHILE (MB000000 | MB000001) == 1; "MB000000 = 1 or MB000001 = 1
WHILE (MB000000 | !MB000001) == 1; "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
WHILE MB000000 <> 0; ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
WHILE 1 == MB000000; ⇒ Syntax error
WHILE MB000000 == MB000001; ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
WHILE MB000000; ⇒ Syntax error
WHILE (0); ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
WHILE (MB000000 == 0) & (MB000001 == 1); ⇒ Syntax error
```

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (==, <>, >, <, >=, <=) can be used for these data types.

Specify a register on either the left or the right side.

```
WHILE MW000000 == 3; "MW000000 = 3
WHILE ML000000 <> ML000002; "ML000000 ≠ ML000002
WHILE 1.23456 >= MF00000; "1.23456 ≥ MF00000
```


■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

WHILE MW00000 = (MW00001/3);	"MW00000 = (MW00001 ÷ 3)
WHILE (ML00000 & F0000000H) <> ML00002;	"(ML00000 ^ F0000000H) ≠ ML00002
WHILE 1.23456 >= (MF00000 * MF00002);	"1.23456 ≥ (MF00000 × MF00002)

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

WHILE 0 = = 3;	⇒ Syntax error
WHILE (3.14 * 2 * 1000) > 9000.0;	⇒ Syntax error

- When there is no numeric comparison instruction:

WHILE MW000000;	⇒ Syntax error
WHILE (-1);	⇒ Syntax error

- When more than one numeric comparison instruction is used:

WHILE (MW00000 < 0) & (MW000001 > 0);	⇒ Syntax error
---------------------------------------	----------------

Programming Example


A programming example that uses the WHILE and WENDX instructions is given below.

The following programming increments register ML00000 up to 100.

ML00000 = 0	
WHILE ML00000 == 100;	"Repetition instruction"
ML00000 = ML00000 + 1;	"Increment ML00000."
WENDX;	"Wait for one scan, then end the repetition instruction."
END;	

Parallel Execution Instructions (PFORK, JOINTO, and PJOINT)

The PFORK instruction performs parallel execution for blocks (i.e., forks) with the specified labels. After each fork has been executed, execution is merged at the label designated by the JOINTO instruction. A maximum of 8 forks (i.e., parallel processes) can be specified. Refer to the following section for details on labels.

 *Block Format (page 5-2)*

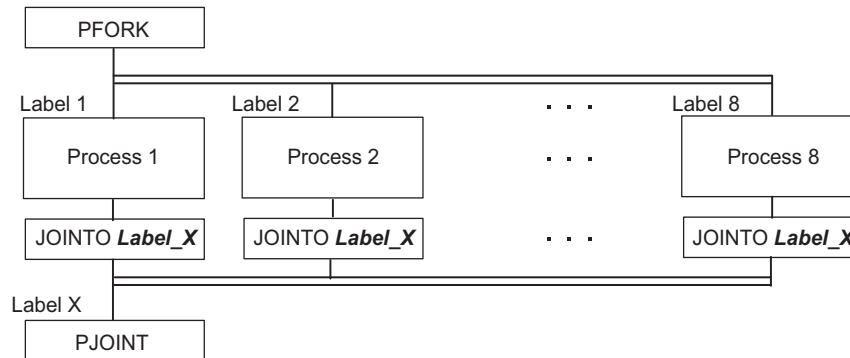


Fig. 6.63 Using the PFORK, JOINTO, and PJOINT Instructions

In the above figure, the labeled blocks specified by the PFORK instruction (Process 1, Process 2, Process 3, etc.) are executed in parallel. After each fork has been executed, execution is merged at the label designated by the JOINTO instruction. These instructions enable the designation of any combination of instructions for parallel execution, such as axis movement instructions and sequence instructions, or axis movement instructions and other axis movement instructions.

■ Instructions Designated before PFORK

Values set by instructions executed before a PFORK instruction such as FMX, ABS/INC, F reference, IFP, PLN, IAC/IDC, etc., are inherited by the forks that are executed in parallel by the parallel execution instruction. These instructions can also be executed within individual forks. After merging the forks, processing will continue using the values that were set in the leftmost process.

■ Nesting Parallel Execution Instructions in Subprograms

Do not use a parallel execution instruction in a subprogram that will be called from a parallel execution instruction in another subprogram.

■ Parallel Execution Instructions in Subprograms

The following restrictions apply to parallel execution instructions in subprograms.

- The parallel execution of up to eight processes is allowed in a subprogram.
The actual number of processes that can be executed in parallel depends on the parallel execution mode that was set in the main program.
A motion program alarm occurs if the maximum number of processes that can be executed in parallel is exceeded.
- The MSEE instruction can be used only in the block specified by the first label.

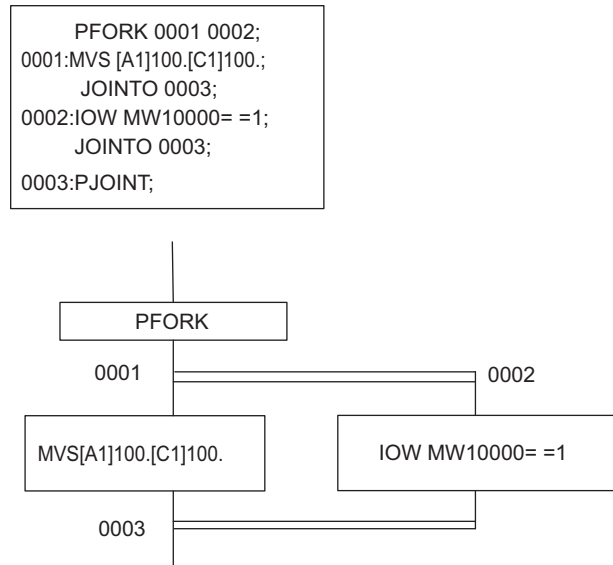


Fig. 6.64 Parallel Execution Instructions in Subprograms



Important

- An error (“Duplicate labels are defined”) will occur if the same label is used more than once in a program.
- If the number of PFORK forks and the number of labels are different, an error will occur.

Format

The format of the PFORK, JOINTO, and PJOINT instructions is as follows:

```
PFORK Label_1 Label_2 Label_3 .....
```

```
Label_1: Process_1
    JOINTO Label_X;
Label_2: Process_2
    JOINTO Label_X;
Label_3: Process_3
    JOINTO Label_X;
Label_X:PJOINT
```

Programming Example

A programming example that uses the PFORK, JOINTO, and PJOINT instructions is given below.

```

MOV [A1]100.[B1]150.;
MVS [A1]200.[B1]250. F1000;
PFORK 0001 0002 0003;
0001:MVS [A1]300.[B1]100.
      JOINTO 0004;
0002:MW12345=MW10000+MW10002;
      IOW MB120001= =1;
      JOINTO 0004;
0003:MVS [C1]100.[D1]100. F3000;
      JOINTO 0004;
0004:PJOINT;
      MOV [A1]500.[B1]500.[C1]500.;
      .
      .

```

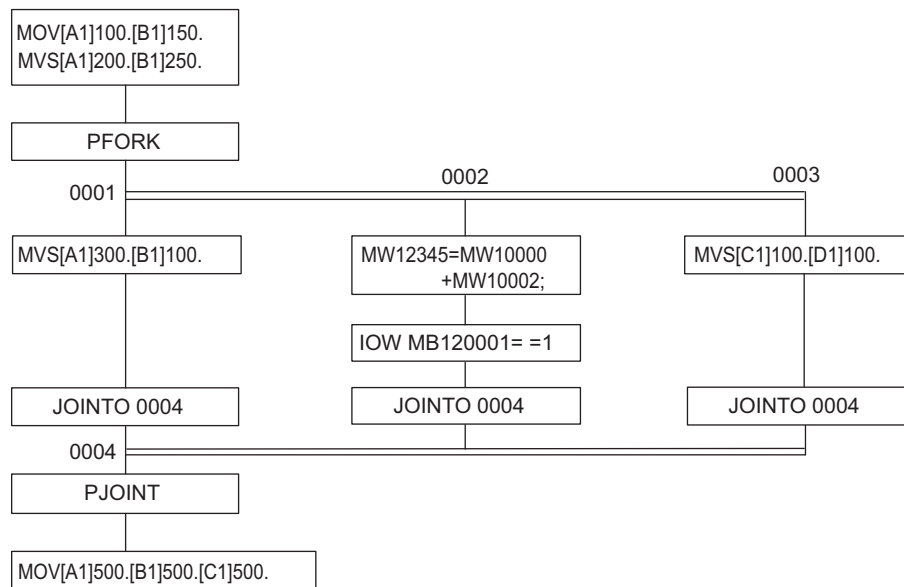


Fig. 6.65 Programming Example for the PFORK, JOINTO, and PJOINT Instructions

Selective Execution Instructions (SFORK, JOINTO, SJOINT)

The SFORK, JOINTO, and SJOINT instructions are used to execute a label following a “?” when the specified conditional expression is satisfied. After each process has been executed, execution is merged at the block with the label specified for the JOINTO instruction. Up to 16 conditional expressions including DEFAULT can be designated.

If not all of the designated conditional expressions are satisfied, the labeled block following DEFAULT? is executed.

DEFAULT can be specified only for the last conditional expression.

DEFAULT can be omitted in motion programs, but not in sequence programs.

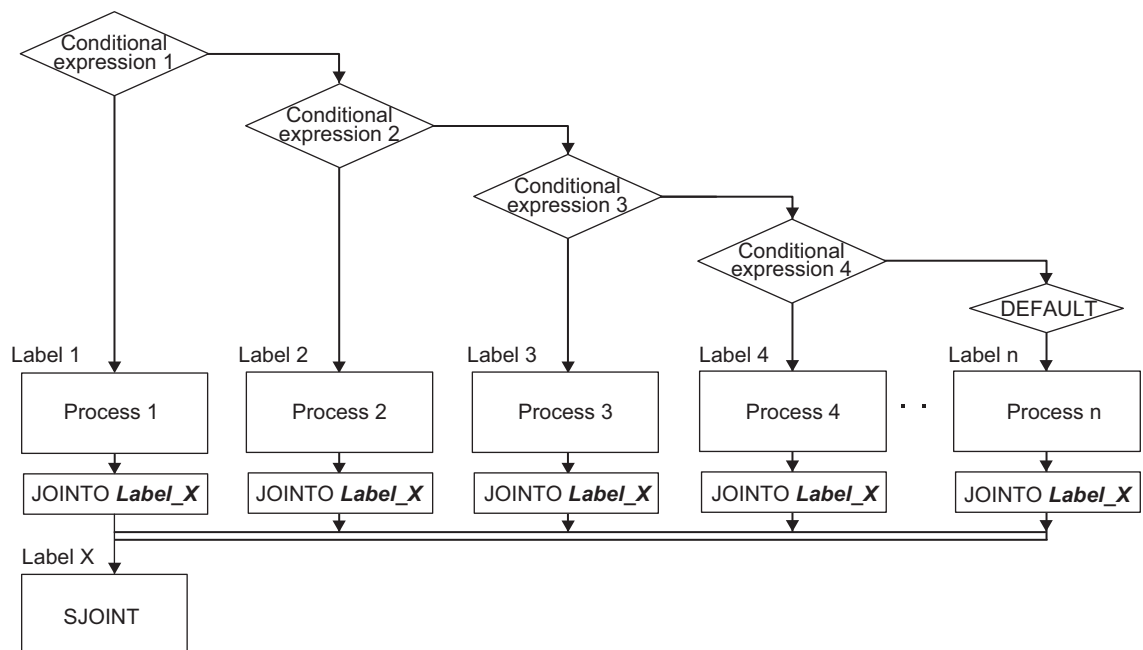


Fig. 6.66 Using the SFORK, JOINTO, and SJOINT Instructions

Information

1. The conditional expressions are examined in order from conditional expression 1. When more than one conditional expression is satisfied, processing is executed from the label that first satisfies the conditional expression.
2. Be sure to use a conditional expression that can actually be satisfied when you use SFORK in the motion program. If a condition is not satisfied, processing will remain in wait status at the SFORK instruction block until a condition is satisfied.

Format

The format of the SFORK, JOINTO, and SJOINT instructions is as follows:

```
SFORK Conditional_expression_1 ? Label_1, Conditional_expression_2 ? Label_2, Conditional_ex-  
pression_3 ? Label_3, Conditional_expression_4 ? Label_4,  
..., DEFAULT ? Label_n;
```

```
Label_1: Process_1  
    JOINTO Label_X
```

```
Label_2: Process_2  
    JOINTO Label_X
```

```
Label_3: Process_3  
    JOINTO Label_X
```

```
Label_4: Process_4  
    JOINTO Label_X
```

```
•  
•
```

```
Label_n: Process_n  
    JOINTO Label_X
```

```
Label_X:SJOINT
```

The conditional expressions that can be used with the SFORK instruction are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
MB000000 == 0? Label "MB000000 = 0
MB000000 == 1? Label "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
(MB000000 & MB000001) == 1? Label "MB000000 = 1 and MB000001 = 1
(MB000000 & !MB000001) == 1? Label "MB000000 = 1 and MB000001 = 0
(MB000000 | MB000001) == 1? Label "MB000000 = 1 or MB000001 = 1
(MB000000 | !MB000001) == 1? Label "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
MB000000 <> 0? Label ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
1 == MB000000 ? Label ⇒ Syntax error
MB000000 = = MB000001? Label ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
MB000000? Label ⇒ Syntax error
(0)? Label ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
(MB000000 = = 0) & (MB000001 = = 1)? Label ⇒ Syntax error
```

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (==, <>, >, <, >=, <=) can be used for these data types.

Specify a register on either the left or the right side.

```
MW000000 = = 3? Label "MW000000 = 3
ML000000 <> ML000002? Label "ML000000 ≠ ML000002
1.23456 >= MF00000? Label "1.23456 ≥ MF000000
```

■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

MW00000 = (MW00001/3)? <i>Label</i>	"MW00000 = (MW00001 ÷ 3)
(ML00000 & F0000000H) <> ML00002? <i>Label</i>	"(ML00000 ∧ F0000000H) ≠ ML00002
1.23456 >= (MF00000 * MF00002)? <i>Label</i>	"1.23456 ≥ (MF00000 × MF00002)

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

0 == 3 ? <i>Label</i>	⇒ Syntax error
(3.14 * 2 * 1000) > 9000.0? <i>Label</i>	⇒ Syntax error

- When there is no numeric comparison instruction:

MW000000? <i>Label</i>	⇒ Syntax error
(-1)? <i>Label</i>	⇒ Syntax error

- When more than one numeric comparison instruction is used:

(MW00000 < 0) & (MW000001 > 0)? <i>Label</i>	⇒ Syntax error
--	----------------

Programming Example

A programming example that uses the SFORK, JOINTO, and SJOINT instructions is given below.

```

MOV [A1]100.[B1]150.;
MVS [A1]200.[B1]250.F1000;
SFORK MW00100= =1 ? 0001,MW00100= =2 ? 0002,MW00100= =3 ? 0003,DEFAULT ? 0004;
0001:MVS [A1]300.[B1]100.F3000;
JOINTO 0005
0002:MVS [A1]300.[C1]100.F3000;
JOINTO 0005
0003:MVS [C1]300.[S]100.F3000;
JOINTO 0005
0004:JOINTO 0005;
0005:SJOINT;
MOV[A1]500.[B1]500.[C1]500.

```

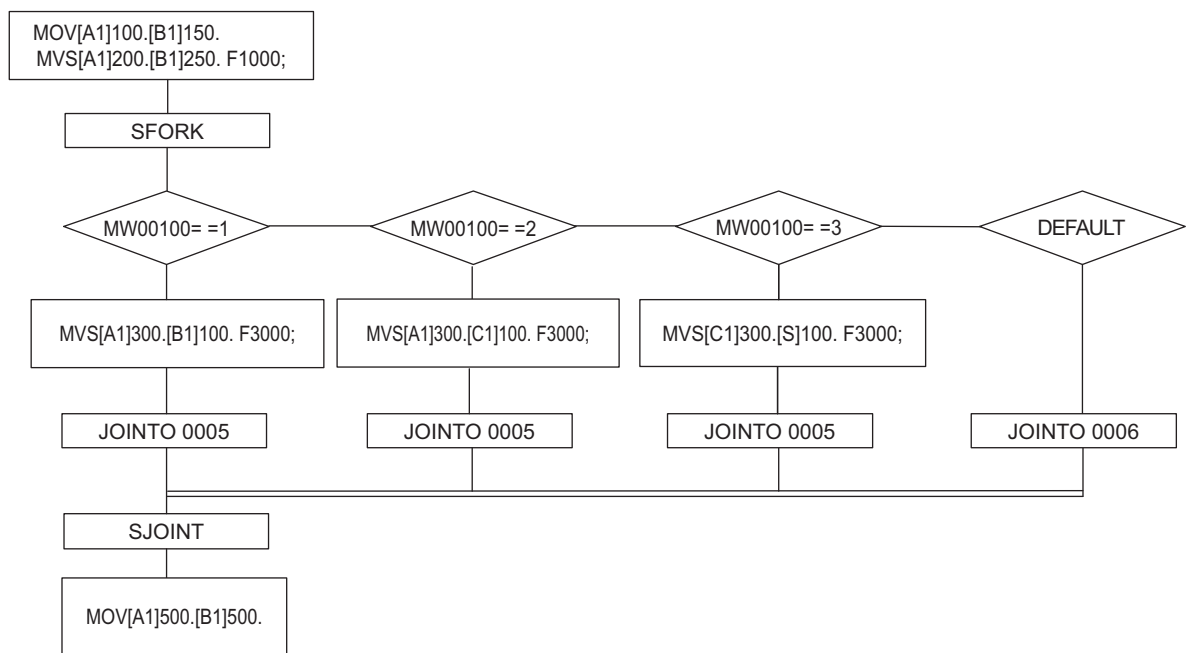


Fig. 6.67 Programming Example for the SFORK, JOINTO, and SJOINT Instructions

Call Motion Subprogram (MSEE)

The MSEE instruction is used in a motion program to call a subprogram that is stored in the motion program memory.

Up to 8 subprogram calls can be nested.

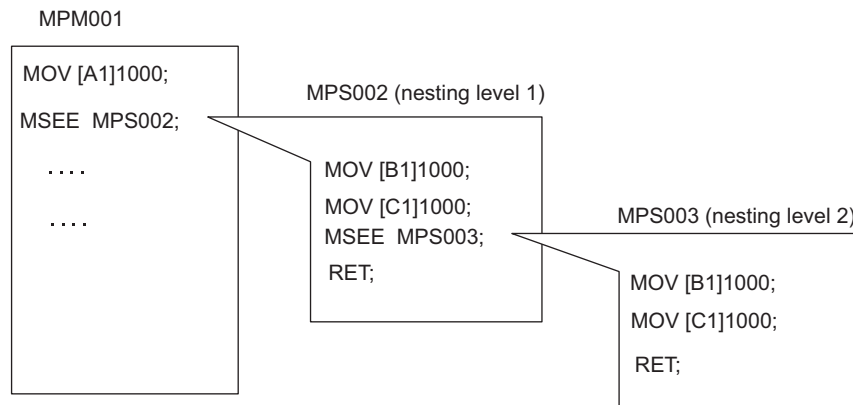


Fig. 6.68 Calling Subprograms

The RET instruction must be executed at the end of a subprogram.



Important

Subprogram Restrictions

If a main program is called by the MSEE instruction, the program will not be executed.

Format

The format of the MSEE instruction is as follows:

MSEE MPS *Subprogram_number*;

Item	Applicable Data
Subprogram number	Any number between 001 and 512

Programming Example

A programming example that uses the MSEE instruction to call motion subprogram MPS101 is given below.

```
MSEE MPS101;
```

Call Sequence Subprogram (SSEE)

The SSEE instruction is used in a sequence program to call a subprogram that is stored in the sequence program memory.

Up to 8 subprogram calls can be nested.

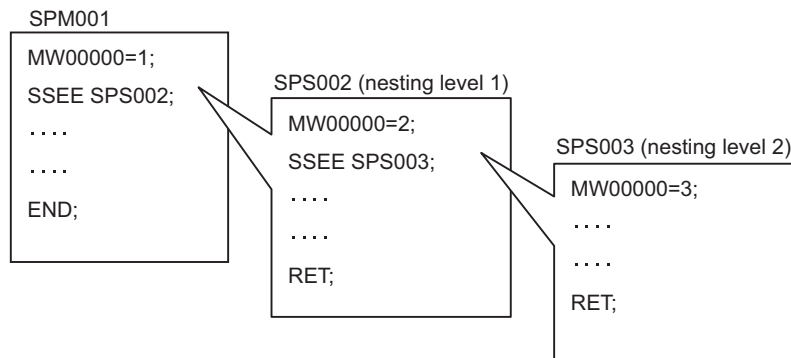


Fig. 6.69 Calling Subprograms

The RET instruction must be executed at the end of a subprogram.



Note

Subprogram Restrictions

The following restrictions apply to sequence programs in subprograms.

If a main program is called by the SSEE instruction, the program will not be executed.

Format

The format of the SSEE instruction is as follows:

SSEE SPS *Subprogram_number*;

Item	Applicable Data
Subprogram number	Any number between 001 and 512

Programming Example

A programming example that uses the SSEE instruction to call sequence subprogram SPS101 is given below.

```
SSEE SPS101;
```

Call User Function from Motion Program (UFC)

The UFC instruction is used in a motion program to call a user function.

When execution of the called user function is completed, the block after the UFC instruction block will be executed.



Important

The YB000000 output bit is used to determine if execution of a user function that was called from the motion program has been completed.

- If Execution of the User Function Is Completed When YB000000 Turns OFF
Execution of the user function is recognized as not being completed and the user function is called again in the next scan.
- If Execution of the User Function Is Completed When YB000000 Turns ON
Execution of the user function is recognized as being completed and the UFC instruction proceeds to the next block.

Format

The format of the UFC instruction is as follows:

UFC Function_name Input_data, Input_address, Output_data;

Item	Applicable Data
Function Name	ASCII, 8 bytes
Input data	Maximum: 16 data items (At least 1 data item is required.)
Input address	Maximum: 1 address
Output data*	Maximum: 16 data items (At least 1 data item is required.)

* You can omit the input address. The format “*Input_data, Output_data*” means that no input address is specified. At least one input data item and one output data item are required.

Programming Example

A programming example that uses the UFC instruction is given below.

UFC KANSUU MB000000 IW0010 MB000002, MA00100 ,
 Function Name Input data Input address
MB000001 MW00200 ML00201;
 Output data

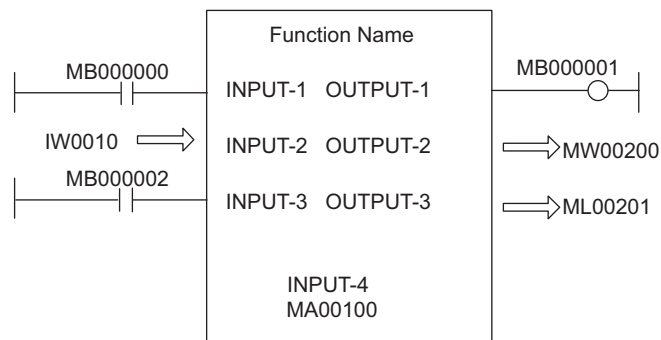
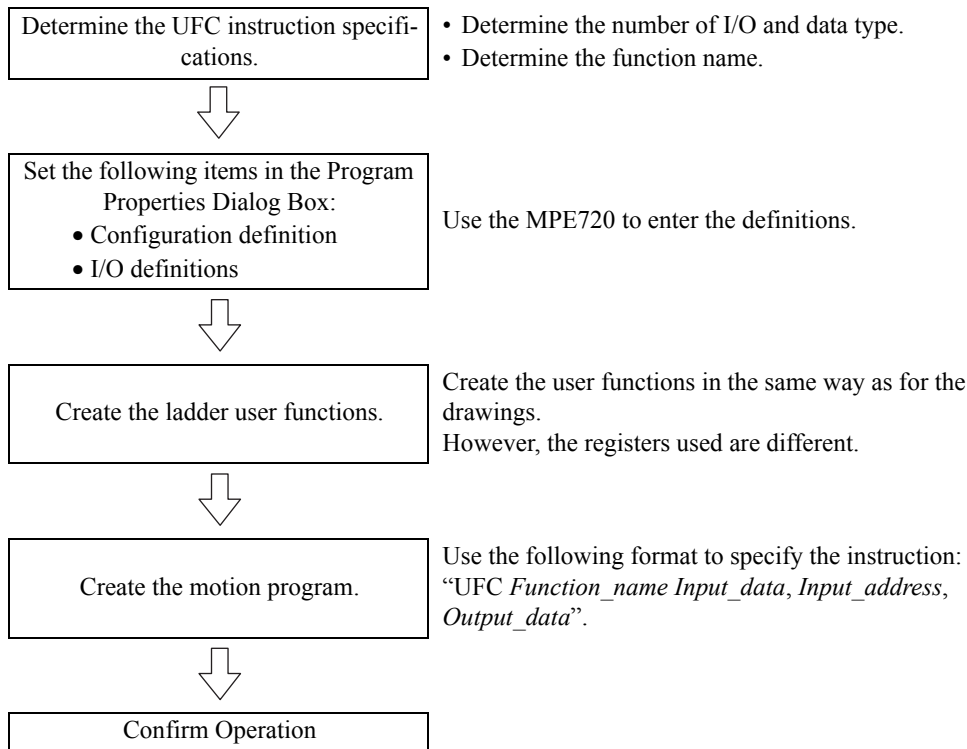


Fig. 6.70 Programming Example for the UFC Instruction

UFC Instruction Specification Procedure

The procedure for specifying the UFC instruction is given below.



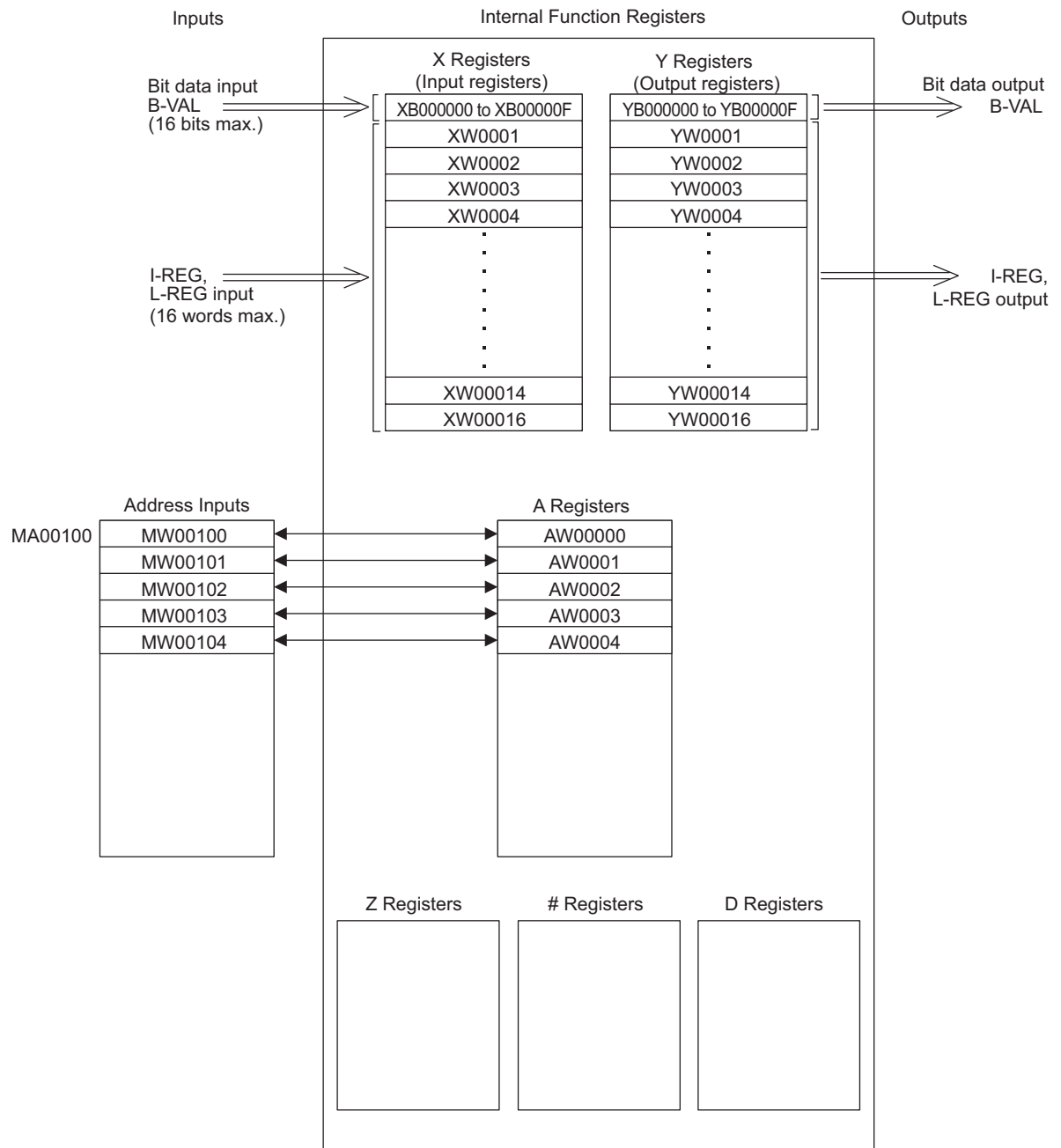
Data Types of Registers Used in User Functions

The following data types can be used.

Data Type	Type
B	Bit data
W	Integers
L	Double-length integers
Q	Quadruple-length integers
F	Real numbers
D	Double-length real numbers

Relationship between I/O Registers and Internal Function Registers

The relationship between the I/O registers specified in the UFC instruction and the function registers is shown below.



The 12 types of registers listed in the following table can be used in functions.

Table 6.2 Function Registers

Type	Name	Designation Method	Description	Features
X	Function Input Registers	XB,XW,XL,XQ,XF, XDnnnnn	These registers are used for inputs to functions. <ul style="list-style-type: none"> • Bit inputs: XB000000 to XB00000F • Integer inputs: XW00001 to XW00016 • Double-length integers: XL00001 to XL00015 • Quadruple-length integers: XQ00001 to XQ00013 • Real numbers: XF00001 to XF00015 • Double-length real numbers: XD00001 to XD00013 	Function-specific
Y	Function Output Registers	YB,YW,YL,YQ,YF, Ydnnnnn	These registers are used for inputs to functions. <ul style="list-style-type: none"> • Bit inputs: YB000000 to YB00000F • Integer inputs: YW00001 to YW00016 • Double-length integers: YL00001 to YL00015 • Quadruple-length integers: YQ00001 to YQ00013 • Real numbers: YF00001 to YF00015 • Double-length real numbers: YD00001 to YD00013 	
Z	Function Internal Registers	ZB,ZW,ZL,ZQ,ZF, ZDnnnnn	These are internal registers that are unique within each function. These registers are used for internal processing in functions.	
A	Function External Registers	AB,AW,AL,AQ,AF, ADnnnnn	These are external registers that use the address input value as the base address. For linking with S, M, I, O, #, and DANnnnn. Register address nnnnn is a decimal number.	
#	# Registers	#B,#W,#L,#Q,#F, #Dnnnnn	These registers are read-only in programs. These registers can be referenced only from the corresponding drawing. The actual usable range is specified by the user from the MPE720. Register address nnnnn is a decimal number.	
D	D Registers	DB,DW,DL,DQ,DF, DDnnnnn	These registers are unique to each drawing. These registers can be referenced only from the corresponding drawing. The actual usable range is specified by the user from the MPE720. Register address nnnnn is a decimal number.	
S	System Registers	SB,SW,SL,SQ,SF, SDnnnnn	Same as DWG registers. These registers are used for both drawings and functions. Care must be taken when using them to reference the same function from drawings with different priority levels. Register address nnnnn is a decimal number. Register address hhhhh is a hexadecimal number.	Shared by All Drawings
M	Data Registers	MB,MW,ML,MQ,MF, MDnnnnnnn		
G	Registers	GB,GW,GL,GQ,GF, GDnnnnnnn		
I	Input Registers	IB,IW,IL,IQ,IF, IDhhhhh		
O	Output Registers	OB,OW,OL,OQ,OF, ODhhhhh		
C	Constant register	CB,CW,CL,CQ,CF, CDnnnnn		

Note: SA, MA, IA, OA, DA, #A, and CA can also be used inside functions.

The following example shows the data transfer between I/O registers.

Motion Program Notation

UFC TESTFUNC DB000000 DB000001 MW00030 MW00032, MA00100,DB000002 MW00040

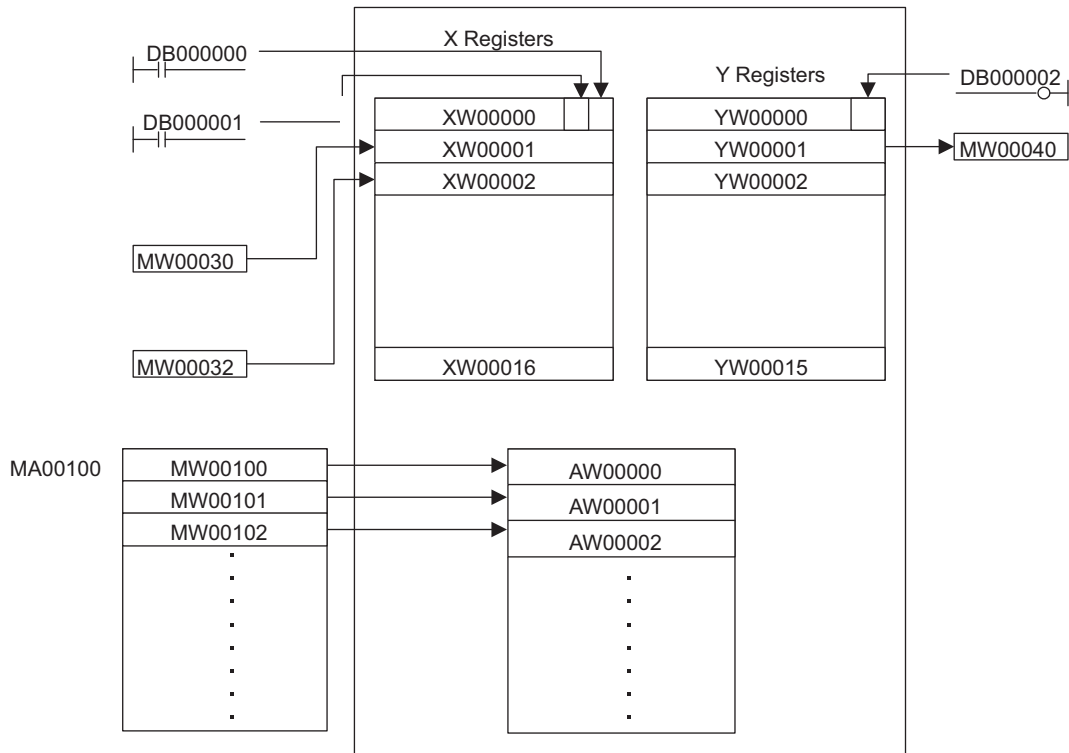


Fig. 6.71 Motion Program Notation

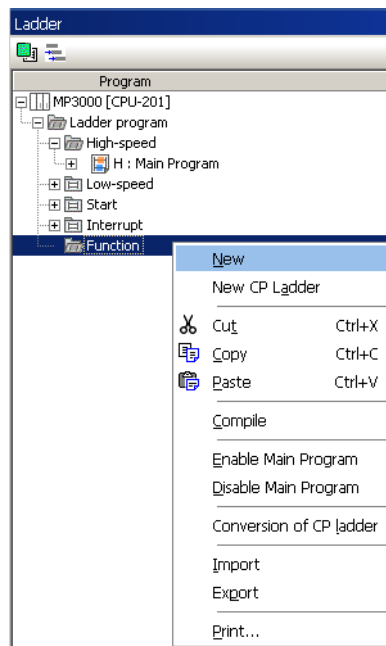
Creating User Functions

The procedure used to create user functions is demonstrated here with the following user function specifications as an example.

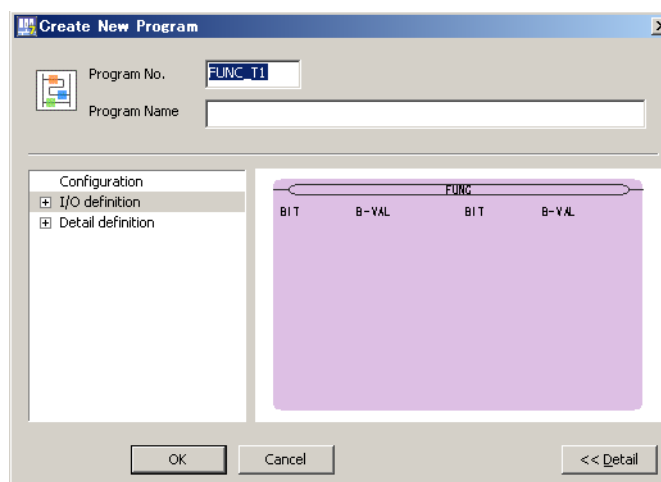
Specification	Motion Program
Specify the servo axis number and speed data and set this information in the OL□□□10 setting parameter.	MW00030 = Servo axis number (1 or 2) ML00032 = Feed forward speed UFC FUNC-T1 MW00030 ML00032,,DB000001;

Use the following procedure to create the user function.

1. Open the Ladder Pane. Right-click **Function** under **Ladder Program** and select **New** from the menu.

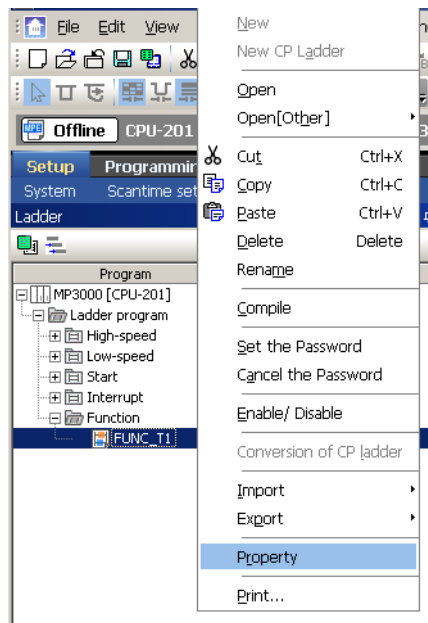


2. Enter FUNC-T1 for the *Program Number* in the Create New Program Dialog Box.



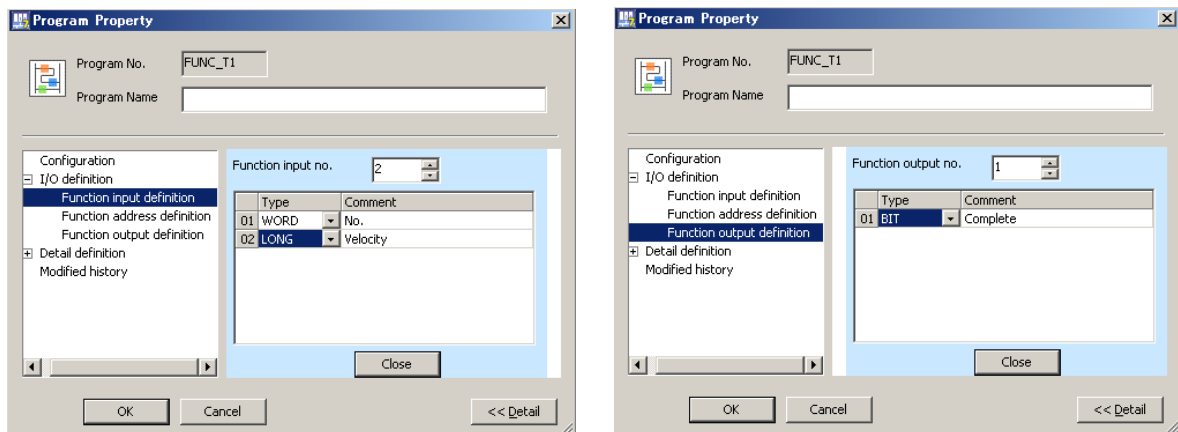
The Ladder Pane is displayed.

- Right-click **FUNC-T1** and select **Property** from the menu.

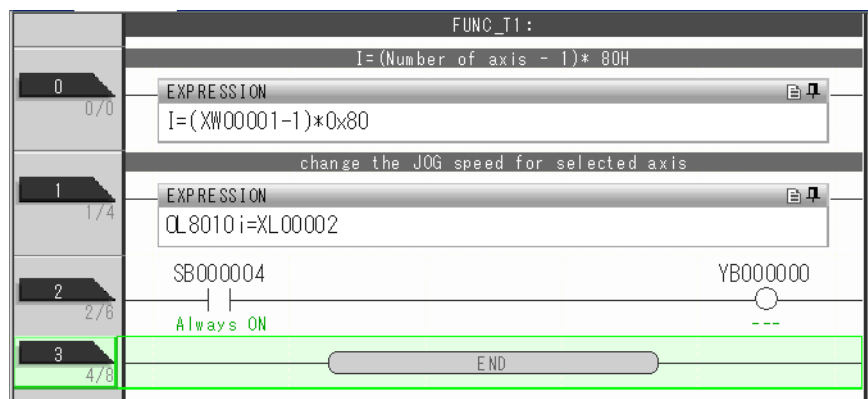


- In the Program Properties Dialog Box, click **Function input definition** and **Function output definition** under **I/O Definitions** and set the number of function inputs and outputs and their data types.

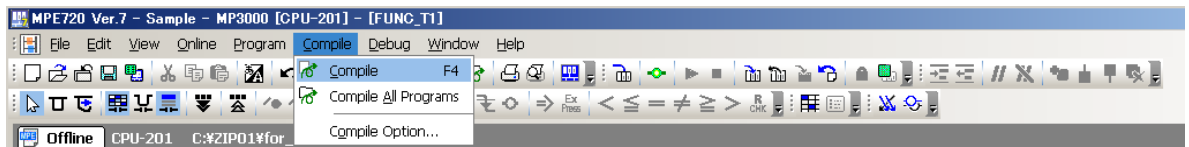
The code “UFC FUNC-T1 MW00030 ML00032, ,DB000001;” produces the following settings.



- Close the DWG Configuration Definition Tab Page and edit the user function program in the Ladder Pane.



6. Select **Compile** – **Compile** from the menu bar.



7. Create a program in the Motion Editor that calls the user function.

```
MW00030 = 1;  
ML00032 = 500;  
UFC FUNC-T1 MW00030 ML00032, , DB000001;  
END;
```

This concludes the process to create a user function that is called from the motion program.
Execute the motion program and check the operation.

Program End (END)

The END instruction ends program operation.

No other instructions can be executed in the same block as an END instruction.

The program ends operation after execution of the block containing the END instruction has been completed.

If there is a movement instruction in the previous block, the program operation ends after the in-position check is completed.

Format

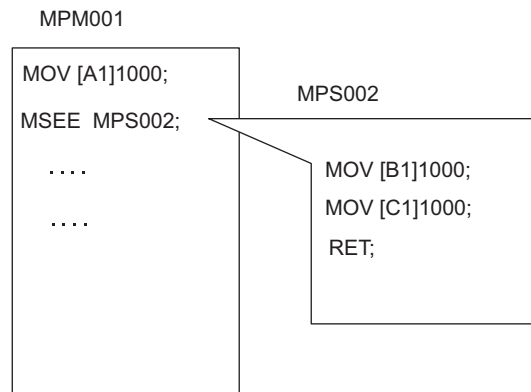
The format of the END instruction is as follows:

```
END;
```

Subprogram Return (RET)

The RET instruction ends a subprogram.

After operation of the called subprogram is ended with the RET instruction, execution proceeds to the block after the MSEE or SSEE instruction in the main program or subprogram that called the subprogram.



Format

The format of the RET instruction is as follows:

```
RET;
```

Dwell Time (TIM)

The TIM instruction causes execution to pause for a specified period of time before the execution of the next block begins.

Format

The format of the TIM instruction is as follows:

```
TIM Twait_time;
```

Item	Unit	Applicable Data	Setting Range [s]
Wait time	0.01s	Directly designated value	0.00 to 600.00
		Indirect designation with an integer	0.00 to 327.67

Programming Example

A programming example that uses the TIM instruction is given below.

```
MOV [A1]100;  
TIM T250 ;
```

The TIM instruction is executed after positioning is completed.

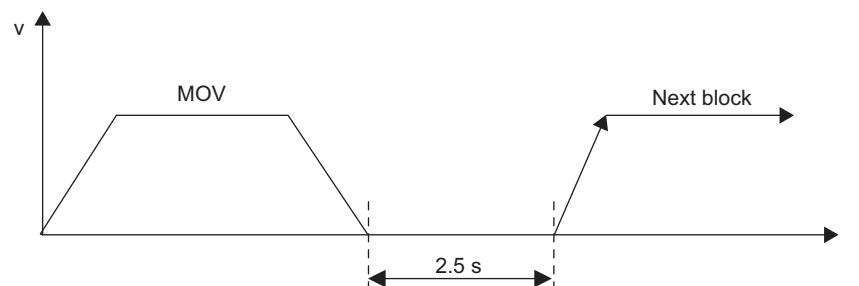


Fig. 6.73 Programming Example for the TIM Instruction

Dwell Time (TIM1MS)

The TIM1MS instruction causes execution to pause for a specified period of time before the execution of the next block begins.

The unit for the time is 1 ms.

Format

The format of the TIM1MS instruction is as follows:

```
TIM1MS Twait_time;
```

Item	Unit	Applicable Data	Setting Range [s]
Wait time	1ms	Directly designated value	0.000 to 60.000
		Integer registers (excluding # and C registers)	0.000 to 32.767

Programming Example

A programming example that uses the TIM1MS instruction is given below.

```
MOV [A1]1000;"Positioning
TIM1MS T5;"Wait for 5 ms after positioning is completed.
MOV [A1]1000;"Positioning
END;
```

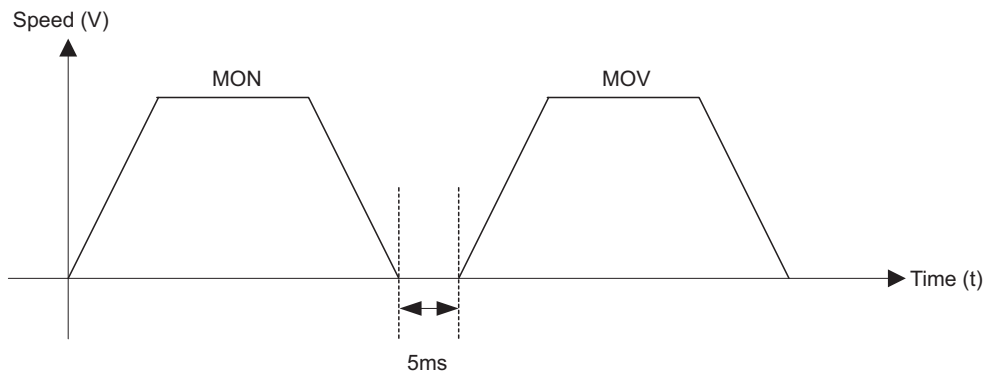


Fig. 6.74 Programming Example for TIM1MS Instruction

I/O Variable Wait (IOW)

The IOW instruction waits until the status specified by the conditional expression is satisfied, and then execution proceeds to the next block.

Format

The format of the IOW instruction is as follows:

```
IOW  IB00001&IB00002 == 1;
      ①
```

Description	Application	Applicable Data
①	Conditional expression	<ul style="list-style-type: none"> • All integer, double-length integer, or real number registers (excluding # and C registers) • Same as above except with a subscript. • Subscript registers • Constant

The conditional expressions that can be used with the IOW instruction are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
IOW MB000000 == 0;      "MB000000 = 0
IOW MB000000 == 1;      "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
IOW (MB000000 & MB000001) == 1;  "MB000000 = 1 and MB000001 = 1
IOW (MB000000 & !MB000001) == 1;  "MB000000 = 1 and MB000001 = 0
IOW (MB000000 | MB000001) == 1;  "MB000000 = 1 or MB000001 = 1
IOW (MB000000 | !MB000001) == 1;  "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
IOW MB000000 <> 0;          ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
IOW 1 == MB000000;          ⇒ Syntax error
IOW MB000000 == MB000001;   ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
IOW MB000000;              ⇒ Syntax error
IOW (0);                    ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

IOW (MB000000 = = 0) & (MB000001 = = 1); ⇒ Syntax error

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (==, <>, >, <, >=, <=) can be used for these data types.

Specify a register on either the left or the right side.

IOW MW000000 = = 3;	"MW000000 = 3
IOW ML000000 <> ML000002;	"ML000000 ≠ ML000002
IOW 1.23456 >= MF000000;	"1.23456 ≥ MF000000

■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

IOW MW000000 = = (MW000001/3);	"MW000000 = (MW000001 ÷ 3)
IOW (ML000000 & F0000000H) <> ML000002;	"(ML000000 ∧ F0000000H) ≠ ML000002
IOW 1.23456 >= (MF000000 * MF000002);	"1.23456 ≥ (MF000000 × MF000002)

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

IOW 0 = = 3;	⇒ Syntax error
IOW (3.14 * 2 * 1000) > 9000.0;	⇒ Syntax error

- When there is no numeric comparison instruction:

IOW MW0000000;	⇒ Syntax error
IOW (-1);	⇒ Syntax error

- When more than one numeric comparison instruction is used:

IOW (MW000000 < 0) & (MW000001 > 0); ⇒ Syntax error

Programming Example

A programming example that uses the IOW instruction is given below.

```
IOW (MB001001&MB001002) = = 1;  
MOV [A1]1000;
```

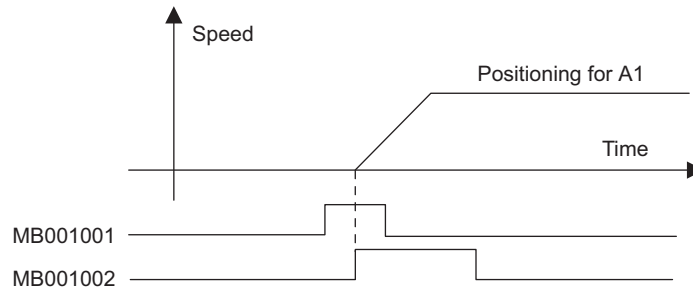


Fig. 6.75 Programming Example for IOW Instruction

One Scan Wait (EOX)

The EOX instruction causes program execution to wait for one scan.
The block after the EOX instruction is executed in the next scan.

Format

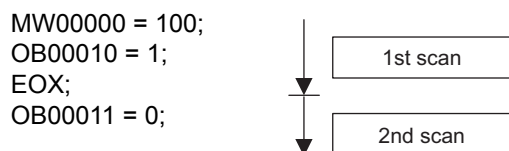
The format of the EOX instruction is as follows:

```
EOX;
```

Programming Example

A programming example that uses the EOX instruction is given below.

- Used with Sequence Instructions



- Used with a WHILE Instruction

```
WHILE OB00010 = = 1;  
EOX;  
WEND;
```

Disable Single-block Signal (SNGD) and Enable Single-block Signal (SNGE)

The SNGD and SNGE instructions are used to specify whether to disable or enable single-step operation in Debug Mode.

The blocks between the SNGD and SNGE instructions are executed continuously without single-block stops, regardless of the single-block operation mode setting.



Terms

Single-block Operation Mode

In single-block operation mode, a stop is executed for each block.

Format

The format of the SNGD instruction is as follows:

```
SNGD;
The code you want to execute continuously without stopping
SNGE;
```

Programming Example

A programming example that uses the SNGD and SNGE instructions is given below.

In this example, blocks 1 to 3 between the SNGD and SNGE instructions are executed continuously without single-block stops, even in single-block operation mode.

```
MVS [A1]0 [B1]0;
SNGD;
MVS [A1]100 [B1]200; "①"
MB000101 = 1;        "②"
MB000102 = 1;        "③"
SNGE;
MB000103 = 1;
```

6.5 Numeric Operation Instructions


There are eight numeric operation instructions. You can use these instructions in motion programs or in sequence programs.

The following table lists the numeric operation instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
=	Substitute	$Result = Math_expression$	Substitutes the results of an operation. Calculations are performed from left to right with no order of priority.	<input type="radio"/>	<input type="radio"/>
+	Add	$MW\Box = MW\Box + MW\Box;$	Performs integer and real number addition. If both integers and real numbers are included, calculations are performed with real numbers.	<input type="radio"/>	<input type="radio"/>
-	Subtract	$MW\Box = MW\Box - MW\Box;$	Performs integer and real number subtraction. If both integers and real numbers are included, calculations are performed with real numbers.	<input type="radio"/>	<input type="radio"/>
++	Extended Add	$MW\Box = MW\Box ++ MW\Box;$	Performs extended addition of integers.	<input type="radio"/>	<input type="radio"/>
--	Extended Subtract	$MW\Box = MW\Box -- MW\Box;$	Performs extended subtraction of integers.	<input type="radio"/>	<input type="radio"/>
*	Multiply	$MW\Box = MW\Box * MW\Box;$	Performs integer and real number multiplication. If both integers and real numbers are included, calculations are performed with real numbers.	<input type="radio"/>	<input type="radio"/>
/	Divide	$MW\Box = MW\Box / MW\Box;$	Performs integer and real number division. If both integers and real numbers are included, calculations are performed with real numbers.	<input type="radio"/>	<input type="radio"/>
MOD	Modulo	$MW\Box = MW\Box / MW\Box;$ $MW\Box = MOD;$	When programmed in the next block after a division, MOD stores the remainder in the designated register.	<input type="radio"/>	<input type="radio"/>

Note: The \Box in the above formats indicates a register address.

Refer to the following section for details on the priority of numeric operations.

 5.3 Operation Priority Levels (page 5-11)

Substitute (=)

This instruction substitutes the operation result on the right side of the expression into the register on the left side.

Format

The format of the = (Substitute) instruction is as follows:

Result = Math expression;

① ②

Description	Application	Usable Registers
①	Result	<ul style="list-style-type: none"> All bit, integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Math expression	<ul style="list-style-type: none"> All bit, integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The = (Substitute) instruction can be used in motion programs, sequence programs, and ladder programs.

Programming examples that use the = (Substitute) instruction are given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	MB001000 = 1;	
W	MW00100 = 12345;	
L	ML00100 = 1234567;	
F	MF00100 = 1.2345;	
Q	MQ00100 = 123456789;	
D	MD00100 = 1.234567;	

Add (+)

The + (Add) instruction performs integer or real number addition on the right side and stores the result of that operation in the register on the left side. Constants can also be used instead of registers for the addition operation on the right side. If both integers and real numbers are included, the result is stored in the data type on the left side.

Format

The format of the + (Add) instruction is as follows:

$$\underbrace{\text{MW00101}}_{\textcircled{1}} = \underbrace{\text{MW00100}}_{\textcircled{2}} + \underbrace{12345}_{\textcircled{3}} ;$$

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant
③	Data to add	

Programming Examples

The + (Add) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the + (Add) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00101 = MW00100+12345;	
L	ML00106 = ML00102+ML00104;	
F	MF00202 = MF00200+1.23456;	

Information If an operation is performed with registers of different data types, the result is stored according to the data type on the left side. Refer to the following sections for details on data types.

Global Registers (page 4-5)

Local Registers (page 4-6)

Subtract (-)

The - (Subtract) instruction performs integer or real number subtraction on the right side and stores the result of that operation in the register on the left side. Constants can also be used instead of registers for the addition operation on the right side. If both integers and real numbers are included, the result is stored in the data type on the left side.

Format

The format of the - (Subtract) instruction is as follows:

$$\underset{\textcircled{1}}{\text{MW00101}} = \underset{\textcircled{2}}{\text{MW00100}} - \underset{\textcircled{3}}{\text{12345}} ;$$

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant
③	Data to subtract	

Programming Examples

The - (Subtract) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the - (Subtract) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	-	-
W	MW00101 = MW00100-12345;	
L	ML00106 = ML00102-ML00104;	
F	MF00202 = MF00200-1.23456;	

Extended Add (++)

The ++ (Extended Add) instruction adds integer values.

Overflows are not treated as operation errors. Instead, the calculation continues from the maximum value in the negative direction.

Underflows are not treated as operation errors. Instead, the calculation continues from the maximum value in the positive direction.

Otherwise, this instruction is the same as the + (Add) instruction.

■ Integers

Decimal: 0 → 1 ··· 32767 → -32768 ··· -1 → 0

Hexadecimal: 0000 → 0001 ··· 7FFF → 8000 ··· FFFF → 0000

■ Double-length Integers

Decimal: 0 → 1 ··· 2147483647 → -2147483648 ··· -1 → 0

Hexadecimal: 00000000 → 00000001 ··· 7FFFFFFF → 80000000 ··· FFFFFFFF → 00000000

■ Quadruple-length Integers

Decimal: 0 → 1 ··· 9223372036854775807 → -9223372036854775808 ··· -1 → 0

Hexadecimal: 0000000000000000 → 0000000000000001 ··· 7FFFFFFFFFFFFFFF →
8000000000000000 ··· FFFFFFFFFFFFFFFF → 0000000000000000

Format

The format of the ++ (Extended Add) instruction is as follows:

$$\underbrace{\text{MW00101}}_{\textcircled{1}} = \underbrace{\text{MW00100}}_{\textcircled{2}} + + \underbrace{12345}_{\textcircled{3}};$$

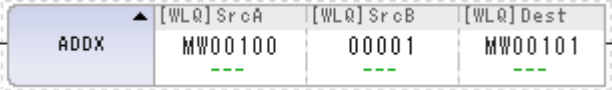


Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript.
③	Data to add	<ul style="list-style-type: none"> Subscript registers Constant

Note: A compiler error will occur if a real number is used.

Programming Examples

The ++ (Extended Add) instruction can be used in motion programs, sequence programs, and ladder programs.

Programming examples that use the ++ (Extended Add) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00101 = MW00100++ +1;	
L	ML00106 = ML00102++ +ML00104;	
F	–	–
Q	MQ00116 = MQ00108++ +MQ00112;	

Extended Subtract (--)

The -- (Extended Subtract) instruction subtracts integer values.

Overflows are not treated as operation errors. Instead, the calculation continues from the maximum value in the negative direction.

Underflows are not treated as operation errors. Instead, the calculation continues from the maximum value in the positive direction.

Otherwise, this instruction is the same as the - (Subtract) instruction.

■ Integers

Decimal: 0 → -1 ··· -32768 → 32767 ··· 1 → 0

Hexadecimal: 0000 → FFFF ··· 8000 → 7FFF ··· 0001 → 0000

■ Double-length Integers

Decimal: 0 → -1 ··· -2147483648 → 2147483647 ··· 1 → 0

Hexadecimal: 00000000 → FFFFFFFF ··· 80000000 → 7FFFFFFF ··· 00000001 → 00000000

■ Quadruple-length Integers

Decimal: 0 → -1 ··· -9223372036854775808 → 9223372036854775807 ··· 1 → 0

Hexadecimal: 0000000000000000 → FFFFFFFFFFFFFFFF ··· 8000000000000000 →
7FFFFFFFFFFFFFFF ··· 0000000000000001 → 0000000000000000

Format

The format of the -- (Extended Subtract) instruction is as follows:

$$\underbrace{\text{MW00101}}_{\textcircled{1}} = \underbrace{\text{MW00100}}_{\textcircled{2}} \text{ -- } \underbrace{\text{12345}}_{\textcircled{3}};$$

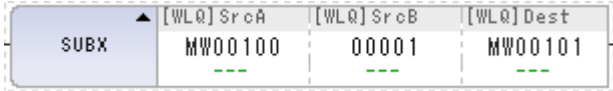
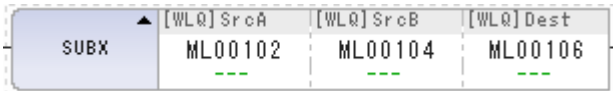

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript.
③	Data to add	<ul style="list-style-type: none"> Subscript registers Constant

Note: A compiler error will occur if a real number is used.

Programming Examples

The -- (Extended Subtract) instruction can be used in motion programs, sequence programs, and ladder programs.

Programming examples that use the -- (Extended Subtract) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	--	--
W	MW00101 = MW00100 - -1;	
L	ML00106 = ML00102 - -ML00104;	
F	--	--
Q	MQ00116 = MQ00108 - -MQ00112	

Multiply (*)

The * (Multiply) instruction performs integer or real number multiplication on the right side and stores the result of that operation in the register on the left side. Constants can also be used instead of registers for the multiplication operation on the right side. If both integers and real numbers are included, the result is stored in the data type on the left side.

Format

The format of the * (Multiply) instruction is as follows:

$$\textcircled{1} \quad \textcircled{2} \quad \textcircled{3}$$

$$\underline{\text{MW00101}} = \underline{\text{MW00100}} * \underline{\text{12345}} ;$$

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant
③	Data to multiply	

Programming Examples

The * (Multiply) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the * (Multiply) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00102 = MW00100 * MW00101	
L	ML00106 = ML00102 * ML00104;	
F	MF00202 = MF00200 * 1.23456;	

Divide (/)

The / (Divide) instruction performs integer or real number division on the right side and stores the result of that operation in the register on the left side. Constants can also be used instead of registers for the division operation on the right side. If both integers and real numbers are included, the result is stored in the data type on the left side.

Format

The format of the / (Divide) instruction is as follows:

$$\underset{\textcircled{1}}{\text{MW00101}} = \underset{\textcircled{2}}{\text{MW00100}} / \underset{\textcircled{3}}{\text{12345}} ;$$

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers)
③	Data to divide	<ul style="list-style-type: none"> Same as above except with a subscript. Subscript registers Constant

Programming Examples

The / (Divide) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the / (Divide) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00102 = MW00100/MW00101;	
L	ML00106 = ML00102/ML00104;	
F	MF00202 = MF00200/1.23456;	

Modulo (MOD)

When the MOD instruction is specified in the next block after a division instruction, the remainder of the division operation is stored in the specified variable.

Format

The format of the MOD instruction is as follows:

MW00001 = 1000 / 999;

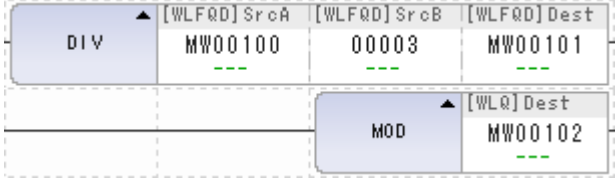
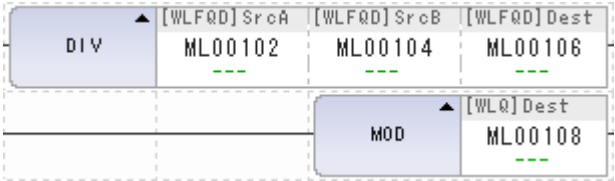
MW00002 = MOD;

①

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All registers with integer and double-length integer data types (excluding # and C registers) Same as above except with a subscript. Subscript registers

Programming Examples

The MOD instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the MOD instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00101 = MW00100/3; MW00102 = MOD;	
L	ML00106 = ML00102/ML00104; ML00108 = MOD;	
F	–	–

Example Programming Example for the MOD Instruction (Double-length Integers)

ML00106 = ML00100 * ML00102/ML00104;

(173575) (100000) (60000) (34567)

ML00108 = MOD;

(32975)



Important

The MOD instruction must always be executed immediately after the division instruction. If it is not executed in the next block after the division instruction, the result will not be reliable.

6.6

Logic Operation Instructions

Logic operation instructions are used to perform logical TRUE or FALSE operations on numbers. There are four logic operation instructions. You can use these instructions in motion programs or in sequence programs.


The following table lists the logic operation instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
	OR (Inclusive OR)	$MB\Box = MB\Box MB\Box;$ $MB\Box = MB\Box 1;$ $MW\Box = MW\Box MW\Box;$ $MW\Box = MW\Box 00FFH;$ $ML\Box = ML\Box ML\Box;$ $ML\Box = ML\Box 00FF00FFH;$ $MQ\Box = MQ\Box MQ\Box;$ $MQ\Box = MQ\Box 00FF00FF 00FF00FFH;$	Performs a bit or integer inclusive OR operation.	○	○
&	AND (AND)	$MB\Box = MB\Box \& MB\Box;$ $MB\Box = MB\Box \& 1;$ $MW\Box = MW\Box \& MW\Box;$ $MW\Box = MW\Box \& 00FFH;$ $ML\Box = ML\Box \& ML\Box;$ $ML\Box = ML\Box \& 00FF00FFH;$ $MQ\Box = MQ\Box \& MQ\Box;$ $MQ\Box = MQ\Box \& 00FF00FF 00FF00FFH;$	Performs a bit or integer AND operation.	○	○
^	XOR (Exclusive OR)	$MW\Box = MW\Box \wedge MW\Box;$ $MW\Box = MW\Box \wedge 00FFH;$ $ML\Box = ML\Box \wedge ML\Box;$ $ML\Box = ML\Box \wedge 00FF00FFH;$ $MQ\Box = MQ\Box \wedge MQ\Box;$ $MQ\Box = MWQ\Box \wedge 00FF00FF 00FF00FFH;$	Performs an integer exclusive OR operation.	○	○
!	NOT (Logical Complement)	$MB\Box = !MB\Box;$ $MB\Box = !1;$ $MW\Box = !MW\Box;$ $MW\Box = !00FFH;$ $ML\Box = !ML\Box;$ $ML\Box = !00FF00FFH;$ $MQ\Box = !MQ\Box;$ $MQ\Box = !00FF00FF 00FF00FFH;$	Returns the inverse of the specified bit.	○	○

Note: The \Box in the above formats indicates a register address.

Although operations that combine math operations are also possible, real number operations cannot be performed.

Refer to the following section for details on the priority of numeric operations.

 5.3 Operation Priority Levels (page 5-11)

Inclusive OR (|)

The | (OR) instruction performs an inclusive OR for the immediately preceding operation result and the specified registers, and then returns the result of that operation. Real number registers cannot be used.

Table 6.3 Inclusive OR Truth Table for (A = B | C)

B	C	A
0	0	0
0	1	1
1	0	1
1	1	1

Format

The format of the | (OR) instruction is as follows:



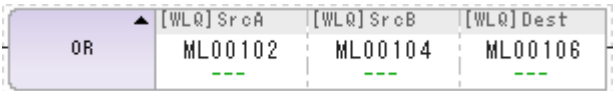
MW00100 = DW00102 | AAAAH;

① ② ③

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②, ③	Data input	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The | (OR) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the | (OR) instruction are given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	MB001000 = MB001010 MB001011;	
W	MW00100 = MW00101 MW00102	
L	ML00106 = ML00102 ML00104;	
F	—	—

AND (&)

The & (AND) instruction performs an inclusive AND for the immediately preceding operation result and the specified registers, and then returns the result of that operation. Real number registers cannot be used.

Table 6.4 AND Truth Table for (A = B & C)

B	C	A
0	0	0
0	1	0
1	0	0
1	1	1

Format

The format of the & (AND) instruction is as follows:


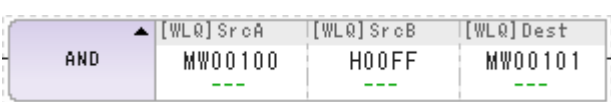
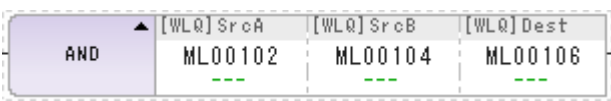
$$\text{MW00100} = \text{DW00102} \& \text{AAAAH};$$

①
②
③

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②, ③	Data input	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The & (AND) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the & (AND) instruction are given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	MB001000 = MB001010&MB001011;	
W	MW00101 = MW00100& 00FFH;	
L	ML00106 = ML00102& ML00104;	
F	—	—

Exclusive OR (^)

The ^ (XOR) instruction performs an exclusive OR for the immediately preceding operation result and the specified registers, and then returns the result of that operation. Real number registers cannot be used.

Table 6.5 XOR Truth Table for (A = B ^ C)

B	C	A
0	0	0
0	1	1
1	0	1
1	1	0

Format

The format of the ^ (XOR) instruction is as follows:

$$\text{MW00100} = \text{DW00102} \wedge \text{AAAAH};$$

① ② ③

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②, ③	Data input	<ul style="list-style-type: none"> All integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The ^ (XOR) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the ^ (XOR) instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00101 = MW00100 ^ 00FFH;	
L	ML00106 = ML00102 ^ ML00104;	
F	–	–

NOT (!)

The ! (NOT) instruction inverts the data in the specified register and returns the result of that operation. Real number registers cannot be used.

Format

The format of the ! (NOT) instruction is as follows:




MB001000 = ! MB001010;
 ① ②

Description	Application	Usable Registers
①	Data output	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All bit, integer, double-length integer, or quadruple-length integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant*

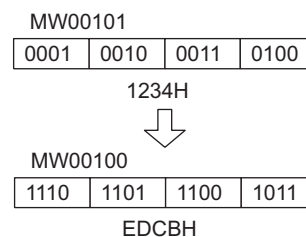
* Bit data constants cannot be specified.

Programming Examples

The ! (NOT) instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the ! (NOT) instruction are given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	MB001000 = !MB001010;	
W	MW00100 = !MW00101;	
L	ML00100 = !ML00102	
F	—	—

Example Programming Example for the ! (NOT) Instruction
 MW00100 = !MW00101;



6.7 Numeric Comparison Instructions

This section explains the numeric comparison instructions that are used in conditional expressions. There are six numeric comparison instructions. You can use these instructions in motion programs or in sequence programs.

The following table lists the numeric comparison instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
= =	Equal	IF MB□ = = MB□; WHILE MB□ = = MB□; IF MW□ = = MW□; WHILE MW□ = = MW□; IF ML□ = = ML□; WHILE ML□ = = ML□; IF MF□ = = MF□; WHILE MF□ = = MF□; IF MQ□ = = MQ□; WHILE MQ□ = = MQ□; IF MD□ = = MD□; WHILE MD□ = = MD□;	Used in an IF or WHILE conditional expression. If the left side and right side are the same, the condition is TRUE.	○	○
<>	Mismatch	IF MW□ <> MW□; WHILE MW□ <> MW□; IF ML□ <> ML□; WHILE ML□ <> ML□; IF MF□ <> MF□; WHILE MF□ <> MF□; IF MQ□ <> MQ□; WHILE MQ□ <> MQ□; IF MD□ <> MD□; WHILE MD□ <> MD□;	Used in an IF or WHILE conditional expression. If the left side and the right side do not match, the condition is TRUE.	○	○
>	Greater Than	IF MW□ > MW□; WHILE MW□ > MW□; IF ML□ > ML□; WHILE ML□ > ML□; IF MF□ > MF□; WHILE MF□ > MF□; IF MQ□ > MQ□; WHILE MQ□ > MQ□; IF MD□ > MD□; WHILE MD□ > MD□;	Used in an IF or WHILE conditional expression. If the left side is greater than the right side, the condition is TRUE.	○	○

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
<	Less Than	IF MW□ < MW□; WHILE MW□ < MW□; IF ML□ < ML□; WHILE ML□ < ML□; IF MF□ < MF□; WHILE MF□ < MF□; IF MQ□ < MQ□; WHILE MQ□ < MQ□; IF MD□ < MD□; WHILE MD□ < MD□;	Used in an IF or WHILE conditional expression. If the left side is less than the right side, the condition is TRUE.	○	○
≥	Greater Than or Equal To	IF MW□ ≥ MW□; WHILE MW□ ≥ MW□; IF ML□ ≥ ML□; WHILE ML□ ≥ ML□; IF MF□ ≥ MF□; WHILE MF□ ≥ MF□; IF MQ□ ≥ MQ□; WHILE MQ□ ≥ MQ□; IF MD□ ≥ MD□; WHILE MD□ ≥ MD□;	Used in an IF or WHILE conditional expression. If the left side is greater than or equal to the right side, the condition is TRUE.	○	○
≤	Less Than or Equal To	IF MW□ ≤ MW□; WHILE MW□ ≤ MW□; IF ML□ ≤ ML□; WHILE ML□ ≤ ML□; IF MF□ ≤ MF□; WHILE MF□ ≤ MF□; IF MQ□ ≤ MQ□; WHILE MQ□ ≤ MQ□; IF MD□ ≤ MD□; WHILE MD□ ≤ MD□;	Used in an IF or WHILE conditional expression. If the left side is less than or equal to the right side, the condition is TRUE.	○	○

Note: The □ in the above formats indicates a register address.

Numeric Comparison Instructions (==, <>, >, <, >=, <=)

These instructions are used to determine the value of conditional expressions for instructions such as branching instructions, repetition instructions, instructions for repetition with one scan wait, or I/O wait instructions.

The following table lists the six numeric comparison instructions.

Comparison Instruction	Description
==	Equal
<>	Mismatch
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To

Format

The formats of the numeric comparison instructions are as follows:

IF MB001000 == 1;
①





Description	Application	Usable Registers
①	Conditional expression	<ul style="list-style-type: none"> All bit*, integer, double-length integer, quadruple-length integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers

* Only the == (Match) instruction can be used in bit data conditional expressions.

Programming Examples

Numeric comparison instructions can be used in motion programs, sequence programs, and ladder programs.

Programming examples that use the numeric comparison instructions are given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	IF MB001000 == 1;	
W	IF MW00100 <> 10;	
L	IF ML00100 > 10000;	
F	IF MF00100 >= 3.0;	

The conditional expressions that can be used with numeric comparison instructions are described below.

◆ Bit Data Comparison

■ Format

The == (Match) instruction is used for numeric comparison.

Specify a register on the left, and either 0 or 1 on the right.

```
IF MB000000 == 0;      "MB000000 = 0
IF MB000000 == 1;      "MB000000 = 1
```

■ Operations for Conditional Expressions

&, |, and ! (AND, OR, and NOT) can be used for logical expressions.

```
IF (MB000000 & MB000001) == 1;      "MB000000 = 1 and MB000001 = 1
IF (MB000000 & !MB000001) == 1;     "MB000000 = 1 and MB000001 = 0
IF (MB000000 | MB000001) == 1;     "MB000000 = 1 or MB000001 = 1
IF (MB000000 | !MB000001) == 1;    "MB000000 = 1 or MB000001 = 0
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- If the <> (Mismatch) instruction is used for numeric comparison:

```
IF MB000000 <> 0;          ⇒ Syntax error
```

- When a numerical value is specified on the left, and a register on the right:

```
IF 1 == MB000000;         ⇒ Syntax error
IF MB000000 == MB000001; ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
IF MB000000;             ⇒ Syntax error
IF (0);                  ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
IF (MB000000 == 0) & (MB000001 == 1); ⇒ Syntax error
```

◆ Integer, Double-length Integer, or Real Number Data Comparison

■ Format

All numeric comparison instructions (==, <>, >, <, >=, <=) can be used for these data types.

Specify a register on either the left or the right side.

```
IF MW000000 == 3;"MW000000 = 3
IF ML000000 <> ML000002;"ML000000 ≠ ML000002
IF 1.23456 >= MF000000;"1.23456 ≥ MF000000
```

■ Operations for Conditional Expressions

Numeric and logic operations can be used in the expression.

```
IF MW00000 = (MW00001/3);"MW00000 = (MW00001 ÷ 3)
IF (ML00000 & F0000000H) <> ML00002;"(ML00000 ∧ F0000000H) ≠ ML00002
IF 1.23456 >= (MF00000 * MF00002);"1.23456 ≥ (MF00000 × MF00002)
```

■ Examples of Syntax Errors

A syntax error occurs in the following cases:

- When a constant is specified both on the left and right:

```
IF 0 = 3; ⇒ Syntax error
IF (3.14 * 2 * 1000) > 9000.0; ⇒ Syntax error
```

- When there is no numeric comparison instruction:

```
IF MW000000; ⇒ Syntax error
IF (-1); ⇒ Syntax error
```

- When more than one numeric comparison instruction is used:

```
IF (MW00000 < 0) & (MW000001 > 0); ⇒ Syntax error
```

6.8

Data Manipulations

Data manipulation instructions copy, move, and perform other operations on the data in the specified registers.

There are six data manipulation instructions. You can use these instructions in motion programs or in sequence programs.

The following table lists the data manipulation instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
SFR	Right Shift	SFR MB□ N□ W□;	Shifts the bit variable by the specified number of bits to the right.	○	○
SFL	Left Shift	SFL MB□ N□ W□;	Shifts the bit variable by the specified number of bits to the left.	○	○
BLK	Move Block	BLK MW□ MW□ W□;	Copies the areas of specified blocks beginning with the specified transfer source to the specified transfer destination.	○	○
CLR	Clear	CLR MW□ W□;	Clears the desired area to 0's (zeros) beginning with the specified register.	○	○
SETW	Table Initialization	SETW MW□ DW□; W□;	Stores the specified data in all registers starting from the target register to the specified number of registers thereafter.	○	○
ASCII	ASCII Conversion 1	ASCII 'Text_string' MW□;	Converts the specified characters to ASCII text, and stores the results of that operation in the specified registers.	○	○

Note: The □ in the above formats indicates a register address.

Bit Shift Right (SFR)

The SFR instruction shifts the bit string designated by the specified first bit number and bit width by the specified number of bits to the right.

Format

The format of the SFR instruction is as follows:

$$\text{SFR } \underbrace{\text{MB001000}}_{\text{①}} \underbrace{\text{N5}}_{\text{②}} \underbrace{\text{W10}}_{\text{③}} ;$$

Description	Application	Usable Registers
①	First bit	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Number of bits to shift	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript.
③	Bit width	<ul style="list-style-type: none"> Subscript registers Constant

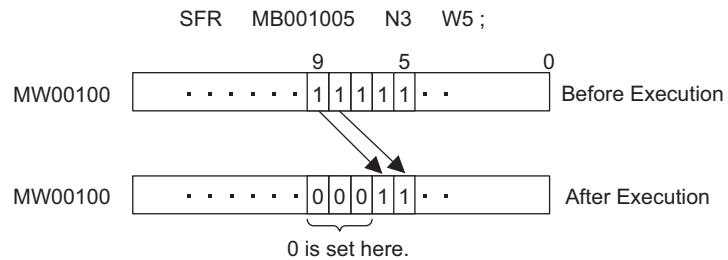
Programming Example

The SFR instruction can be used in motion programs, sequence programs, and ladder programs. A programming example that uses the SFR instruction is given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	SFR MB001000 N5 W10;	
L	–	–
F	–	–

Example Programming Example for SFR Instruction

In this example, five bits starting from MB001005 (bit 5 of MW00100) are shifted three bits to the right.



Information With the SFR instruction, if the number of bits to shift is greater than the bit width, all data in the specified bit width will be set to 0.

Move Block (BLK)

The BLK instruction moves the specified number of words from the beginning of the source register to the beginning of the destination register.

Format


The format of the BLK instruction is as follows:

BLK MW00100 DW00100 W10 ;
 ① ② ③

Description	Application	Usable Registers
①	First register at source	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	First destination register	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript.
③	Number of blocks to be moved	<ul style="list-style-type: none"> Subscript registers Constant

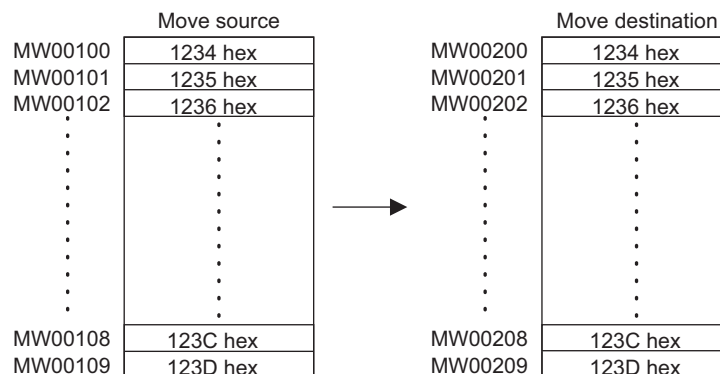
Programming Example

The BLK instruction can be used in motion programs, sequence programs, and ladder programs. A programming example that uses the BLK instruction is given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	BLK MW00100 DW00100 W10;	
L	–	–
F	–	–

Example Programming Example for the BLK Instruction
 MW00100 to MW00109 are moved to MW00200 to MW00209.

BLK MW00100 MW00200 W10;



Information As long as the source registers and destination registers do not overlap, the source data is moved to the destination registers as it is. If the source and destination data overlap, the source data may not be moved to the destination registers as it is.

Clear (CLR)

The CLR instruction clears the specified number of blocks to 0 starting from the first specified data clear register.

Format

The format of the CLR instruction is as follows:

CLR MW00100 W10 ;
 ① ②

Description	Application	Usable Registers
①	First data clear register	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Number of blocks	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Example

The CLR instruction can be used in motion programs, sequence programs, and ladder programs. A programming example that uses the CLR instruction is given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	–	–
W	CLR MW00100 W10;	
L	–	–
F	–	–

Example Programming Example for the CLR Instruction
 The data in registers MW00100 to MW00119 is cleared to 0.

CLR MW00100 W20;

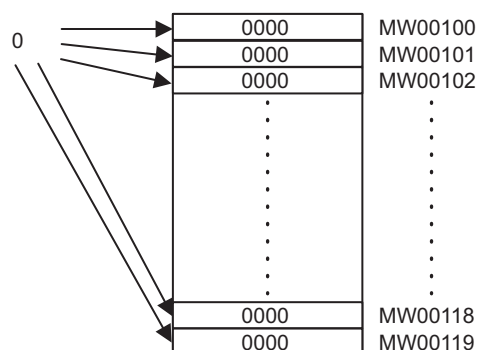
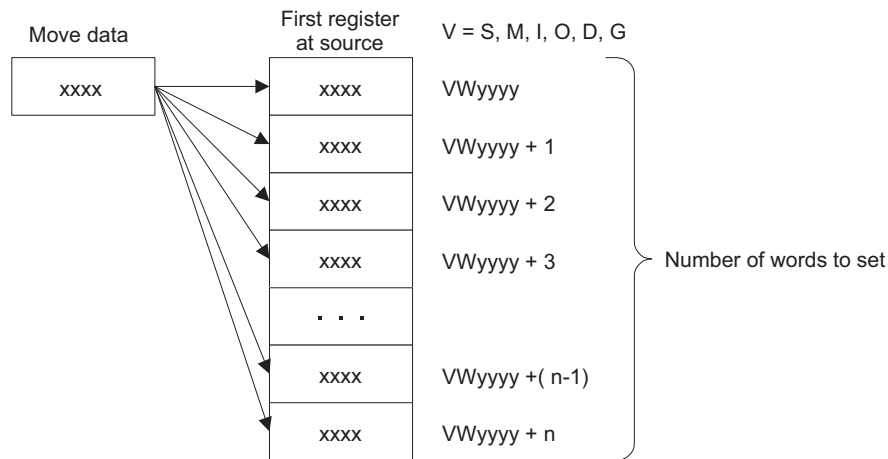


Table Initialization (SETW)

The SETW instruction stores the specified data in all registers starting from the target register to the specified number of registers thereafter. The storage process is performed one word at a time in order of ascending register addresses.



Format

The format of the SETW instruction is as follows:

SETW MW00100 DW00100 W10;

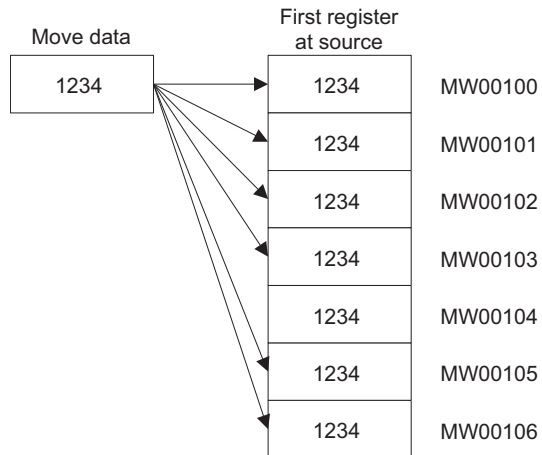
① ② ③

Description	Application	Usable Registers
①	First register at source	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Move data	<ul style="list-style-type: none"> Integer registers (excluding # and C registers) Same as above except with a subscript.
③	Number of words to set	<ul style="list-style-type: none"> Subscript registers Constant

Programming Example

A programming example that uses the SETW instruction is given below.

```
DW00100 = 1234;  
SETW MW00100 DW00100 W7; "Store the value of DW00100 in registers MW00100 to MW00106.  
END;
```



ASCII Conversion 1 (ASCII)

The ASCII instruction converts the specified characters to ASCII text, and stores the result of that operation in the specified integer register. The text string is case sensitive.

The first character is stored in the lower byte of the first word and the second character is stored in the upper byte of the first word. The remaining characters are stored in order in that same way. If the number of characters in the string is odd, the upper byte of the last word in the destination register is 0. The input text string can contain up to 32 characters.

Format

The format of the ASCII instruction is as follows:

```
ASCII 'ABCDEFGH' MW00200;
```

① ②

Description	Application	Usable Registers
①	Text string	ASCII text
②	Destination register address	Integer registers (excluding # and C registers)

The following tables show the characters that can and cannot be used in the ASCII instruction.

◆ Usable Characters

The following table lists the characters that can be used in the ASCII instruction.

Item	ASCII Characters
Alphanumeric characters	a to z, A to Z, 0 to 9
Single-byte symbols	Space character ! # \$ % & () * + , - . / : ; < = > ? @ [\] ^ _ ` { } ~

◆ Unusable Characters

The following table lists the characters that cannot be used in the ASCII instruction.

Item	ASCII Characters
Single quotation mark	'
Double quotation mark	“
Double slash	//
Double-byte characters	All double-byte characters
Single-byte Japanese characters	All single-byte Japanese characters

Programming Examples

Programming examples that use the ASCII instruction are given below.

◆ Storing the Text String “ABCD” in Registers MW00100 to MW00101

ASCII 'ABCD' MW00100;

	Upper byte	Lower byte	
MW00100	42H('B')	41H('A')	MW00100 = 4241H
MW00101	44H('D')	43H('C')	MW00101 = 4443H

◆ Storing the Text String “ABCDEFG” in Registers MW00100 to MW00103

ASCII 'ABCDEFG' MW00100;

	Upper byte	Lower byte	
MW00100	42H('B')	41H('A')	MW00100 = 4241H
MW00101	44H('D')	43H('C')	MW00101 = 4443H
MW00102	46H('F')	45H('E')	MW00102 = 4645H
MW00103	00H	47H('G')	MW00103 = 0047H

The remaining bytes will be 0.

6.9 Basic Functions

Basic function instructions perform special operations through a combination of numeric and logic operations. There are 17 basic function instructions.

The following table lists the basic function instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
SIN	Sine	SIN (MW□); SIN (90);	Calculates the sine. The specifications depend on whether the data type is integer or real number.	○	○
COS	Cosine	COS (MW□); COS (90);	Calculates the cosine. The specifications depend on whether the data type is integer or real number.	○	○
TAN	Tangent	TAN (MF□); TAN (45.0);	Calculates the tangent. Only a real number register can be specified.	○	○
ASN	Arc Sine	ASN (MF□); ASN (90.0);	Calculates the arc sine. Only a real number register can be specified.	○	○
ACS	Arc Cosine	ACS (MF□); ACS (90.0);	Calculates the arc cosine. Only a real number register can be specified.	○	○
ATN	Arc Tangent	ATN (MW□); ATN (45);	Calculates the arc tangent. The specifications depend on whether the data type is integer or real number.	○	○
SQT	Square Root	SQT (MW□); SQT (100);	Calculates the square root. The specifications depend on whether the data type is integer or real number.	○	○
BIN	BCD→BIN	BIN (MW□);	Converts BCD data to binary data.	○	○
BCD	BIN→BCD	BCD (MW□);	Converts binary data to BCD data.	○	○
S { }	Set Bit	$S \{MB\} = MB \& MB$;	If the logic operation result is TRUE, the specified bit turns ON. However, the specified bit is not turned OFF even if the result of the logic operation is FALSE.	○	○
R { }	Reset Bit	$R \{MB\} = MB \& MB$;	If the logic operation result is TRUE, the specified bit turns OFF. However, the specified bit is not turned ON even if the result of the logic operation is FALSE.	○	○
PON	Rising-edge Pulse	$MB = PON (MB \ MB)$; Or $IF \ PON (MB \ MB) = 1$; · · · ; IEND;	The bit output turns ON for one scan when the bit input status changes from OFF to ON.	×	○

Continued on next page.

Continued from previous page.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
NON	Falling-edge Pulse	$MB\Box = \text{NON} (MB\Box MB\Box);$ Or $\text{IF NON} (MB\Box MB\Box) = 1;$ $\dots;$ $\text{IEND};$	The bit output turns ON for one scan when the bit input status changes from ON to OFF.	×	○
TON	On-Delay Timer	$MB\Box = MB\Box \& \text{TON} (\Box MB\Box);$	Counts the time whenever the bit input is ON. The bit output turns ON when the counted value is equal to the set value. Counting unit: 10 ms	×	○
TON1MS	1-ms ON-Delay Timer	$DB\Box = DB\Box \& \text{TON1MS} (\Box DB\Box);$	Counts the time whenever the bit input is ON. The bit output turns ON when the counted value is equal to the set value. Counting unit: 1 ms	×	○
TOF	Off-Delay Timer	$MB\Box = MB\Box \& \text{TOF} (\Box MB\Box);$	Counts the time whenever the bit input is OFF. The bit output turns OFF when the counted value is equal to the set value. Counting unit: 10 ms	×	○
TOF1MS	1-ms OFF-Delay Timer	$DB\Box = DB\Box \& \text{TOF1MS} (\Box DW\Box);$	Counts the time whenever the bit input is OFF. The bit output turns OFF when the counted value is equal to the set value. Counting unit: 1 ms	×	○

Note: The \Box in the above formats indicates a register address.

Sine (SIN)

The SIN instruction returns the sine of the specified integer or real number data as the operation result. Double-length integers cannot be used.

Format

The format of the SIN instruction is as follows:

$$\underbrace{\text{MW00100}}_{\textcircled{1}} = \text{SIN}(\underbrace{3000}_{\textcircled{2}});$$

Description	Application	Unit	Usable Registers
①	Sine value output	—	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Angle input	Angle (°)*	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* The input unit and output results will be different for integer and real number data.

- Integers

Use integers that are between -327.68° and 327.67° . The result of the immediately preceding operation (integer data) is used as the input, and the operation result is returned in an integer register (input unit $1 = 0.01^\circ$). The operation result is multiplied by 10,000 before being output.

- Real Numbers

The result of the immediately preceding operation (real number data) is used as the input, and the sine is returned in a real number register (unit = degrees).

Integer Data	Equivalent	Real Number Data
$\text{MW00102} = \text{SIN}(\text{MW00100});$ (05000) (03000)	$\Rightarrow 0.5 = \text{SIN}30^\circ$	$\text{MF00102} = \text{SIN}(\text{MF00100});$ (0.5) (30.0)



Note

If an integer is input that is not between -327.68° and 327.67° , a correct result will not be obtained.

Programming Examples

The SIN instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the SIN instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	—	—
W	$\text{MW00102} = \text{SIN}(\text{MW00100});$	
L	—	—
F	$\text{DF00202} = \text{SIN}(\text{DF00200});$	

Cosine (COS)

The COS instruction returns the cosine of the specified integer or real number data as the operation result. Double-length integers cannot be used.

Format

The format of the COS instruction is as follows:

$$\underset{\textcircled{1}}{\text{MW00100}} = \text{COS}(\underset{\textcircled{2}}{3000});$$

Description	Application	Unit	Usable Registers
①	Cosine value output	—	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Angle input	Angle (°)*	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* The input unit and output results will be different for integer and real number data.

- Integers

Use integers that are between -327.68° and 327.67° . The result of the immediately preceding operation (integer data) is used as the input, and the operation result is returned in an integer register (input unit $1 = 0.01^\circ$). The operation result is multiplied by 10,000 before being output.

- Real Numbers

The result of the immediately preceding operation (real number data) is used as the input, and the cosine is returned in a real number register (unit = degrees).

Integer Data		Equivalent	Real Number Data	
MW00102 = COS (MW00100);	(05000) (06000)	$\Rightarrow 0.5 = \text{COS}60^\circ$	MF00102 = COS (MF00100);	(0.5) (60.0)



Note

If an integer is input that is not between -327.68° and 327.67° , a correct result will not be obtained.

Programming Examples

The COS instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the COS instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	—	—
W	MW00102 = COS(MW00100);	
L	—	—
F	DF00202 = COS(DF00200);	

Tangent (TAN)

The TAN instruction uses the specified variable or constant (unit = degrees) as the input and returns the tangent in a real number register.

Format

The format of the TAN instruction is as follows:

$$\text{MW00100} = \text{TAN} (1.0);$$

① ②

Description	Application	Unit	Usable Registers
①	Tangent value output	–	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Angle input	Angle (°)*	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* Example: To find the tangent of the input value ($\theta = 45.0^\circ$), the following calculation is performed: $\text{TAN}(\theta) = 1.0$.

$$\text{DF00102} = \text{TAN}(\text{DF00100});$$

(1.0) (45.0)



The TAN instruction can be used only with real number data. A compiling error will occur when the program is compiled if bit, integer, or double-length integer data is specified.

Programming Example

The TAN instruction can be used in motion programs, sequence programs, and ladder programs.

A programming example that uses the TAN instruction is given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	–	–
L	–	–
F	DF00202 = TAN(DF00200);	

Arc Sine (ASN)

The ASN instruction uses the specified variable or constant as the input and returns the arc sine (unit = degrees) in a real number register.

Format

The format of the ASN instruction is as follows:

$$\text{MF00100} = \text{ASN}(\text{0.5});$$

① ②

Description	Application	Unit	Usable Registers
①	Angle output	Angle (°)*	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Sine value input	–	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* Example: To find the arc sine of the input value (0.5), the following calculation is performed: $\text{ASN}(0.5) = 30.0^\circ$.

$$\text{MF00202} = \text{ASN}(\text{MF00200});$$

(30.0) (0.5)



Important

The ASN instruction can be used only with real number data. A compiling error will occur when the program is compiled if bit, integer, or double-length integer data is specified.

Programming Example

The ASN instruction can be used in motion programs, sequence programs, and ladder programs. A programming example that uses the ASN instruction is given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	–	–
L	–	–
F	DF00202 = ASN(DF00200);	

Arc Cosine (ACS)

The ACS instruction uses the specified variable or constant as the input and returns the arc cosine (unit = degrees) in a real number register.

Format

The format of the ACS instruction is as follows:

$$\text{MF00100} = \text{ACS}(\text{0.5});$$

① ②

Description	Application	Unit	Usable Registers
①	Angle output	Angle (°)*	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Cosine value input	—	<ul style="list-style-type: none"> All real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* Example: To find the arc cosine of the input value (0.5), the following calculation is performed: $\text{ACS}(0.5) = 60.0^\circ$.

$$\text{MF00100} = \text{ACS}(\text{MF00102});$$

(60.0) (0.5)



The ACS instruction can be used only with real number data. A compiling error will occur when the program is compiled if bit, integer, or double-length integer data is specified.

Programming Example

The ACS instruction can be used in motion programs, sequence programs, and ladder programs.

A programming example that uses the ACS instruction is given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	—	—
W	—	—
L	—	—
F	DF00202 = ACS(DF00200);	

Arc Tangent (ATN)

The ATN instruction returns the arc tangent of the specified integer or real number data as the operation result.

Double-length integers cannot be used.

Format

The format of the ATN instruction is as follows:

$$\underbrace{\text{MW00100}}_{\textcircled{1}} = \text{ATN} \left(\underbrace{100}_{\textcircled{2}} \right);$$

Description	Application	Unit	Usable Registers
①	Angle output	Angle (°)*	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Tangent value input	–	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

* The input unit and output results will be different for integer and real number data.

- Integers

Use integers that are between -327.68 and 327.67.

The result of the immediately preceding operation (integer data) is used as the input, and the operation result is returned in an integer register (input 1 = 0.01). The operation result is multiplied by 100 before it is output.

- Real Numbers

The result of the immediately preceding operation (real number data) is used as the input, and the arc tangent is returned in a real number register.

Integer Data		Equivalent	Real Number Data	
MW00100 = ATN (MW00102);	(04500)	⇒ 45=ATN(1.0)	MF00100 = ATN (MF00102);	(45.0)
	(00100)			(1.0)

Programming Examples

The ATN instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the ATN instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00102 = ATN(MW00100);	
L	–	–
F	DF00202 = ATN(DF00200);	

Square Root (SQT)

The SQT instruction returns the square root of the specified integer or real number data as the operation result.

Double-length integers cannot be used.

Format

The format of the SQT instruction is as follows:

$$\underset{\textcircled{1}}{\text{MW00100}} = \text{SQT} (\underset{\textcircled{2}}{\text{100}});$$

Description	Application	Usable Registers
①	Square root output	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	Data input	<ul style="list-style-type: none"> All integer, real number, or double-length real number registers (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Note: The input unit and output results will be different for integer and real number data.

- Integer Data

The result is different from that obtained for the mathematical square root, and is calculated with the following formula:

$$\text{sign}(\text{Input data}) \times \sqrt{|\text{Input data}| \times 32,768}$$

sign(Input data): The sign of the input data.

|input data|: The absolute value of the input data.

This is the same as multiplying the result of the mathematical square root by $\sqrt{32768}$. If the input is a negative number, the square root of the absolute value is calculated, and the negative value is given as the operation result. The maximum operation error is ± 2 .



- Real Number Data

The SQT instruction uses the immediately preceding operation result (real number data) as the input and returns the square root as real number data.

	Integer Data	Real Number Data
Positive input value	MW00100 = SQT (MW00102); (01448) (00064) (8) (181)	MF00100 = SQT (MF00102); (8.0) (64.0)
Negative input value	MW00100 = SQT (MW00102); (-01448) (-00064) (8) (181)	MF00100 = SQT (MF00102); (-8.0) (-64.0)

Programming Examples

The SQT instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the SQT instruction are given below.

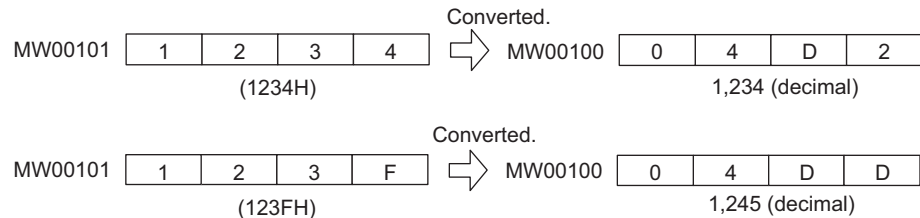
Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	–	–
W	MW00102 = SQT(MW00100);	
L	–	–
F	DF00202 = SQT(DF00200);	

BCD to Binary (BIN)

The BIN instruction converts BCD data to binary data.

Only integer data can be used. If non-BCD data is specified, a correct result cannot be obtained.

Example BCD to Binary Conversion Example



Note

If non-BCD data is specified, a correct result cannot be obtained.

Format

The format of the BIN instruction is as follows:

MW00100 = BIN (1234H);

① ②

Description	Application	Usable Registers
①	Binary output	<ul style="list-style-type: none"> All registers with integer and double-length integer data types (excluding # and C registers) Same as above except with a subscript. Subscript registers
②	BCD input	<ul style="list-style-type: none"> All registers with integer and double-length integer data types (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The BIN instruction can be used in motion programs, sequence programs, and ladder programs.

Programming examples that use the BIN instruction are given below.

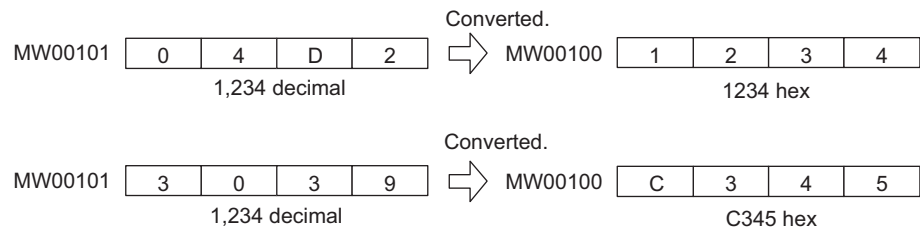
Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	-	-
W	MW00101 = BIN(MW00100);	
L	ML00102 = BIN(ML00100);	
F	-	-

Binary to BCD (BCD)

The BCD instruction converts binary data to BCD data.

Only integer data can be used. If the binary data exceeds 270F or is a negative value, a correct result cannot be obtained.

Example Binary to BCD Conversion Example



Note

If the binary data exceeds 270F, a correct result cannot be obtained.

Format

The format of the BCD instruction is as follows:

$$\frac{\text{MW00100}}{\text{A}} = \text{BCD} \left(\frac{\text{1234}}{\text{B}} \right);$$

Description	Application	Usable Registers
A	BCD output	<ul style="list-style-type: none"> All registers with integer and double-length integer data types (excluding # and C registers) Same as above except with a subscript. Subscript registers
B	Binary input	<ul style="list-style-type: none"> All registers with integer and double-length integer data types (excluding # and C registers) Same as above except with a subscript. Subscript registers Constant

Programming Examples

The BCD instruction can be used in motion programs, sequence programs, and ladder programs. Programming examples that use the BCD instruction are given below.

Data Type	Motion Programs/Sequence Programs	Ladder Programming
B	—	—
W	MW00101 = BCD(MW00100);	
L	ML00102 = BCD(ML00100);	
F	—	—

Set Bit (S{ })

The S{ } instruction turns ON the specified bit if the result of the specified logic operation is TRUE. However, the specified bit is not turned OFF even if the result of the logic operation is FALSE.

Format

The format of the S{ } instruction is as follows:

$$S \{ \underset{\textcircled{1}}{\text{MB001000}} \} = \underset{\textcircled{2}}{\text{MB001010} \& \text{MB001011}};$$

Description	Application	Usable Registers
①	Specified bit	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Logic operation expression	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript. Constant

Programming Example

The S{ } instruction can be used in motion programs, sequence programs, and ladder programs. A programming example that uses the S{ } instruction is given below.

Data Type	Motion Programs/ Sequence Programs	Ladder Programming
B	S{MB001000} = MB001010&MB001011;	
W	—	—
L	—	—
F	—	—

Rising-edge Pulse (PON)

The PON instruction turns ON the bit output for one scan when the bit input changes from 0 to 1. The register that stores the previous bit output value is used as the work register for PON instruction processing. Set a register that is not used in any other processes.

Format

The format of the PON instruction is as follows:

$$\underbrace{\text{DB000002}}_{\textcircled{1}} = \text{PON} (\underbrace{\text{DB000000}}_{\textcircled{2}} \underbrace{\text{DB000001}}_{\textcircled{3}}) ;$$

Description	Application	Usable Registers
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers Same as above except with a subscript.
③	Storage for the previous bit output value	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.

Programming Example

A programming example that uses the PON instruction is given below.

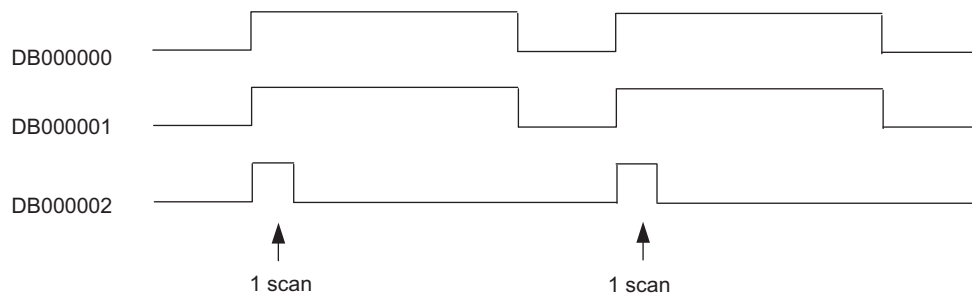
◆ Outputting to a Coil

```
DB000002 = PON(DB000000 DB000001);
```

• Equivalent Ladder Programming Example



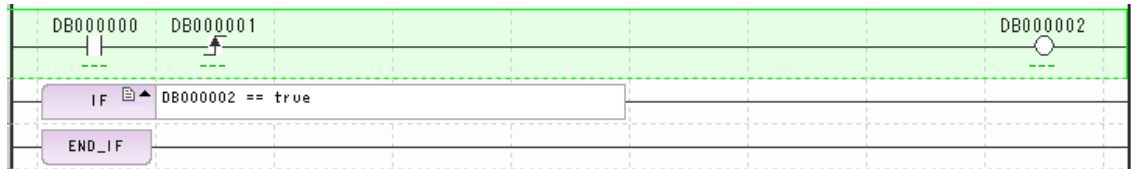
• Timing Charts



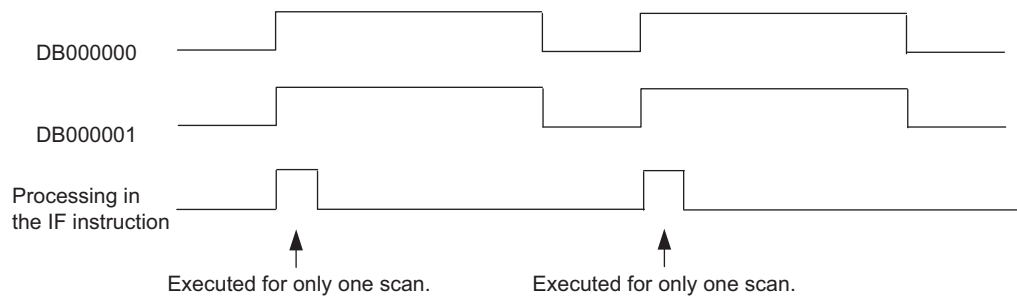
◆ When Used with the IF Instruction

```
IF PON(DB000000 DB000001) == 1;
  •
  •
IEND;
```

• Equivalent Ladder Programming Example



• Timing Charts



Falling-edge Pulse (NON)

The NON instruction turns ON the bit output for one scan when the bit input changes from 1 to 0. The register that stores the previous bit output value is used as the work register for NON instruction processing. Set a register that is not used in any other processes.

Format

The format of the NON instruction is as follows:

$$\underbrace{\text{DB000002}}_{\textcircled{1}} = \text{NON} (\underbrace{\text{DB000000}}_{\textcircled{2}} \underbrace{\text{DB000001}}_{\textcircled{3}}) ;$$

Description	Application	Usable Registers
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers Same as above except with a subscript.
③	Storage for the previous bit output value	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.

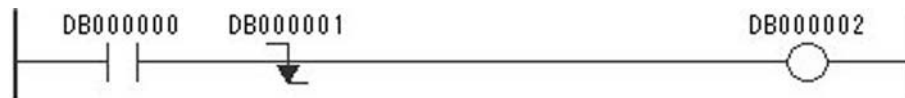
Programming Example

A programming example that uses the NON instruction is given below.

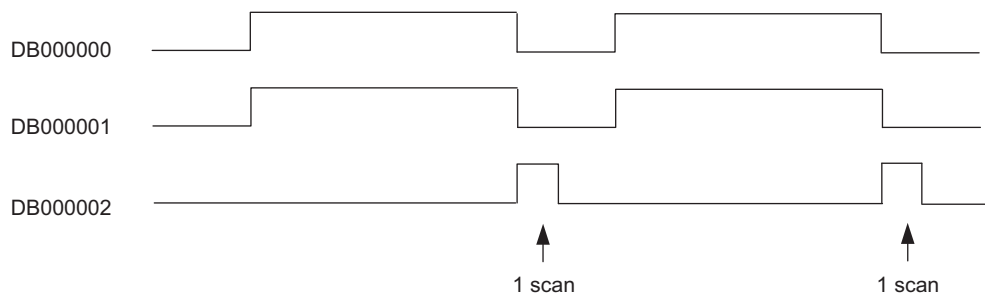
◆ Outputting to a Coil

```
DB000002 = NON(DB000000 DB000001);
```

• Equivalent Ladder Programming Example



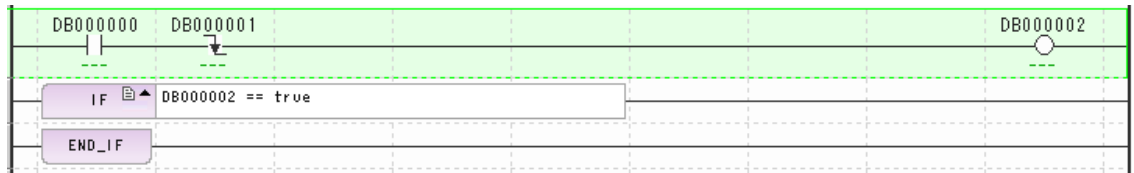
• Timing Charts



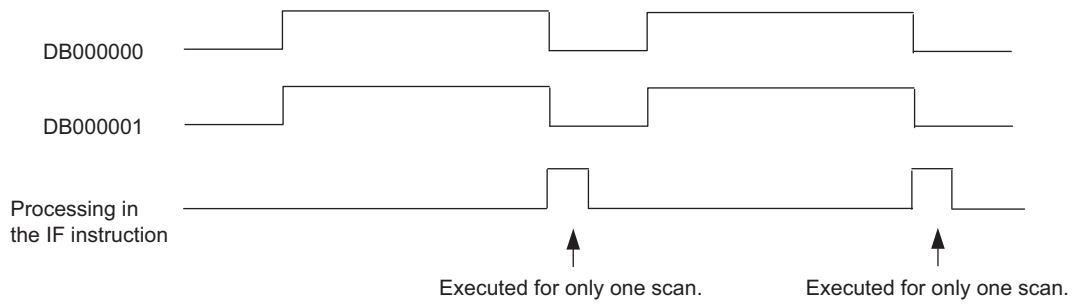
◆ When Used with the IF Instruction

```
IF NON(DB000000 DB000001) == 1;
  •
  •
IEND;
```

• Equivalent Ladder Programming Example



• Timing Charts



On-delay Timer: Measurement unit = 10 ms (TON)

The TON instruction counts the duration that the bit input is ON with a measurement unit of 10 ms.

The bit output turns ON when the counted value is equal to the set value.

If the bit input turns OFF during counting, the timer stops. If the bit input turns ON again, the timer starts counting again from 0. The actual counted time (in units of 10 ms) is stored in the Count register.

Format

The format of the TON instruction is as follows:

DB000001 = DB000000 & TON (500 DW00001);

① ② ③ ④

Description	Application	Usable Registers
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers Same as above except with a subscript.
③	Set Value	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript. Constants (0 to 65,535 (655.35 s) in 10 ms intervals)
④	Register for timer counting	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript.



Important

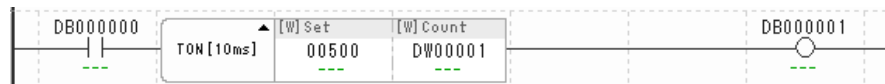
- Time is not counted while the debugging operation is stopped. Counting starts again from the current value after the debugging operation restarts.
- Never omit the “DB□□□□□ &” bit input.

Programming Example

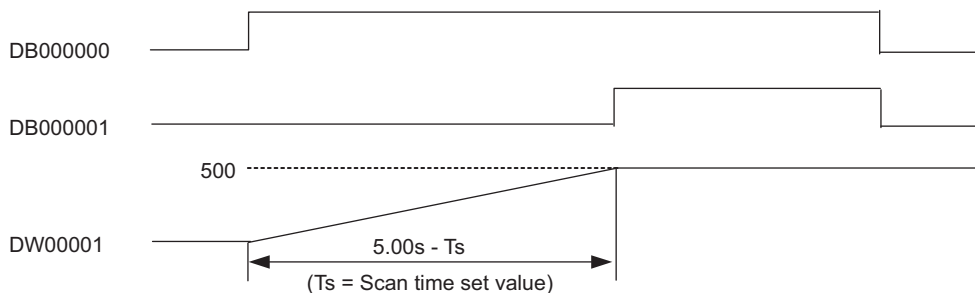
A programming example that uses the TON instruction is given below.

DB000001 = DB000000 & TON (500 DW00001);
 ↑ Set to 5 seconds.

- Equivalent Ladder Programming Example



- Timing Charts



1-ms ON-Delay Timer (TON1MS)

The TON1MS instruction counts the duration that the bit input is ON with a measurement unit of 1 ms, and turns ON the bit output when the counted value is equal to the set value.

If the bit input turns OFF during counting, the timer stops. If the bit input turns ON again, the timer starts counting again from 0. The actual counted time (in units of 1 ms) is stored in the Count register.

Format

The format of the TON1MS instruction is as follows:

```
DB000001 = DB000000 & TON1MS( 500 DW000001);
```

①
②
③
④

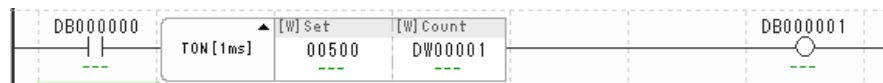
Description	Application	Applicable Data
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
③	Set Value	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript. Constants (0 to 65,535 (65.535 s) in 1 ms intervals)
④	Register for timer counting	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript.

Programming Example

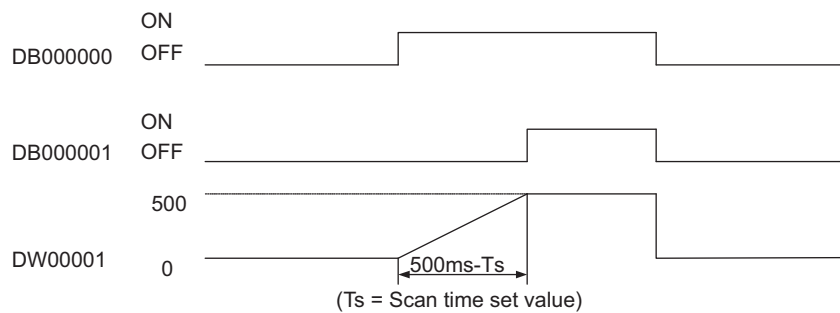
A sequence programming example and a ladder programming example that use the TON1MS instruction are given below.

```
DB000001 = DB000000 & TON1MS(500 DW000001);
```

• Equivalent Ladder Programming Example



• Timing Charts



Off-delay Timer: Measurement unit = 10 ms (TOF)

The TOF instruction counts the duration that the bit input is OFF with a measurement unit of 10 ms.

The bit output turns OFF when the counted value is equal to the set value.

If the bit input turns ON during counting, the timer stops. If the bit input turns OFF again, the timer starts counting again from 0. The actual counted time (in units of 10 ms) is stored in the Count register.

Format

The format of the TOF instruction is as follows:

$$\text{DB000001} = \text{DB000000} \ \& \ \text{TOF} \ (\text{500} \ \text{DW00001});$$

①
②
③
④

Description	Application	Usable Registers
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers Same as above except with a subscript.
③	Set Value	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript. Constants (0 to 65,535 (655.35 s) in 10 ms intervals)
④	Register for timer counting	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript.



Important

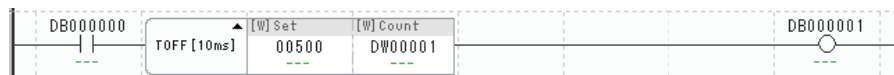
1. Time is not counted while the debugging operation is stopped. Counting starts again from the current value after the debugging operation restarts.
2. Never omit the “DB□□□□□ &” bit input.

Programming Example

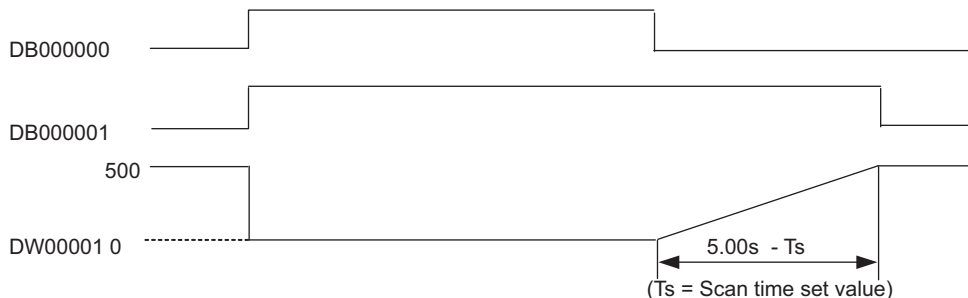
A programming example that uses the TOF instruction is given below.

$$\text{DB000001} = \text{DB000000} \ \& \ \text{TOF} \ (\text{500} \ \text{DW00001});$$

- Equivalent Ladder Programming Example



- Timing Charts



1-ms OFF-Delay Timer (TOF1MS)

The TOF1MS instruction counts the duration that the bit input is OFF with a measurement unit of 1 ms, and turns OFF the bit output when the counted value is equal to the set value.

If the bit input turns ON during counting, the timer stops. If the bit input turns OFF again, the timer starts counting again from 0. The actual counted time (in units of 1 ms) is stored in the Count register.

Format

The format of the TOF1MS instruction is as follows:

```
DB000001 = DB000000 & TOF1MS( 500 DW000001 );
```

①
②
③
④

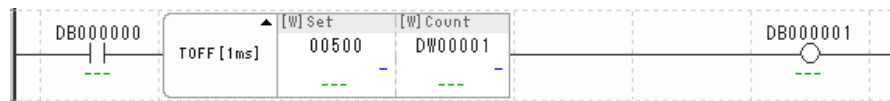
Description	Application	Applicable Data
①	Bit output	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
②	Bit input	<ul style="list-style-type: none"> All bit data registers (excluding # and C registers) Same as above except with a subscript.
③	Set Value	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript. Constants (0 to 65,535 (65.535 s) in 1 ms intervals)
④	Register for timer counting	<ul style="list-style-type: none"> All integer data registers Same as above except with a subscript.

Programming Example

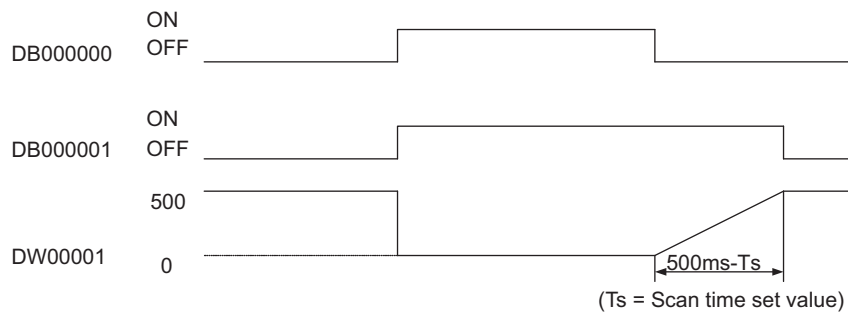
A sequence programming example and a ladder programming example that use the TOF1MS instruction are given below.

```
DB000001 = DB000000 & TOF1MS(500 DW000001);
```

• Equivalent Ladder Programming Example



• Timing Charts



6.10 Vision Instructions

The vision instructions are used to get or analyze images taken with the camera of a YVD-001 Vision Unit.

There are five vision instructions. You can use these instructions only in motion programs.

The following table lists the vision instructions.

Instruction	Name	Format	Description	Motion Programs	Sequence Programs
VCAPI	Capture Image	<i>VCAPI[Logical_circuit_name.Logical_camera_name]Image_memory_number[Logical_circuit_name.Logical_camera_name]Image_memory_number...;</i>	Gets an image from the camera.	○	×
VCAPS	Capture Image with External Trigger Sync	<i>VCAPS[Logical_circuit_name.Logical_camera_name]Image_memory_number[Logical_circuit_name.Logical_camera_name]Image_memory_number...TW(FW)Release_signal;</i>	Gets an image from the camera on an external trigger signal.	○	×
VFIL	Filter Image	<i>VFIL[Logical_circuit_name]Request_parameter[Logical_circuit_name]Request_parameter...;</i>	Applies a filter before image analysis.	○	×
VANA	Analyze Image	<i>VANA[Logical_circuit_name]Request_parameter Response_parameter[Logical_circuit_name]Request_parameter Response_parameter...;</i>	Analyzes an image.	○	×
VRES	Get Analysis Results	<i>VRES[Logical_circuit_name]Request_parameter Response_parameter[Logical_circuit_name]Request_parameter Response_parameter...;</i>	Gets the results of image analysis.	○	×

Features of the MPE720 Engineering Tool



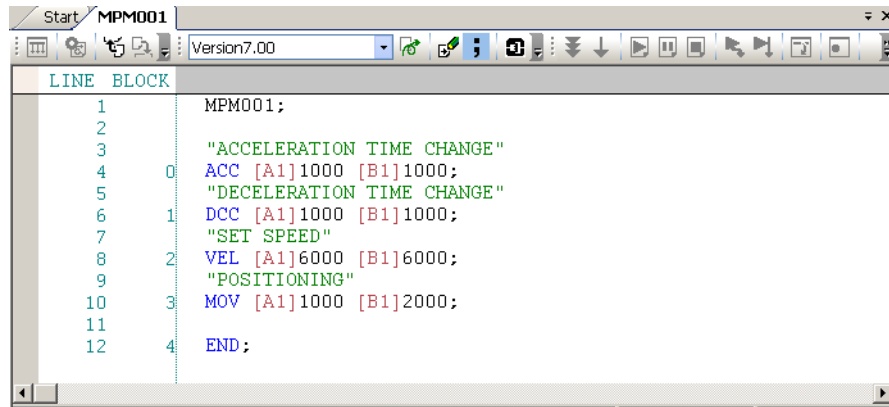
This chapter describes the features of the MPE720 Engineering Tool for motion programs and sequence programs.

7.1	Motion Editor	7-2
7.2	Motion Instruction Entry Assistance	7-5
7.3	Task Assignments	7-9
7.4	Debug Operation	7-11
7.5	Drive Control Panel	7-18
7.6	Test Runs	7-20
7.7	Axis Monitor and Alarm Monitor	7-23
7.8	Cross References	7-27

7.1 Motion Editor

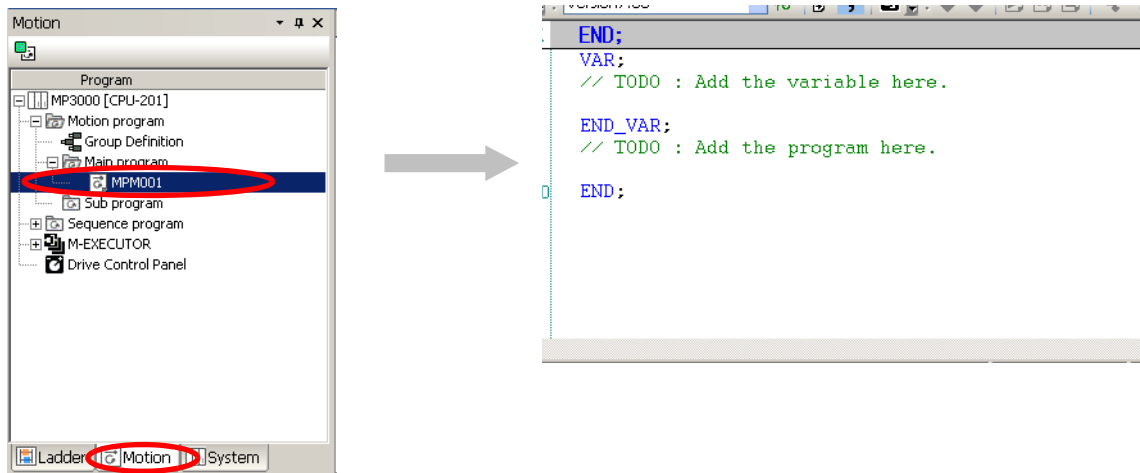
The Motion Editor is a programming tool that is required to create and edit motion programs and sequence programs. It has a full range of functions to create and edit these programs, including text editing, compiling (saving), debugging, and monitoring.

The Motion Editor Tab Page is shown below.

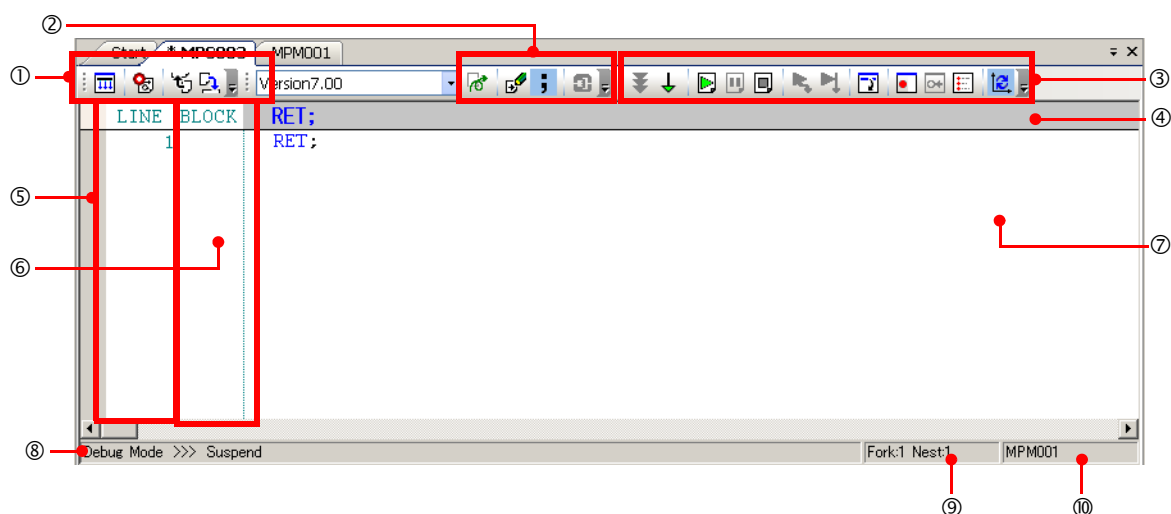


LINE	BLOCK	Code
1		MPM001;
2		
3		"ACCELERATION TIME CHANGE"
4	0	ACC [A1]1000 [B1]1000;
5		"DECELERATION TIME CHANGE"
6	1	DCC [A1]1000 [B1]1000;
7		"SET SPEED"
8	2	VEL [A1]6000 [B1]6000;
9		"POSITIONING"
10	3	MOV [A1]1000 [B1]2000;
11		
12	4	END;

To start the Motion Editor, select a program to open in the Motion Pane.



■ Motion Editor Tab Page



No.	Name	Description	
①	Monitor Toolbar		This icon causes the display to follow the Block Monitor so that it is always visible on the screen.
			This icon displays the Motion Alarm Dialog Box.
			This icon calls a motion subprogram.
			This icon displays the Motion Tasks Tab Page.
②	Programming Toolbar		This icon compiles the currently open program.
			This icon displays the Motion Command Assist Dialog Box.
			This icon automatically adds a semicolon.
			This icon displays the Task Allocation Dialog Box.
③	Motion Debugging Toolbar		This icon performs operation in Debug Operation Mode.
			This icon performs operation in Normal Operation Mode.
			This icon executes the program.
			This icon stops execution of the program.
			This icon forces the program in execution to end.
			This icon performs a step-in operation.
			This icon performs a step-over operation.
			This icon moves the start point for execution.
			This icon sets or removes a breakpoint.
			This icon enables or disables a breakpoint.
			This icon displays a list of all breakpoints.
			This icon updates to the most recent state.

Continued on next page.

Continued from previous page.

No.	Name	Description
④	Input Guidance	The guidance allows you to check the syntax of motion language instructions as you create the motion program. Place the mouse cursor over any motion language instruction (blue text) to display details on how to enter that instruction.
⑤	Line	This is the number of lines of text (instructions, comments, blank lines, etc.) in the currently open program.
⑥	Block	This is the number of lines of actual code executed in the program. This does not include lines such as comments or empty space.
⑦	Editor Area	This is the area where you enter the instruction in the program.
⑧	Status Bar	The status bar displays information such as the current operating mode or alarms that have occurred. Normal Operation Mode Executing: Execution is in progress. Alarm: An alarm occurred. Debug Operation Mode Executing in Debug Operation Mode: Debug Operation Mode Single-block execution stopped: Single-block execution is stopped in Debug Operation Mode. Alarm: An alarm occurred in Debug Operation Mode
⑨	Parallel/Nest	The fork number and nesting level are displayed here.
⑩	Main Program	The number of the main program that is calling the currently open program is displayed.
⑪	Compiler Version	The compiling options are specified. Version 7.00 All MP3000 motion program functions are supported. This setting is used for all programs created in MPE720 version 7. Version 6 Compatible Only MP2000 motion program functions are supported. This setting is used for programs that were created on MPE720 version 6 or version 5.

7.2

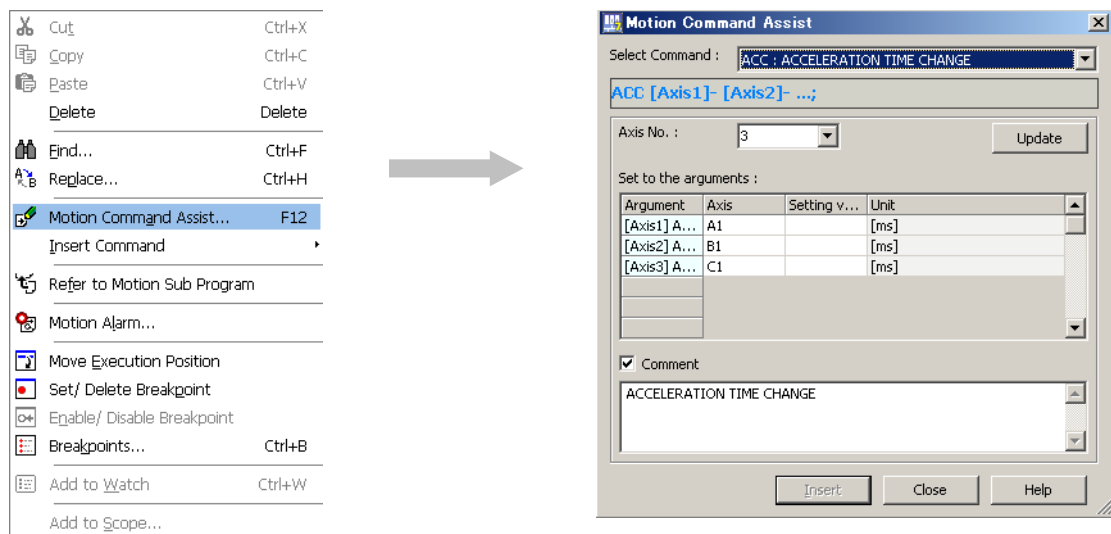
Motion Instruction Entry Assistance

Instruction entry assistance helps you enter motion language instructions when you create motion programs.

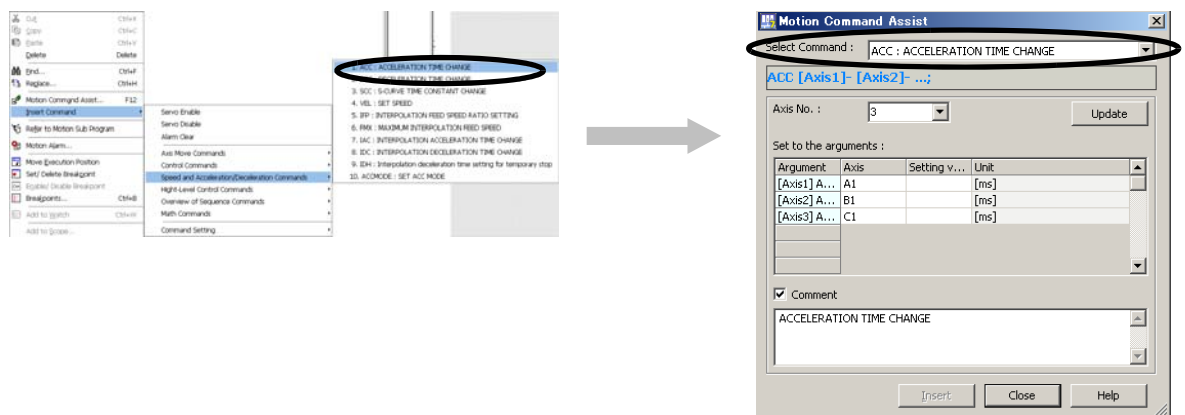
Motion language instructions must be entered in the correct format in the text-based programming language called motion language. You can use the Motion Command Assist Dialog Box to easily select instructions to add to your program.

You can open the Motion Command Assist Dialog Box from the Motion Editor Tab Page. There are two different methods to do so.

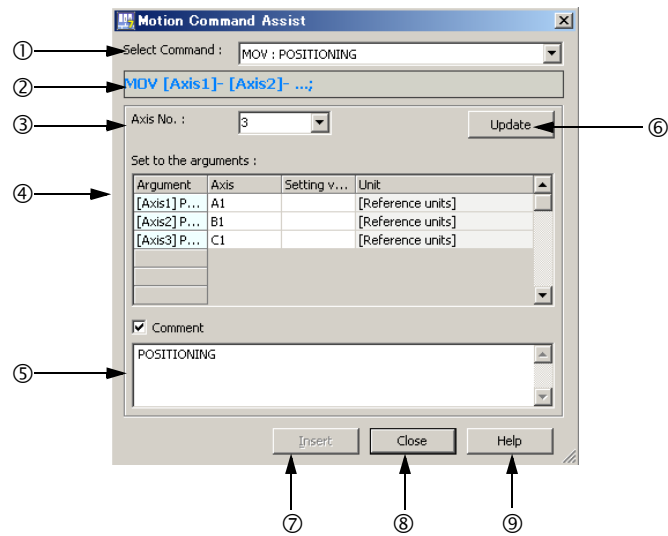
- Right-click and select **Motion Command Assist** from the menu.



- Or, right-click and select the instruction you want to insert under **Insert Command**.

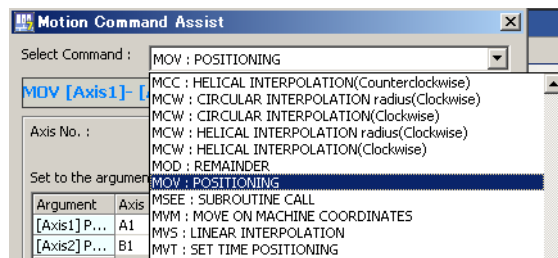


■ Motion Command Assist Dialog Box



① Select Command

Click the arrow to display a list of the instructions that you can insert.



② Instruction Format

This area displays the format of the currently selected instruction.

Example MOV: Positioning

```
MOV [Axis1]- [Axis2]- ...;
```

+: Add

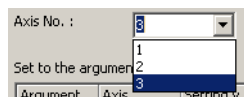
```
ML001 06=ML001 02 + ML001 04;
```

③ Number of Controlled Axes

For axis movement instructions, the number of controlled axes is selected from 1 to the number of axes set in the group definition.

When the number of controlled axes is fixed, the fixed number of axes is displayed and the box is grayed out.

Example MOV (Positioning): Specify the number of controlled axes.



EXM (External Positioning): Fixed number of controlled axes



④ Parameter Settings

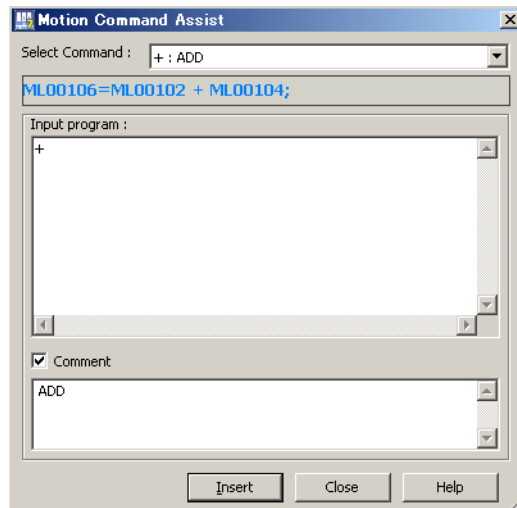
This area allows you to set the parameters (arguments) for the instruction. The setting items are listed in the following table.

Item	Description
Argument	Displays the parameter names that are set as arguments to the instruction. These cannot be changed. Arguments that can be omitted will be designated as optional.
Axis	Displays the logical axis name. Change these as required.
Set value	Specify a constant or register for the set value.
Unit	Displays the unit for the parameter. The unit cannot be changed.

The logical axis names are defined in the group definitions.

The units are displayed according to the motion parameter setting for each axis. If a unit has not been specified in the motion parameter settings, the corresponding cell is displayed in yellow. Place the mouse cursor over a cell to view its tooltip. Follow the instructions to set the motion parameters.

If the selected instruction does not require the number of controlled axes or any parameters to be specified, the **Input program** Text Box is displayed in the Motion Command Assist Dialog Box as shown below. Enter the instruction block referring to the instruction format.



⑤ Comment Check Box and Comment Box

Select the Comment Check Box to enter a comment on the line above the instruction. When this check box is not selected, the text box is grayed out and a comment cannot be entered.

Information The location to insert the comment cannot be changed.

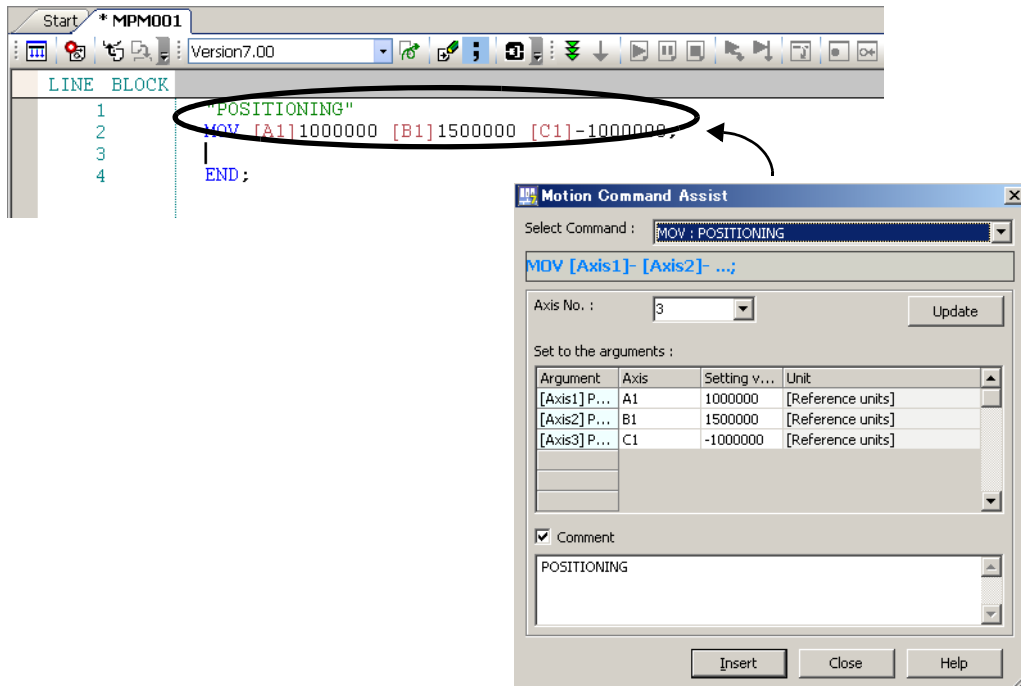
⑥ Update Button

This button updates the display of the Motion Command Assist Dialog Box.

Information Click the **Update** Button to refresh the display after changing any unit-related motion parameter.

⑦ Insert Button

Click the **Insert** Button to insert the instruction in the Motion Command Assist Dialog Box at the cursor position in the Motion Editor.

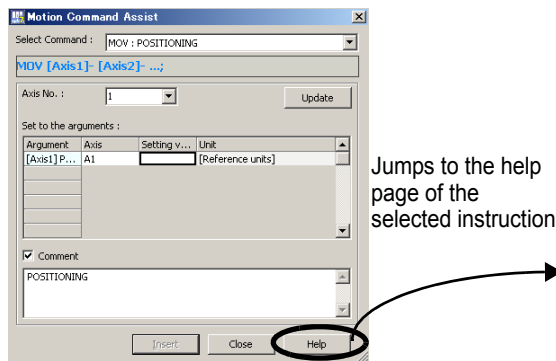


⑧ Close Button

This button closes the Motion Command Assist Dialog Box.

⑨ Help Button

This button displays information on the relevant instruction.



8.2.1 Positioning (MOV)

Motion Programs	Sequence Programs
Applicable	Not Applicable

(1) Overview

The Positioning (MOV) command independently moves each axis from the current position to the end position at positioning speed.

Up to 16 axes can be moved simultaneously. Any axis not specified in the command will not be moved.

The path of movement with the MOV command is different from the linear travel.

Fig. 8.24 Movement Path with MOV Command

CAUTION

The path of movement with the Positioning (MOV) command is not always a straight line. When programming, be sure to check the path to make sure that there are no tools or other obstacles in the way of the workpiece.

Failure to carry out this check may result in damage to equipment, serious personal injury, or even death.

(2) Format

MOV [*Logical axis name 1*] Reference position [*Logical axis name 2*] Reference position [*Logical axis name 3*] Reference position ... ;


Item	Unit	Usable Data
Reference position	Reference unit	<ul style="list-style-type: none"> Directly designated value Double integer type register (Indirect designation)

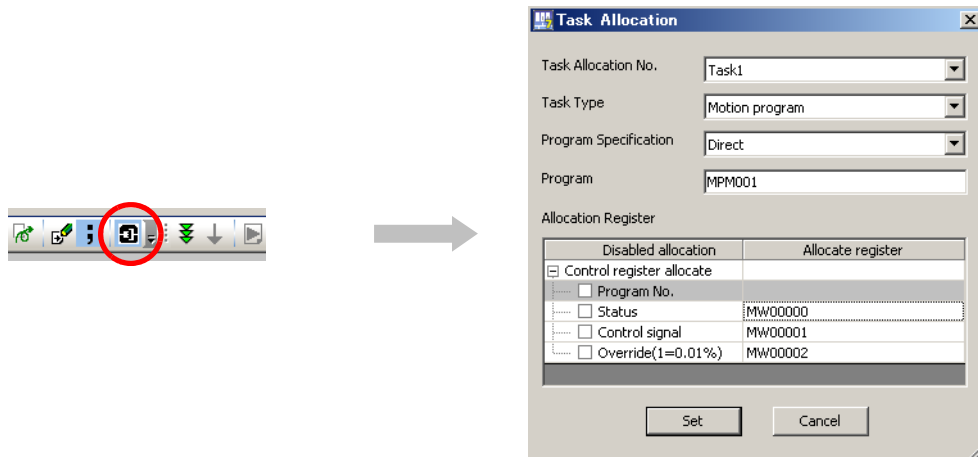
7.3 Task Assignments

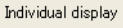
Task allocation is used to call motion programs or sequence programs.

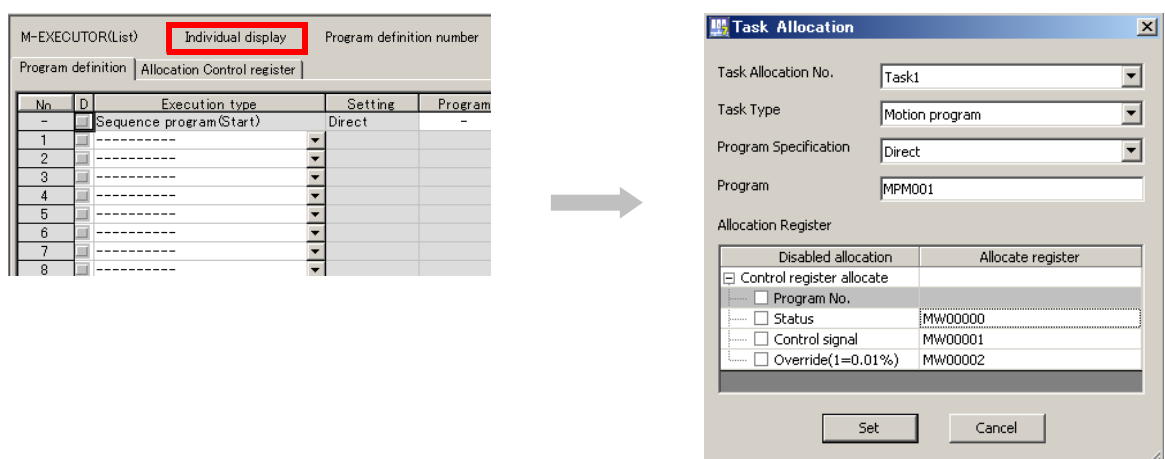
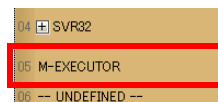
The Task Allocation Dialog Box makes it easy to register the motion programs and sequence programs that you create in the MP3000 system.

There are two methods to display the Task Allocation Dialog Box, as described below.

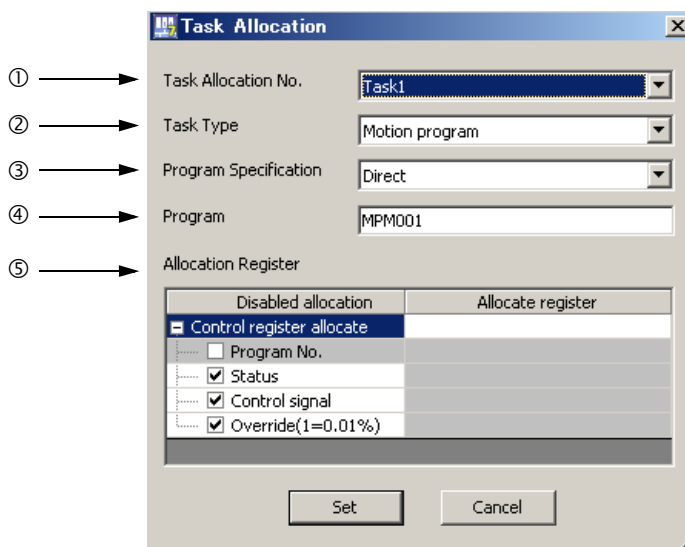
- Click the  icon in the Motion Editor.



- Open the Detail Definition for the M-EXECUTOR under Module Configuration, then click the  icon.



■ Task Allocation Dialog Box



① Task Allocation No.

This box displays the task number that is assigned to the program. You can select the task number when you click the [] Icon on the toolbar in the Motion Editor Tab Page.

② Task Type

Set the execution type of the program.

Execution Type	Supported Programs	Execution Condition
Startup sequence programs	Sequence programs	Startup (These programs are executed once when the power supply is turned ON.)
L-scan sequence programs		Started at a fixed interval. (These drawings are executed once every low-speed scan cycle.)
H-scan sequence programs		Started at a fixed interval. (These drawings are executed once every high-speed scan cycle.)
Motion programs	Motion programs	Request for Start of Program Operation control signal (The program is executed when the Request for Start of Program Operation is turned ON.)

③ Program Specification

Set the program designation method.

The designation method can be different for each program.

Designation Method	Motion Programs	Sequence Programs	Description
Direct Designation	Supported.	Supported.	The program is specified with the program number. Examples: MPM001 or SPM002
Indirect Designation	Supported.	Not supported.	The program is specified by specifying a register that contains the program number. Example: OW0C0C (If 1 is stored in OW0C0C, MPM001 is executed.)

④ Program

Set the program number.

⑤ Allocation Register

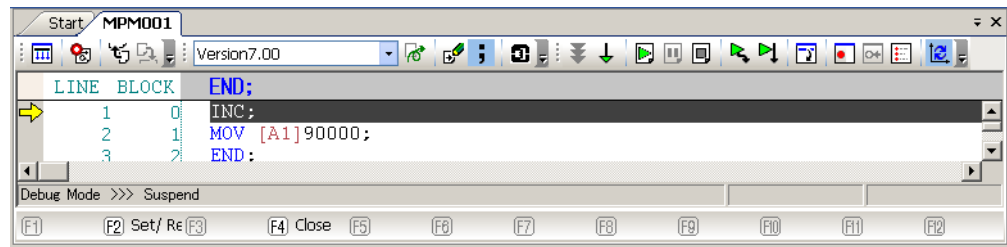
This area is used to assign registers. The assigned registers exchange data in realtime with the M-EXECUTOR control registers. I, O, and M registers can be assigned.


7.4 Debug Operation

The Debug Operation Mode allows you to monitor the line of the motion or sequence program that is current being executed. This makes it easier to find bugs in the program.

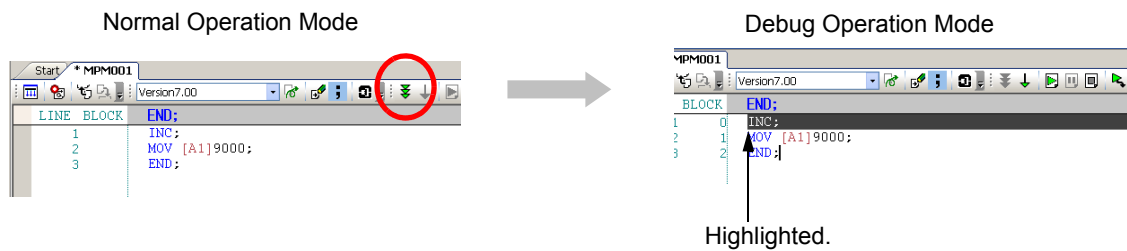
You can pause the execution of a program, set breakpoints, perform single-step execution (single-block execution), and perform other operations to ensure proper operation of the programs that you developed.

In Debug Operation Mode, the program line that is being executed is displayed at the top of the tab page as shown below.



To start Debug Operation Mode, first connect to the Machine Controller, then click the [] Icon on the Motion Editor Tab Page.

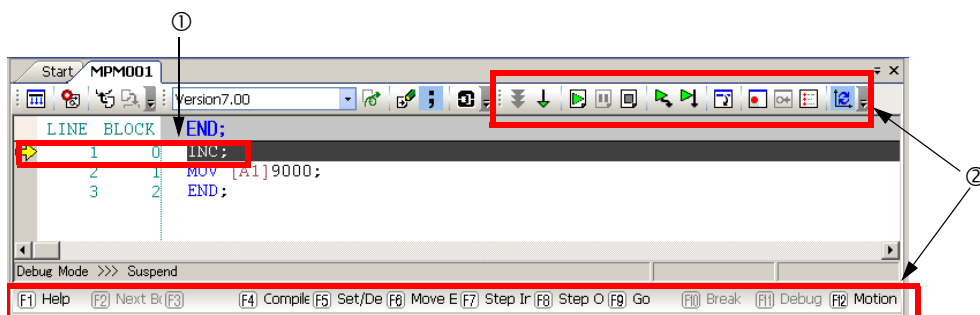
In Debug Operation Mode, the program line that is being executed is highlighted at the top of the tab page.



Note

You must register the program to execute before you can start Debug Operation Mode.

■ Debug Interface



① Current Program Line

The program line that is currently being executed is displayed in blue.

If an alarm has occurred in the motion program, the line will be displayed in red.

Refer to the following manual for details on motion program alarms.

MP3000 Series MP3200/MP3300 Troubleshooting Manual (Manual No.: SIEP C880725 01)

② Toolbar Icons and Function Keys

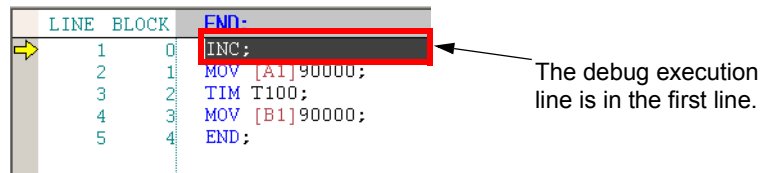
The following table describes the icons and function keys that are used in Debug Operation Mode.

Function	Icon	Key Operation	Description	Motion Programs	Sequence Programs
Debug Operation Mode		F1	Starts Debug Operation Mode.	○	○
Normal Operation Mode		F11	Ends Debug Operation Mode and starts the continuous execution of the program in Normal Operation Mode.	○	○
Move Start Point		F6	Moves the start point for execution.	○	○
Breakpoint Set/Remove		F7	Sets or removes a breakpoint. Displays the breakpoints in the program.	○	○
Step In		F4	Executes one block. For an MSEE or SSEE instruction, debugging will move to the first line of the subprogram.	○	○
Step Over		F5	Executes one block. For an MSEE or SSEE instruction, the subprogram will be executed and debugging will continue at the next block after the MSEE or SSEE instruction.	○	○
Execute		F8	Continuously executes a motion program in Debug Operation Mode.	○	○
Break		F10	Pauses the execution of a motion program in Debug Operation Mode.	○	○
End		F2	Ends execution of the motion program.	○	×
Update Current Position		–	Updates the current position coordinates.	○	×
Set Motion Task		–	Sets the fork number, level number, and task of the selected program.	○	○
Breakpoint Enable/Disable	–	–	Enables or disables breakpoints. Use the Debug Menu or the pop-up menu for this setting.	○	○
Add Quick Watch	–	–	Registers a register in the Quick Reference Pane. Use the pop-up menu for this setting.	○	○

Note: ○: Possible, ×: Not possible.

• Debug Operation Mode

In Debug Operation Mode, the program is executed one line at a time. Debugging starts from the first line in the program.



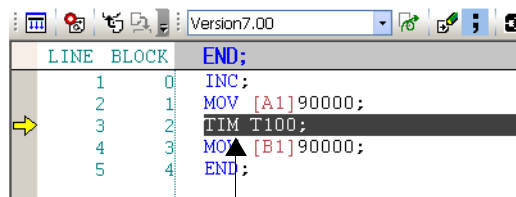
Information

The line where debugging starts when the operating mode is changed to Debug Operation Mode depends on whether the program you are editing is a motion program or a sequence program, as described below.

- When Debug Operation Mode Is Started for a Motion Program that Is Not Currently Running
As shown in the above example, debugging starts from the first line in the program.
- When Debug Operation Mode Is Started for a Motion Program that Is Currently Running
When Debug Operation Mode is started during axis operation, debugging starts from the next block position after the axis completes its movement.
- When Debug Operation Mode Is Started for a Sequence Program that Is Not Currently Running
Debugging cannot be performed.
- When Debug Operation Mode Is Started for a Sequence Program that Is Currently Running
As shown in the above example, debugging starts from the first line in the program.

• Normal Operation Mode

In Normal Operation Mode, the program is executed from the beginning to the end without interruption. Debugging is canceled and program execution restarts from the currently executing line. All breakpoints that have been set are removed.

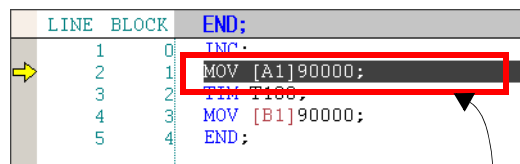


Operation is resumed from the current program execution line.

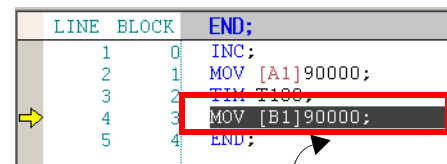
• Move Start Point

This moves the first line of execution to the selected line.

Click the Move First Line for Execution  Icon.



Click the line you want to be executed first.



This moves the first line for execution to the selected line.

Note: The line MOV [A1]90000; is not executed.

• Breakpoint Set/Remove 

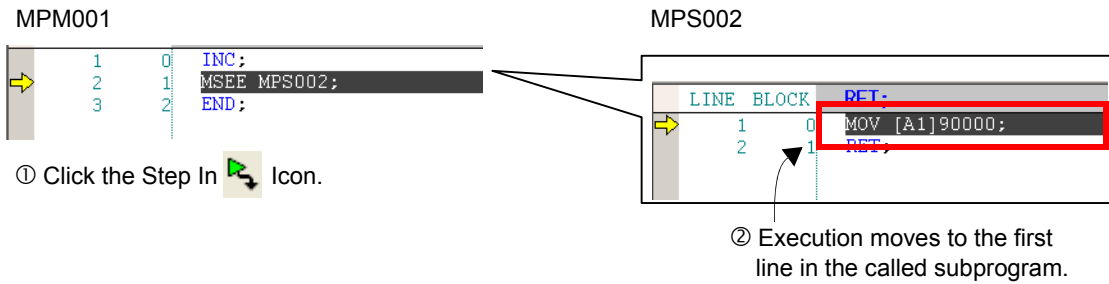
This icon sets or removes a breakpoint. You can set a maximum of up to four breakpoints. Clicking the button for a line for which a breakpoint has been already set will delete the breakpoint.

1	0	INC;
2	1	MOV [A1]90000;
3	2	MOV [B1]90000;
4	3	END;

• Step In 

This icon executes one line of the program.

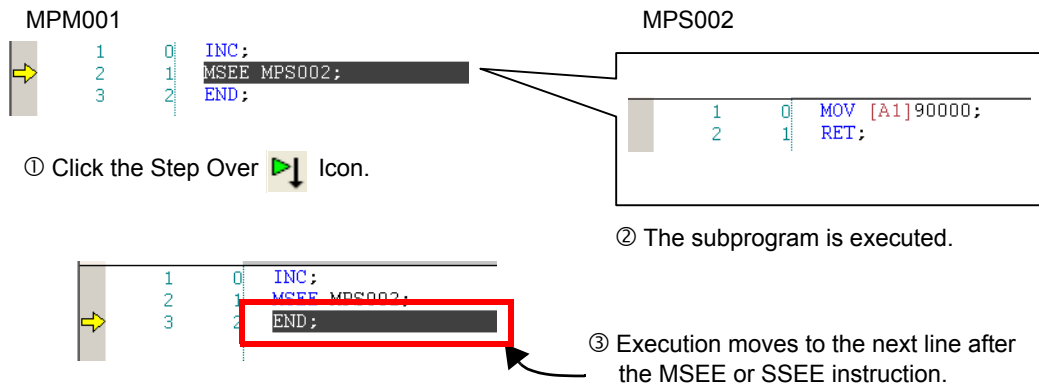
If this icon is clicked at an MSEE or SSEE instruction, execution jumps to the first line of the called sub-program.



• Step Over 

This icon executes one line of the program.


If this icon is clicked at an MSEE or SSEE instruction, the called subprogram is executed and then execution moves to the next line.



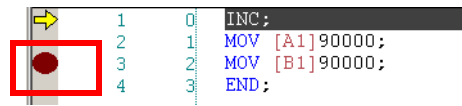
Information You can use the SNGD and SNGE instructions to set multiple processes as one unit for the step execution processing.

The instruction blocks that are between SNGD and SNGE instructions form the processing unit for execution of step in or step over.

SNGD;
Instruction blocks to be processed as one unit for execution of step in or step over
SNGE;

- **Execute**  This icon executes the program without stopping. Execution stops at any line with a breakpoint.

① Set a breakpoint.

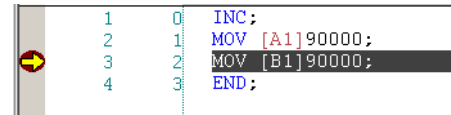


```

1 0 INC;
2 1 MOV [A1]90000;
3 2 MOV [B1]90000;
4 3 END;

```

② Click the Execute  icon.




```

1 0 INC;
2 1 MOV [A1]90000;
3 2 MOV [B1]90000;
4 3 END;

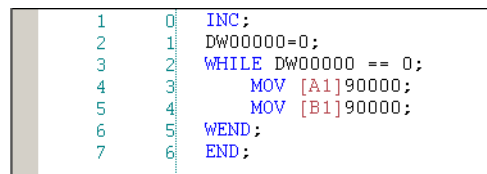
```

③ Execution stops at the line with the breakpoint.

- **Break** 

This icon pauses execution of the program in Debug Operation Mode. To resume the program, click the  icon.


① Execute the motion program.

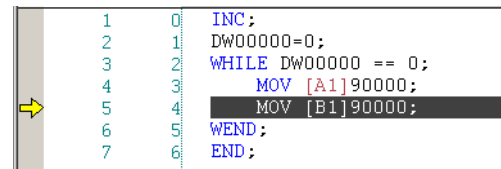


```

1 0 INC;
2 1 DW00000=0;
3 2 WHILE DW00000 == 0;
4 3     MOV [A1]90000;
5 4     MOV [B1]90000;
6 5 WEND;
7 6 END;

```

② Click the Break  icon.



```

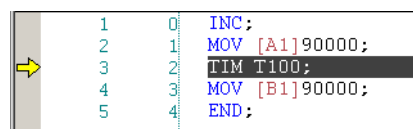
1 0 INC;
2 1 DW00000=0;
3 2 WHILE DW00000 == 0;
4 3     MOV [A1]90000;
5 4     MOV [B1]90000;
6 5 WEND;
7 6 END;

```

③ Program execution stops.

- **End** 

This icon forces execution of the program in Debug Operation Mode to stop.




```


1 0 INC;
2 1 MOV [A1]90000;
3 2 TIM T100;
4 3 MOV [B1]90000;
5 4 END;


```

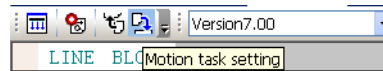
All of the processing after the current program execution line is not executed.


- **Update Current Position** 

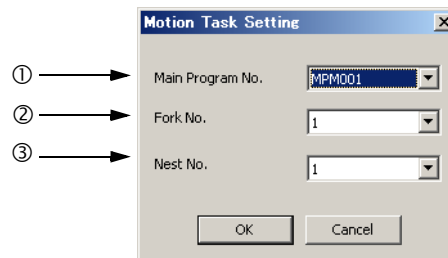
This icon has the same function as the PLD command. When this icon is selected, the operation of the PLD instruction is processed by the system when the **Step In**, **Step Over**, or **Execute** Icon is clicked. Refer to the following section for details on the PLD instruction.

 *Update Program Current Position (PLD) (page 6-120)*

- Set Motion Task  (Subprograms Only)
Set the subprogram information to use for monitoring or debugging subprograms. The currently running main program is displayed, and you can set which main program to call subprograms from.



- Set Call Stack  (Subprograms Only)
This icon allows you to set more detailed subprogram information.



① Main Program No.

This box sets the main program from which to call subprograms.

② Fork No.

This box sets the fork of the main program from which to call subprograms.

For example, set 3 for the fork number to perform debugging and program monitoring of MPS004.

<For MPM001>

```
PFORK Label1 Label2 Label3 Label4;
```

```
Label1: "Fork 1
```

```
  MSEE MPS002;
```

```
  JOINTO LabelX;
```

```
Label2: "Fork 2
```

```
  MSEE MPS003;
```

```
  JOINTO LabelX;
```

```
Label3: "Fork 3
```

```
  MSEE MPS004;
```

```
  JOINTO LabelX;
```

```
Label4: "Fork 4
```

```
  MSEE MPS005;
```

```
  JOINTO LabelX;
```

```
LabelX: PJOINT;
```

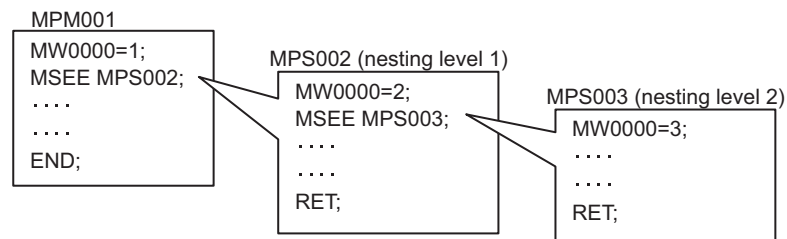
```
....
```

```
END;
```

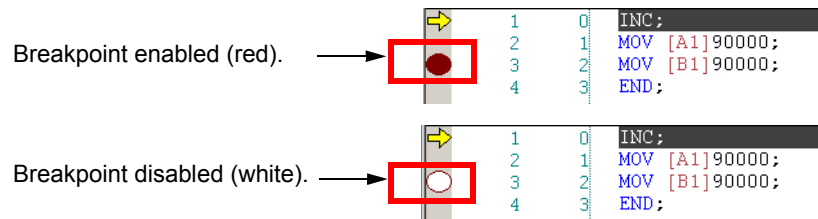
③ Nest No.

This box sets the nesting level of the call to the subprogram.

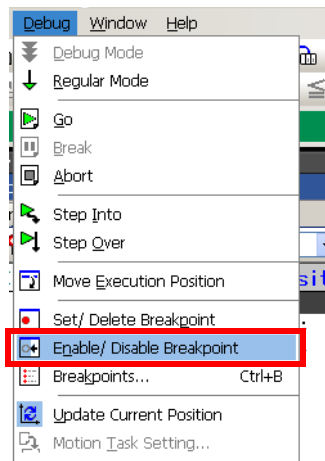
For example, set 2 for the nesting level to perform debugging and program monitoring of MPS003.



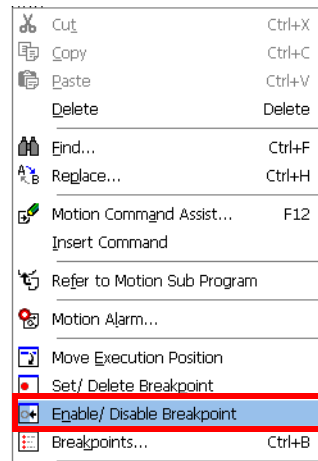
- Breakpoint Enable/Disable
This icon enables or disables a breakpoint.



Debug Menu

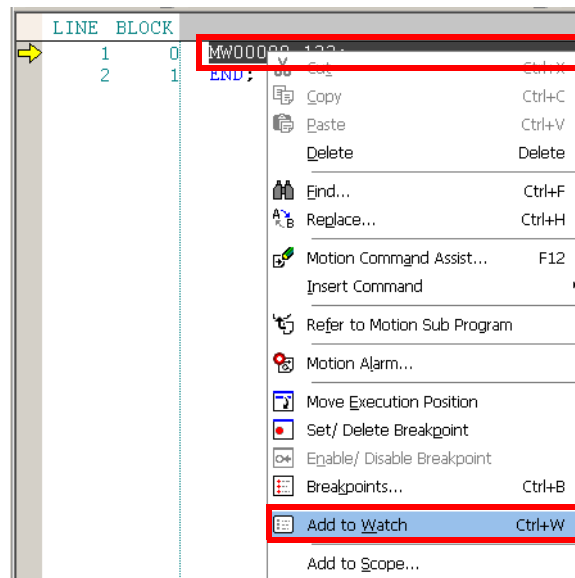


Pop-up Menu in Motion Editor Tab Page



- Add Quick Watch
Any register that is displayed on the Motion Editor can be registered to the Watch Page of quick references. You can monitor the values of registers that are registered as quick references.

1. Right-click the register to monitor and select **Add to Watch** from the menu.



2. The register is added to the Quick Reference Watch Tab Page.

Variable	Value	Comment
OB90000	ON	A1~Servo ON
OB90800	ON	B1~Servo ON
MW00000	123	

7.5 Drive Control Panel

The Drive Control Panel allows you to perform test runs of programs and monitor the operating status of programs that are currently in execution.

To execute a motion program, the program must be registered in the MP3000 system and the program start request must be issued using the user application.

If you want to execute a motion program before you create the user application, you can perform a test run from the Drive Control Panel Dialog Box.

You can send commands, such as Request for Start of Program, Request for Stop of Program, and Alarm Reset Request, from the Drive Control Panel.

Task	Task1	Task2	Task3	Task4
Main program	MPM001	No allocate	No allocate	No allocate
<input type="checkbox"/> Motion Program Control Signals	OW0C01 H0000	SW03323 H0000	SW03381 H0000	SW03439 H0000
Bit 0 : Start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pause request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stop request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 3 : Single block mode selection	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Single block start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 5 : Alarm reset request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 6 : Program continuous operation start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Skip1 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Skip2 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : System work number setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : Interpolation override setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="checkbox"/> Status	IW0C00 H0000	SW03322 H0000	SW03380 H0000	SW03438 H0000
Bit 0 : Running	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pausing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stopped	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Stopped under single block mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Alarm	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Stopped at break point	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit B : Debugging mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : Start request signal history	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : No system work error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit F : Main program number limit error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

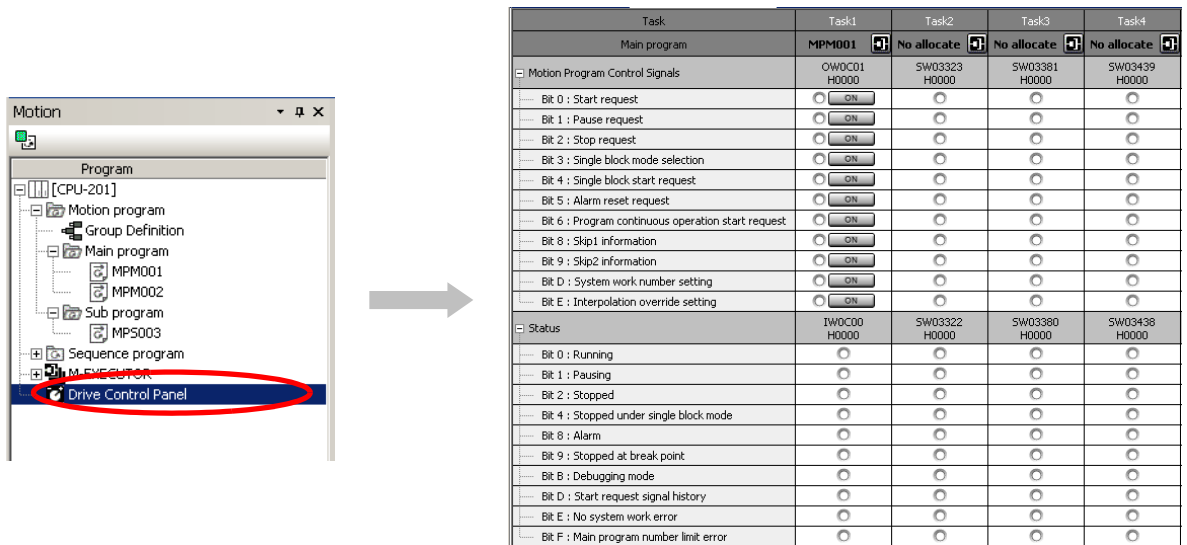
Note: The Drive Control Panel does not support setting breakpoints or one-step execution (single-block execution) like Debug Operation Mode.







Important

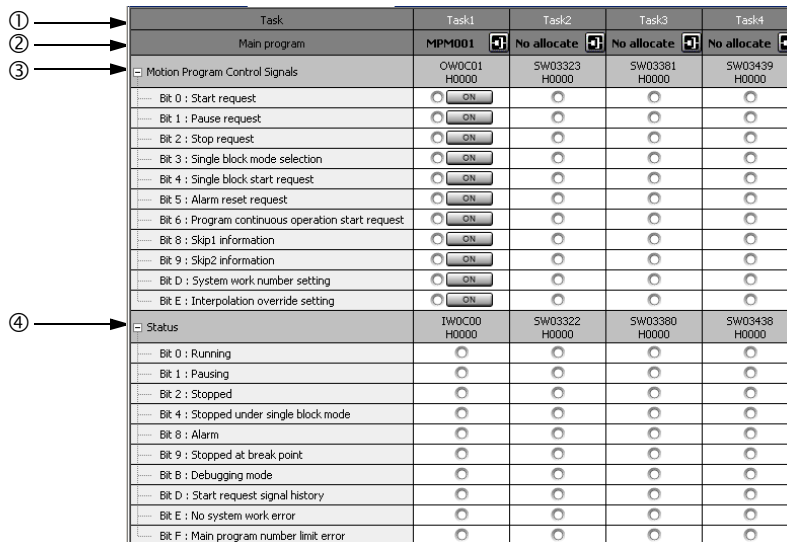
1. Make sure the area is safe before moving the axes with a test run operation.
2. Be sure not to overwrite the motion program control registers from a sequence program or ladder program. Doing so may disable the control from the Drive Control Panel.
3. Do not simultaneously execute axes movement instruction for one axis from more than one program. Unexpected operation may occur.





Click the [] Icon in the Motion Pane to open the Drive Control Panel.



Task	Task1	Task2	Task3	Task4
Main program	MPM001 	No allocate 	No allocate 	No allocate 
Motion Program Control Signals	OW0C01 H0000	SW03323 H0000	SW03381 H0000	SW03439 H0000
Bit 0 : Start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pause request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stop request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 3 : Single block mode selection	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Single block start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 5 : Alarm reset request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 6 : Program continuous operation start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Skip1 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Skip2 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : System work number setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : Interpolation override setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Status	IW0C00 H0000	SW03322 H0000	SW03380 H0000	SW03438 H0000
Bit 0 : Running	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pausing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stopped	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Stopped under single block mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Alarm	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Stopped at break point	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit B : Debugging mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : Start request signal history	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : No system work error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit F : Main program number limit error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

■ Drive Control Panel



Task	Task1	Task2	Task3	Task4
Main program	MPM001 	No allocate 	No allocate 	No allocate 
Motion Program Control Signals	OW0C01 H0000	SW03323 H0000	SW03381 H0000	SW03439 H0000
Bit 0 : Start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pause request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stop request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 3 : Single block mode selection	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Single block start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 5 : Alarm reset request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 6 : Program continuous operation start request	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Skip1 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Skip2 information	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : System work number setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : Interpolation override setting	<input type="radio"/> ON	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Status	IW0C00 H0000	SW03322 H0000	SW03380 H0000	SW03438 H0000
Bit 0 : Running	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 1 : Pausing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 2 : Stopped	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 4 : Stopped under single block mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 8 : Alarm	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit 9 : Stopped at break point	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit B : Debugging mode	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit D : Start request signal history	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit E : No system work error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Bit F : Main program number limit error	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

① Task

This row displays the task numbers.

② Main Program

This row displays the numbers of the main programs for which to perform the test run.

The program number must be set in the M-EXECUTOR Program Execution Definitions in advance.

③ Control Signals

This row displays the control signal status details.

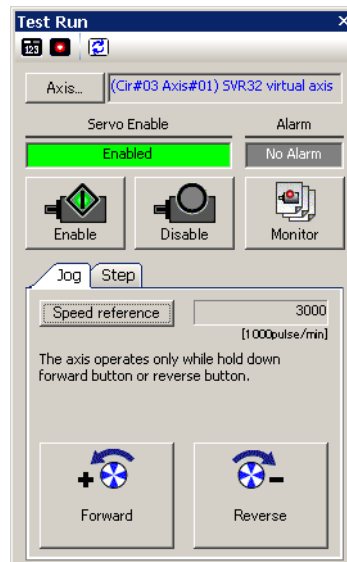
④ Status

This row displays the status of the executed control signal.

7.6 Test Runs

You can perform a test run of the axes that are connected to the Machine Controller from the Test Run Dialog Box.

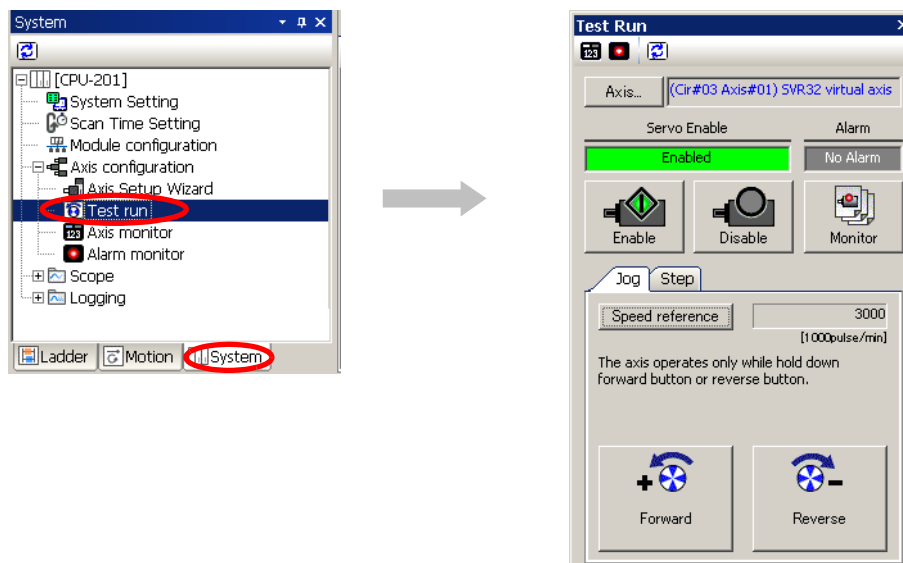
This allows you to turn the Servo ON or OFF and perform jogging and step operations without writing a program.



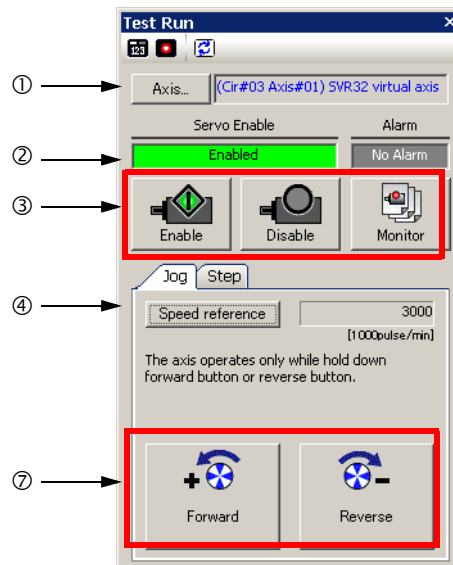
Important

1. Make sure the area is safe before moving the axes with a test run operation.
2. Before starting operation, design the system to enable stopping axis movement whenever necessary.
3. Stop the execution of all ladder and sequence programs before you start a test run.

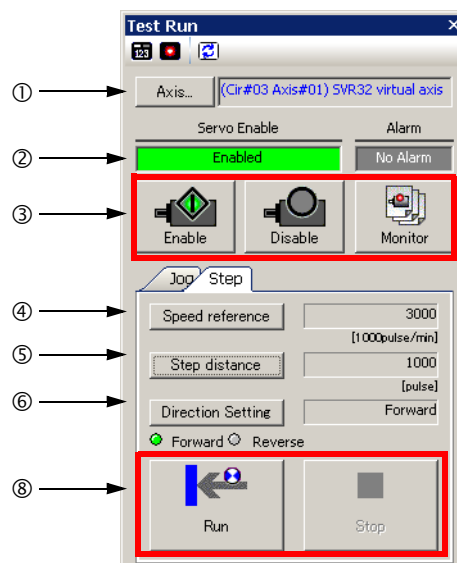
To display the Test Run Dialog Box, double-click **Test Run** in the System Pane.



■ Test Run Dialog Box



Click the **Jog** or **Step** Tab to switch between jogging and step execution.



- ① **Axis Selection**
This button is used to select the axis for the test run.
- ② **Servo Enabled/Disabled and Alarm Display**
These indicators display the ON/OFF status of the Servo and the current alarm status for the axis.
- ③ **Enable, Disable, and Monitor**
These buttons turn the Servo ON or OFF. These operations will change the setting parameters for the axis. Click the **Monitor** Button to display details on alarms for the axis.

④ Speed Reference

Use this button to set the speed reference. These operations will change the setting parameters for the axis.

⑤ Step Distance

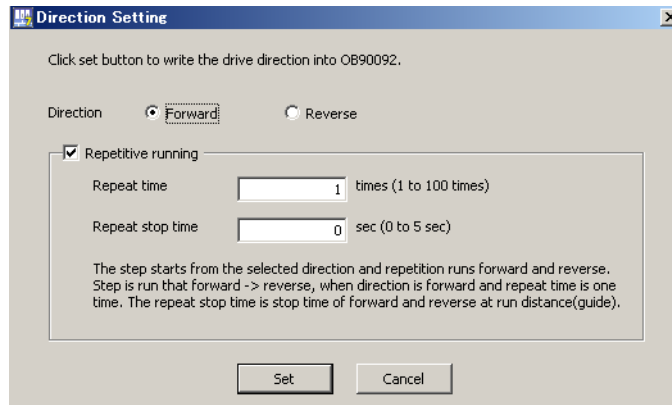
This button sets the step travel distance for step execution. These operations will change the setting parameters for the axis.

⑥ Direction Setting

This button displays the Direction Setting Dialog Box to set the axis direction for step execution.

Select either the **Forward** or **Reverse** Option in the Direction Setting Dialog Box. These operations will change the setting parameters for the axis.

You can also specify repetitive run operation in this dialog box.



⑦ Jog

These buttons are used to perform jogging.

The axis moves in the specified direction while the **Forward** or **Reverse** Button is clicked. The axis stops when the button is released.

⑧ Step

These buttons perform step execution.

Click the **Run** Button to perform one step for the specified axis. Unlike with the jog operation, the button does not need to be continuously pressed.

When the **Repetitive running** check box is selected in the **Direction Setting** dialog box, the step operation is repeated for the specified number of times, and then the axis stops. You can also stop the axis if repetitive operation is in progress.

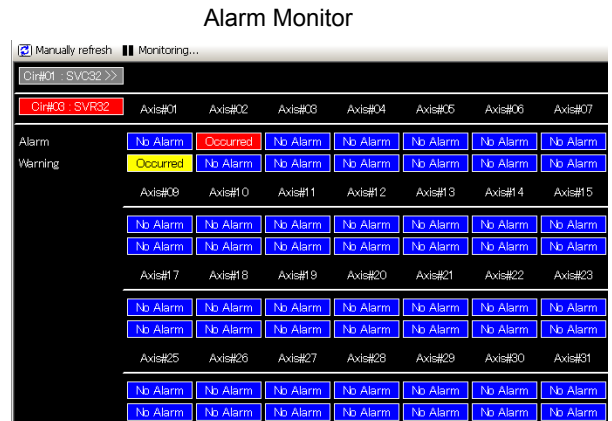
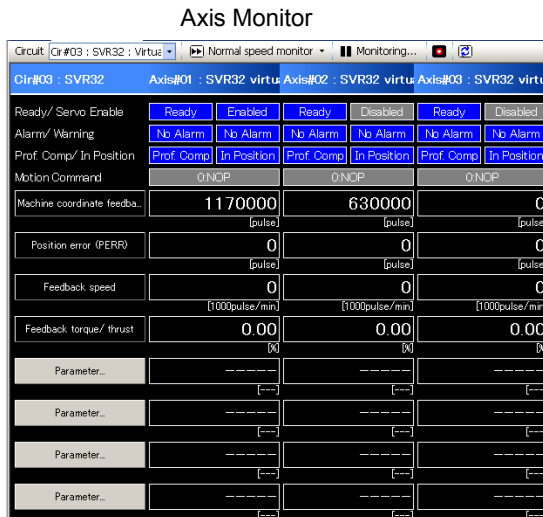
7.7

Axis Monitor and Alarm Monitor

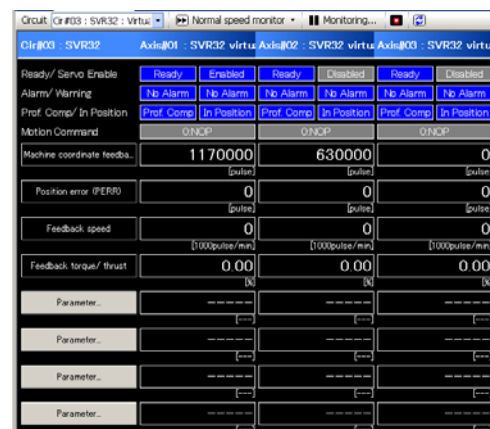
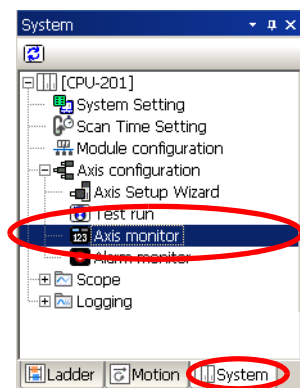
Use the Axis Monitor to monitor the operating status of axes connected to the Machine Controller.

The axis status (operation ready, Servo ON, alarms, warnings, distribution/positioning completed, and motion command) and selected monitor parameters are displayed in the Axis Monitor.

Use the Alarm Monitor to monitor the alarm status of axes connected to the Machine Controller.



To open the Axis Monitor or Alarm Monitor, double-click **Axis monitor** or **Alarm monitor** in the System Pane.



Axis Monitor Tab Page

This section describes the Axis Monitor Tab Page.

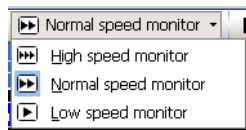


① Circuit

This box is used to select the circuit for which to display the monitor parameters.

② Monitoring Cycle Selection

The monitor cycle is selected here.



③ Stop/Start Monitor

Click this button to start or pause monitoring.

④ Alarm Monitor

Click this icon to display the Axis Alarm Monitor.

⑤ Refresh

Click this icon to update the Axis Monitor display.


⑥ Status Display

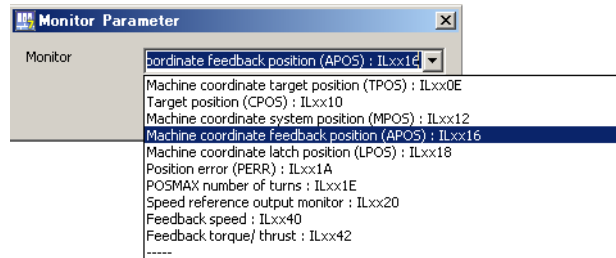
This area displays the operation ready, Servo ON, alarms, warnings, distribution/positioning completed, and motion command status for the axes. The display changes based on the current status.

⑦ Monitor Parameter Selection Area

You can select up to eight monitor parameters to monitor at the same time.

By default, the APOS (Machine Coordinate System Feedback Position), PERR (Position Deviation), Feedback Speed, and Torque/Force Reference Monitor are displayed.

Click the [] Button, and select the desired monitoring parameter from the list in the Monitor Parameter Dialog Box.



Monitor Parameters in **Monitor** List

Monitor Parameter	Register	Unit
Machine Coordinate System Target Position (TPOS)	IL□□□0E	Reference units
Machine Coordinate System Calculated Position (CPOS)	IL□□□10	Reference units
Machine Coordinate System Reference Position (MPOS)	IL□□□12	Reference units
32-bit DPOS (DPOS)	IL□□□14	Reference units
Machine Coordinate System Feedback Position (APOS)	IL□□□16	Reference units
Machine Coordinate System Latch Position (LPOS)	IL□□□18	Reference units
Position Deviation (PERR)	IL□□□1A	Reference units
Number of POSMAX Turns	IL□□□1E	[rev]
Speed Reference Output Monitor	IL□□□20	[pulse/s]
Feedback Speed	IL□□□40	Speed Unit Selection
Torque/Force Reference Monitor	IL□□□42	Torque Unit Selection

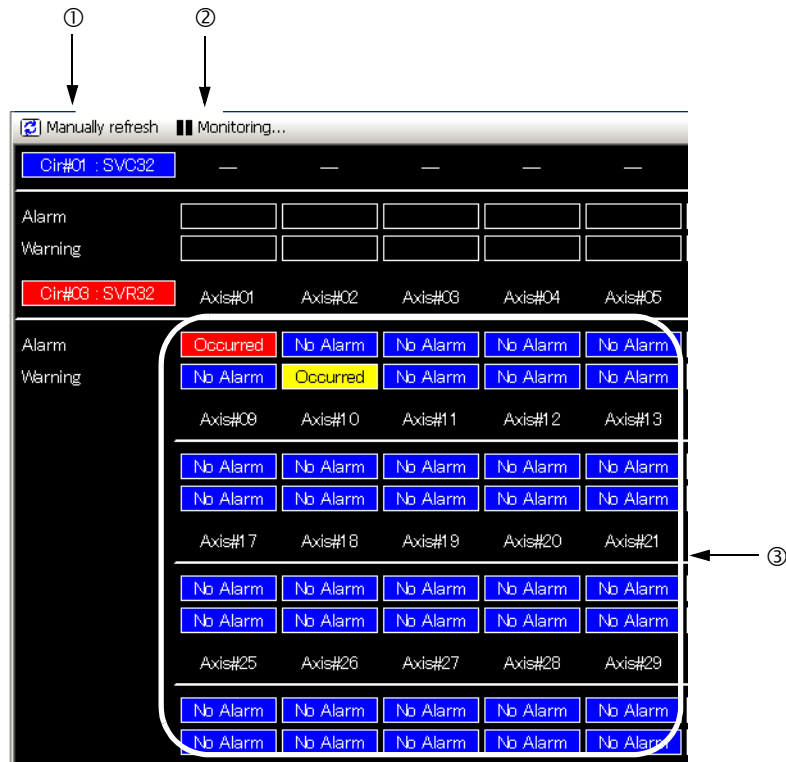
Information If you want to set a monitor parameter that is not in this list, specify the register directly (IW08000, for example).

⑧ Monitor Parameter Display

This area displays the status of the specified monitor parameters.

■ Alarm Monitor Tab Page

This section describes the interface of the Alarm Monitor Tag Page.



① Manually Refresh

Click this button to manually update the alarm and warning information.

② Stop/Start Monitor

Click this button to start or pause monitoring.

③ Alarm/Warning Display

This area displays the alarm and warning status.

Display	Axis Status
No Alarm (Blue)	No alarms or warnings have occurred.
Occurred (Red)	An alarm has occurred.
Occurred (Yellow)	A warning has occurred.

7.8 Cross References

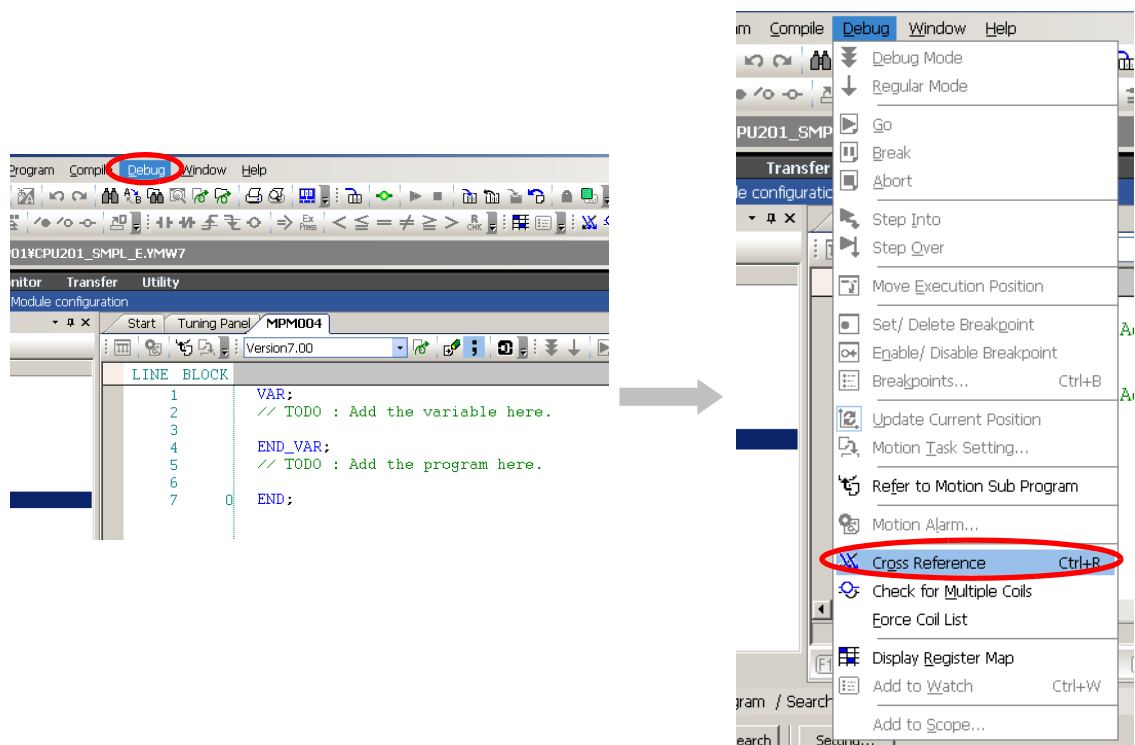
This section describes cross-referencing.

Use cross-referencing to search for variables and registers that are used in programs.

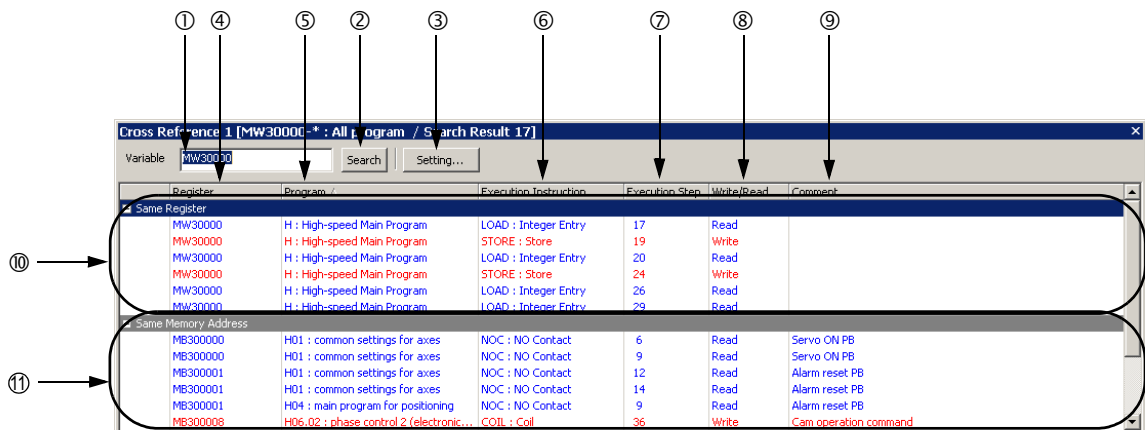
When a search is performed, the program number and block number of any program that uses the register that was searched for are displayed in the Cross Reference Pane.

Register	Program /	Execution Instruction	Execution Step	Write/Read	Comment
Same Register					
MW30000	H : High-speed Main Program	LOAD : Integer Entry	17	Read	
MW30000	H : High-speed Main Program	STORE : Store	19	Write	
MW30000	H : High-speed Main Program	LOAD : Integer Entry	20	Read	
MW30000	H : High-speed Main Program	STORE : Store	24	Write	
MW30000	H : High-speed Main Program	LOAD : Integer Entry	26	Read	
MW30000	H : High-speed Main Program	LOAD : Integer Entry	29	Read	
Same Memory Address					
MB300000	H01 : common settings for axes	NOC : NO Contact	6	Read	Servo ON PB
MB300000	H01 : common settings for axes	NOC : NO Contact	9	Read	Servo ON PB
MB300001	H01 : common settings for axes	NOC : NO Contact	12	Read	Alarm reset PB
MB300001	H01 : common settings for axes	NOC : NO Contact	14	Read	Alarm reset PB
MB300001	H04 : main program for positioning	NOC : NO Contact	9	Read	Alarm reset PB
MB300008	H06.02 : phase control 2 (electronic...	COIL : Coil	36	Write	Cam operation command
MB300008	H06.02 : phase control 2 (electronic...	NCC : NC Contact	68	Read	Cam operation command
MB300008	H06.02 : phase control 2 (electronic...	NCC : NC Contact	109	Read	Cam operation command
MB300008	H06.02 : phase control 2 (electronic...	NCC : NC Contact	114	Read	Cam operation command
MB300008	H06.02 : phase control 2 (electronic...	NOC : NO Contact	124	Read	Cam operation command

Select **Cross Reference** from the Debug Menu to open the Cross Reference Pane.



■ Cross Reference Window



① Variable Box

Enter the variable or register that you want to search for here.

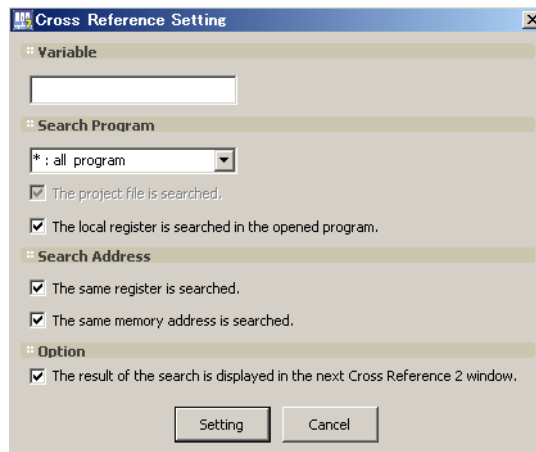
② Search Button

Click this button to perform the cross-reference search.

③ Settings Button

Click this button to set the cross-reference conditions.

When you click the button, the Cross Reference Setting Dialog Box is displayed.



Item	Description
Variable	Enter the variable or register that you want to search for.
Search Program	Specify the program to search.
Search Address	Specify whether to search for the same register or the same address.
Option	Specify how to display the next cross-reference search results.

④ Register

The variable or register address that was searched for is displayed here.

⑤ Program

The number of the program that uses the variable or register address that was searched for is displayed here.

⑥ Execution Instruction

The instruction that uses the variable or register address that was searched for is displayed here.

⑦ Execution Step

The number of the block that uses the variable or register address that was searched for is displayed here.

⑧ Write/Read

This column designates whether the variable or register address that was searched for is written to or read from. If the variable or register address is written to, the text is displayed in red. If the variable or register address is read from, the text is displayed in blue.

⑨ Comment

The comment for the variable or register address that was searched for is displayed here.

⑩ Same Register

This area displays all registers that match the variable name or register type, data type, and address of the variable or register address that was searched for.

⑪ Same Memory Address

This area displays all registers that have the same memory address as the variable or register address that was searched for.

For example, if you searched for MW00000, all locations that use ML00000 would be displayed here.

Specifications

Appendix A

This appendix describes the Units and Modules that support motion programming and the specifications for motion programs.

A.1	Applicable Units and Modules	A-2
A.2	Machine Controller Specifications	A-3

A.1 Applicable Units and Modules

The following Units and Modules support motion programs.

The axes that are connected to any of the Units or Modules that are listed below can be controlled by a motion program.

- MP3000/CPU-20□ SVC32
- MP3000/CPU-20□ SVR32
- MP3000/CPU-30□ SVC
- MP3000/CPU-30□ SVR
- MP2000/SVA-01
- MP2000/SVB-01
- MP2000/SVC-01
- MP2000/PO-01

A.2

Machine Controller Specifications

This section provides the specifications for programs for the Machine Controller.

Specification	CPU-20□ and CPU-30□	Remarks	
Motion Programs	Number of Programs	512 max.	You can create a combined total of 512 motion programs and sequence programs.
	Number of Groups	16 groups	–
	Number of Tasks	32 tasks max. (This is the number of simultaneously executable motion programs.)	–
	Number of Parallel Forks per Task	8 parallel forks max. (select from these 4 modes) <ul style="list-style-type: none"> • 4 main program forks, 2 subprogram forks • 8 main program forks • 2 main program forks, 4 subprogram forks • 8 subprogram forks 	Change the mode using the MPE720.
	Execution Registration	<ul style="list-style-type: none"> • Use the MSEE instruction from a ladder program. • Use the M-EXECUTOR. 	–
	Starting Method	Program execution starts on the rising edge of bit 0 (Request for Start of Program Operation) in the control signals.	–
	Override for Positioning Speeds	Can be specified from 0.01% to 327.67%.	–
	Operating Modes	Absolute Mode and Incremental Mode	The mode is changed with the ABS and INC instructions.
	Reference Units	<ul style="list-style-type: none"> • SVC, SVC 32, SVC-01, SVB-01, SVR, and SVR 32 pulses, mm, deg, inches, or μm • SVA-01/PO-01 pulse, mm, deg, inch 	–
	Minimum Reference Unit	<ul style="list-style-type: none"> • pulse 1 • mm, deg, inch, μm 1, 0.1, 0.01, 0.001, 0.0001, 0.00001 	–
	Reference Range	-2,147,483,648 to 2,147,483,647 (signed, 32-bit data)	–
	Number of Simultaneously Controlled Axes per Task	<ul style="list-style-type: none"> • Positioning, Linear Interpolation, Zero Point Return, Skip Function, and Set-time Positioning 32 axes max. • Circular Interpolation 2 axes • Helical Interpolation 3 axes • External Positioning 1 axis 	–

Continued on next page.

Continued from previous page.

Specification	CPU-20□ and CPU-30□	Remarks	
Sequence Programs	Number of Programs	512 max. (There are three settings for the execution timing: startup processing, high-speed scan processing, or low-speed scan processing.)	You can create a combined total of 512 motion programs and sequence programs.
	Number of Tasks	32 tasks max. (This is the number of simultaneously executable sequence programs.)	—
	Number of Parallel Forks per Task	The PFORK instruction cannot be used.	—
	Execution Registration	Use the M-EXECUTOR.	—
	Starting Method	Automatically started by the system.	The system starts sequence programs that are registered in the M-EXECUTOR.
Accessible Registers	M Registers	1,048,576 words	These registers are backed up with a battery.
	S Registers	65,535 words	These registers are backed up with a battery.
	G Registers	2,097,152 words	These registers are shared by all programs. They are not backed up with a battery.
	I Registers	65,536 words + Setting parameters + Registers for CPU interface	—
	O Registers	65,536 words + Monitor parameters + Registers for CPU interface	—
	C Registers	16,384 words	—
	D Registers	Can be specified from 0 to 16,384 words.	These are internal registers that are unique within each DWG. They can be referenced only within the local drawing.

Sample Programs

Appendix B

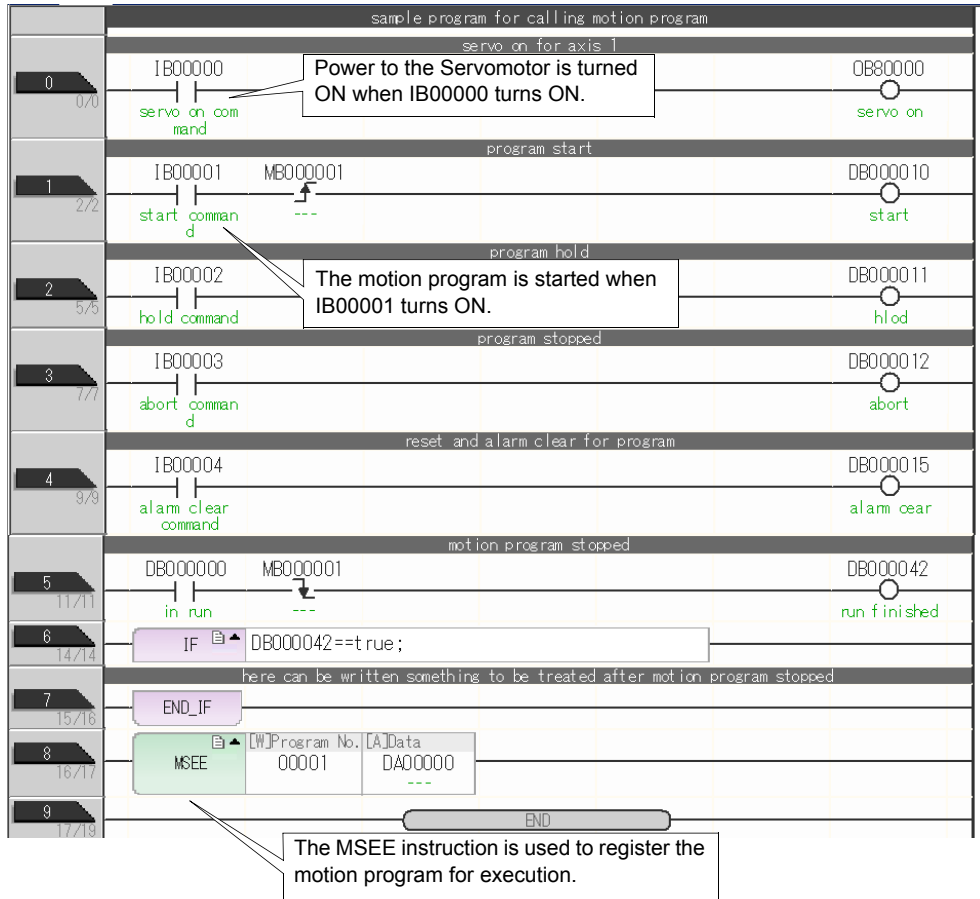
This appendix provides programming examples for motion programs and sequence programs.

B.1	Motion Program Control Program	B-2
B.2	Parallel Processing	B-3
B.3	Performing Speed Control with a Motion Program . .	B-4
B.4	Simple Synchronized Operation with a Virtual Axis . .	B-5
B.5	Sequence Programs	B-7

B.1 Motion Program Control Program

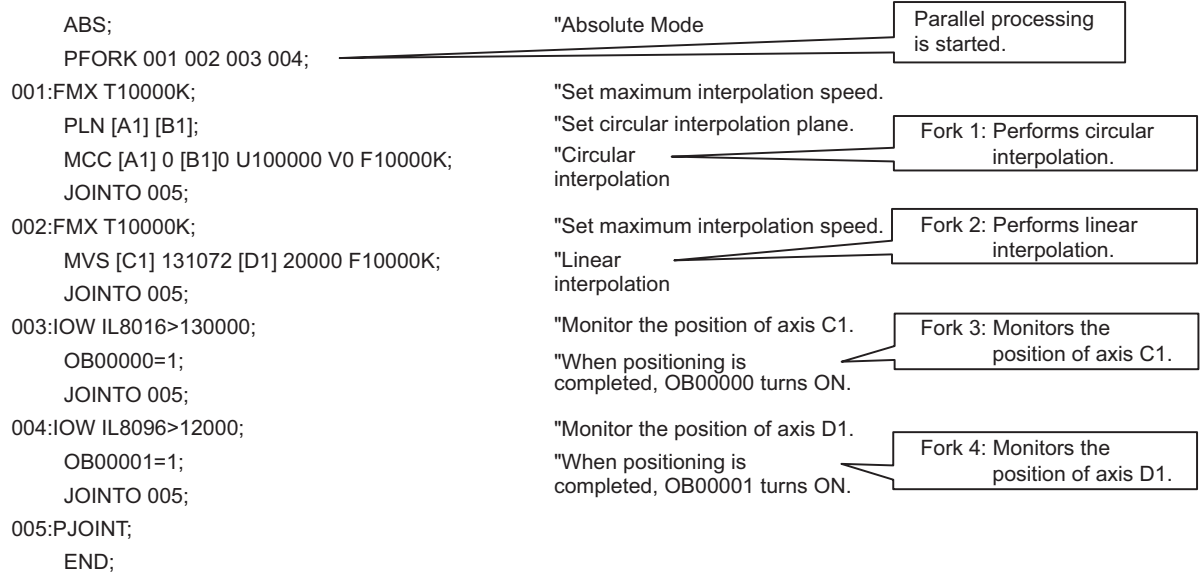
This sample program controls the execution of a motion program.

An example ladder program is given below.

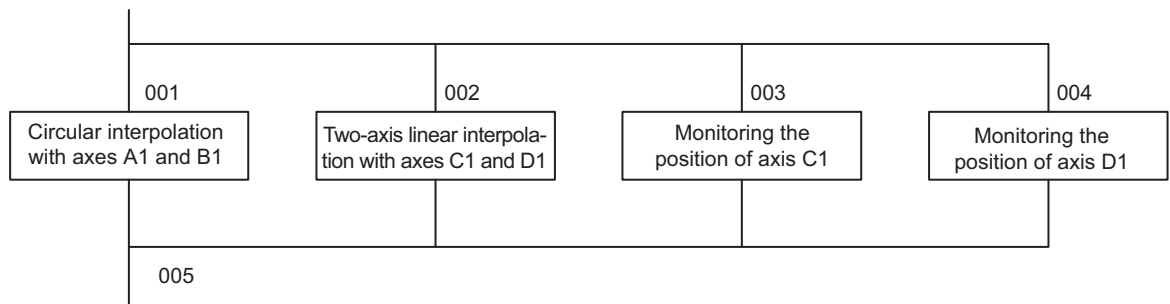


B.2 Parallel Processing

In this example, the PFORK instruction is used in a motion program to perform parallel execution.



The operation of the above sample program is shown in the following figure.



B.3 Performing Speed Control with a Motion Program

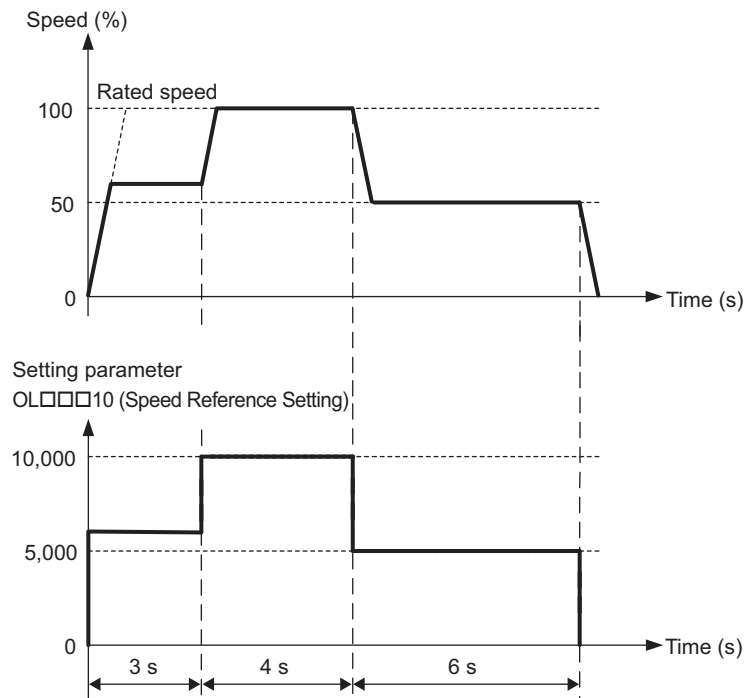
In this example, a motion program is used to perform speed control.

For this example, bits 0 to 3 (Speed Unit Selection) in the OW□□□03 setting parameter are set to 0.01% (percentage of rated speed).

```

OW8008=23;"Speed control mode
OL8010=6000;"Change the speed to 60% of the rated speed.
TIM T300;"Wait 3 seconds.
OL8010=10000;"Change the speed to the rated speed.
TIM T400;"Wait 4 seconds.
OL8010=5000;"Change the speed to 50% of the rated speed.
TIM T600;"Wait 6 seconds.
OW8008=0;"Stop speed control mode.
END;
    
```

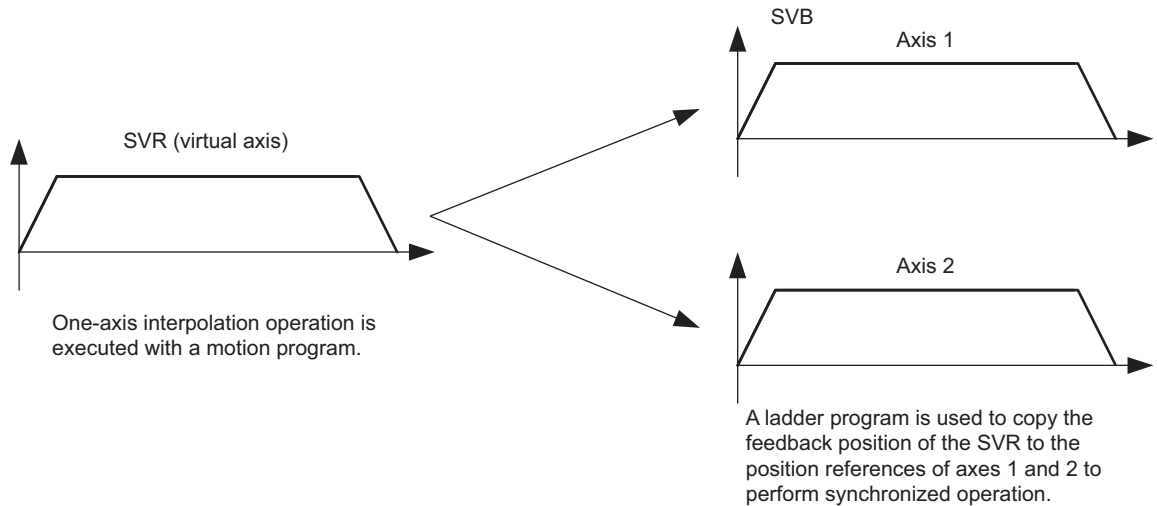
The operation of the above sample program is shown in the following figure.



B.4

Simple Synchronized Operation with a Virtual Axis

This sample program moves an SVR (virtual axis) and distributes the feedback position of the SVR to two physical axes to perform synchronized operation with two axes.



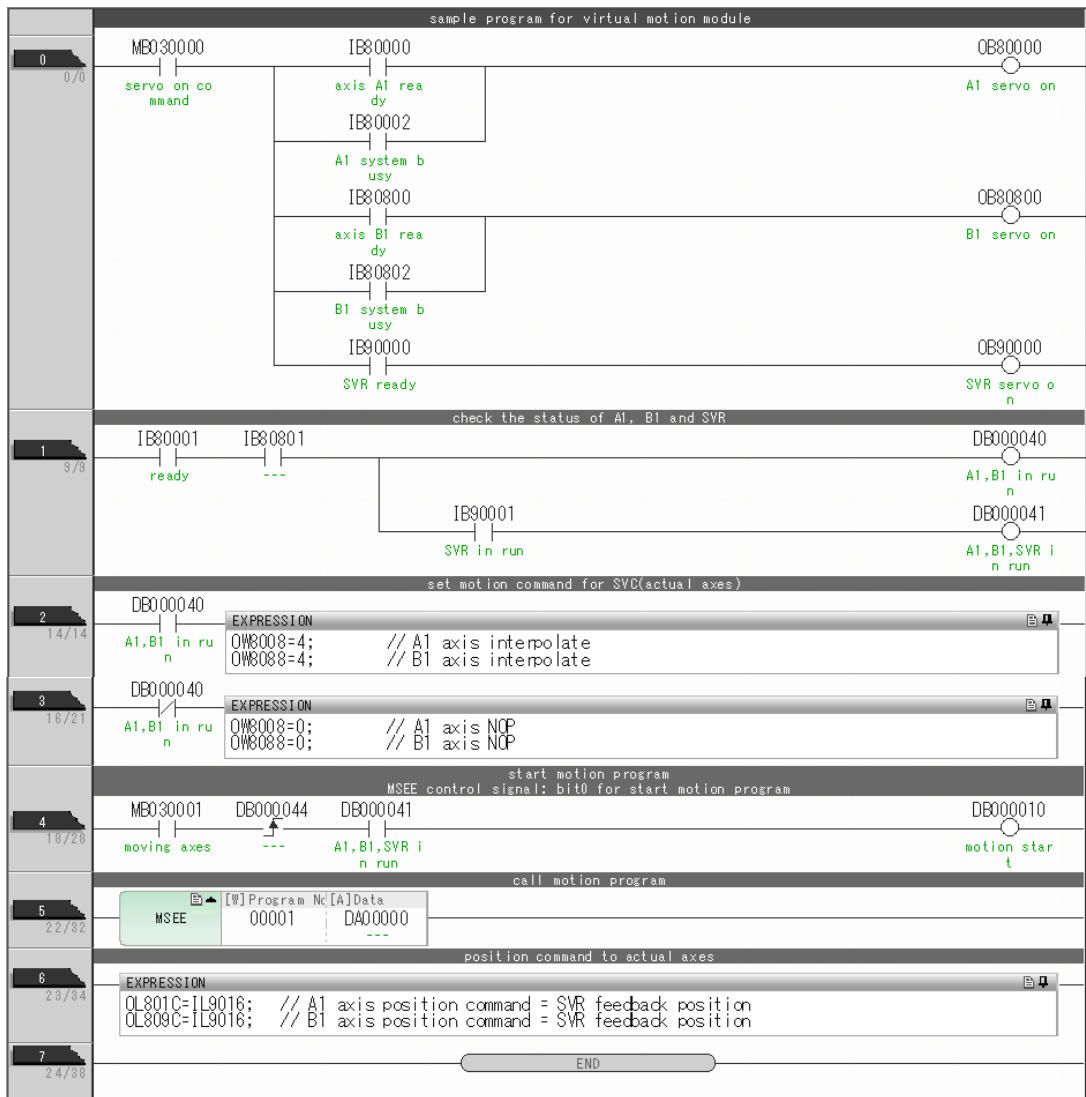
■ Motion Program

```

FMX T1000K;"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 Program



Important

This programming example does not include recovery processing for axis errors. If you decide to incorporate this programming example into your application, be sure to add the necessary programming to ensure safe operation in the event of an axis error.

B.5 Sequence Programs

In this example, a sequence program is used to execute jogging and step operations for a single-axis Servomotor.

■ Main Sequence Program (SPM001)

"SPM001: Main program"

```
SSEE SPS002;           "Settings common to all axes
SSEE SPS003;           "Jogging and step operation processing
END;
```

■ Sequence Subprogram (SPS002)

"SPS002: Settings common to all axes

```
"-----
"Motion Command 0 Detection
"-----
IF IW8008 = = 0;
    MB300010 = 1;
ELSE;
    MB300010 = 0;
IEND;

"-----
"Turn ON the Servo.
"-----
OB80000 = MB300000 & (IB80000 | IB80002);           "Servo ON

"-----
"Resetting Alarms
"-----
OB8000F = MB300001;           "Reset alarm.

"-----
"Speed Unit and Acceleration/Deceleration Rate Unit Selection
"
"Bits 0 to 3: Speed Unit Selection (0: reference units/s, 1: reference units/min, 2: percentage of maximum speed)
"Bits 4 to 7: Acceleration/Deceleration Rate Unit Selection (0: reference units/s^2, 1: ms)
"-----
DW00010 = OW8003 & FF00H;           "Function Settings 1, Work
OW8003 = DW00010 | 0011H;           "Function Settings 1

"-----
"Set linear acceleration/deceleration rate.
"-----
IF MB300020 = = 1;
    OL8036 = 100;           "Linear Acceleration Rate/Acceleration Time Constant
    OL8038 = 100;           "Linear Deceleration Rate/Deceleration Time Constant
IEND;

RET;
```

Turn ON the Servo when MB300000 turns ON.

■ Sequence Subprogram (SPS003)

"SPS003: Jogging and Step Operation Processing"

```

"-----
"Jogging
"-----
IF IB80001 & ( (DB000010 & !DB000011) | (!DB000010 & DB000011) ) = = 1;
    DB000000 = 1;
ELSE;
    DB000000 = 0;
IEND;

DB000001 = PON( DB000000 DB000050 ) & MB300010; "Start jogging.
DB000002 = NON( DB000000 DB000051 ); "Stop jogging.

IF DB000001 = = 1;
    OL8010 = 1000;
    OW8008 = 7; "FEED motion command
IEND;
IF DB000002 = = 1;
    OW8008 = 0; "NOP motion command
IEND;

"-----
"Step operation
"-----
IF IB80001 & ( (DB000012 & !DB000013) | (!DB000012 & DB000013) ) = = 1;
    DB000008 = 1;
ELSE;
    DB000008 = 0;
IEND;

DB000009 = PON( DB000008 DB000058 ) & MB300010; "Start step operation.
DB00000A = NON( DB000008 DB000059 ); "Stop step operation.

IF DB000009 = = 1;
    OL8010 = 1000; "Set the STEP speed.
    OL8044 = 1000; "Set STEP Travel Distance (1,000 pulses).
    OW8008 = 8; "STEP motion command
IEND;

IF DB00000A = = 1;
    OW8008 = 0; "NOP motion command
IEND;

"-----
"Negative Rotation Selection
"-----
OB80092 = ( DB000000 & DB000011 ) | ( DB000008 & DB000013 ); "Select negative rotation.

RET;

```

Start jogging with positive rotation when DB000010 turns ON.

Start jogging with negative rotation when DB000011 turns ON.

Start step operation with positive rotation when DB000012 turns ON.


Start step operation with negative rotation when DB000013 turns ON.

Differences between MP2000-series and MP3000-series Machine Controllers

Appendix C

The differences between the MP2000-series and the MP3000-series Machine Controllers in terms of motion programs are listed in the following table.

■ Motion Programs

Item	MP2000-series Machine Controller	MP3000-series Machine Controller	Remarks
Number of Programs	256	512	This number includes both motion programs and sequence programs.
Number of Groups	8	16	–
Number of Tasks	16	32	This is the number of simultaneously executable programs.
Maximum Number of Controlled Axes per Group	16 axes	32 axes	The timing of transmitting references to slave stations via MECHATROLINK is different between the SVC/SVC32 and the SVB-01/SVC-01. Therefore, interpolation operations cannot be performed between the SVC/SVC32 and the SVB-01/SVC-01.
Number of forks	4 main program forks, 2 subprogram forks	Select from the following four options: <ul style="list-style-type: none"> • 4 main program forks, 2 subprogram forks • 8 main program forks • 2 main program forks, 4 subprogram forks • 8 subprogram forks 	 <i>Parallel Execution of Programs (page 1-7)</i>
G Registers	Not supported.	Supported.	–
Quadruple-length Integers	Not supported.	Supported.	This data type cannot be used for indirect designation in motion language instruction.
Double-length Real Numbers	Not supported.	Supported.	This data type cannot be used for indirect designation in motion language instruction.
Arrays	Not supported.	Supported.	–

■ Debug Operation Mode

Item	MP2000-series Machine Controller	MP3000-series Machine Controller	Remarks
Number of Breakpoints	4	8	–

■ Motion Program Operation When an Alarm Occurs for an Axis Specified in an Axis Movement Instruction

The MP3000-series Machine Controllers are different from the MP2000-series Machine Controllers in that they check for errors in all axes specified in axis movement instructions. If an alarm occurs, all of the specified axes are stopped and NOP motion commands are issued. Therefore, with the MP3000-series Machine Controllers, interlocks do not have to be created in the application for when alarms occur in specified axes, which improves safety in comparison the the MP2000-series Machine Controllers.

Information

The following table describes the motion program operation when an alarm occurs for an axis specified in an axis movement instruction. For the versions in the following table, you can select an MP2000-compatible mode for the motion program operation to use an MP3000-series Machine Controller in an application to replace an MP2000-series Machine Controllers without changing the interlocks.

Machine Controller or MPE720	Applicable Versions
MP3000-series Machine Controller	Version 1.08 or later
MPE720 Version 7	Version 7.21 or later

The following table describes the motion program operation when an alarm occurs for an axis specified in an axis movement instruction.

Axis Movement Instruction	MP2000-series Machine Controller			MP3000-series Machine Controller		
	Axis with Alarm	Axes without Alarms	Motion Program Operation	Axis with Alarm	Axes without Alarms	Motion Program Operation
Positioning (MOV) or Set-time Positioning (MVT)	Stop.	Move to target positions.	References continue to axes without alarms and they move to the target positions.	Stop.	Stop.	A motion program alarm occurs and references to all specified axes are stopped (alarm code: 8F hex (axis alarm)).
External Positioning (EXM)	Stop.	–	References continue until bit 8 (Command Execution Completed) in IW□□□09 (Motion Command Status) turns ON.	Stop.	–	
Zero Point Return (ZRN)	Stop.	Move to zero point.	References continue until bit 5 (Zero Point Return/Setting Completed) in IW□□□0C (Position Management Status) turns ON for all specified axes.	Stop.	Stop.	
Linear Interpolation (MVS),* Circular Interpolation/Helical Interpolation (MCW and MCC),* or Linear Interpolation with Skip Function (SKP)*	Stop.	Stop.	A motion program alarm (84 hex: Duplicated Motion Command) occurs and references to all specified axes are stopped.	Stop.	Stop.	
		Move to target positions.	References continue to axes without alarms and they move to the target positions.			

* The operation is different between the MP2000-series Machine Controllers and MP3000-series Machine Controllers when a software limit alarm occurs for an axis specified in an interpolation instruction. Refer to the following section for details.

■ Operation When a Software Limit Alarm Occurs for an Axis Specified in an Interpolation Instruction

Note: The motion program execution block does not change to the next block for the MP2000-series External Positioning (EXM) and Zero-point Return (ZRN) instructions. Therefore, you must execute a program reset or alarm reset request after a program stop request is executed.

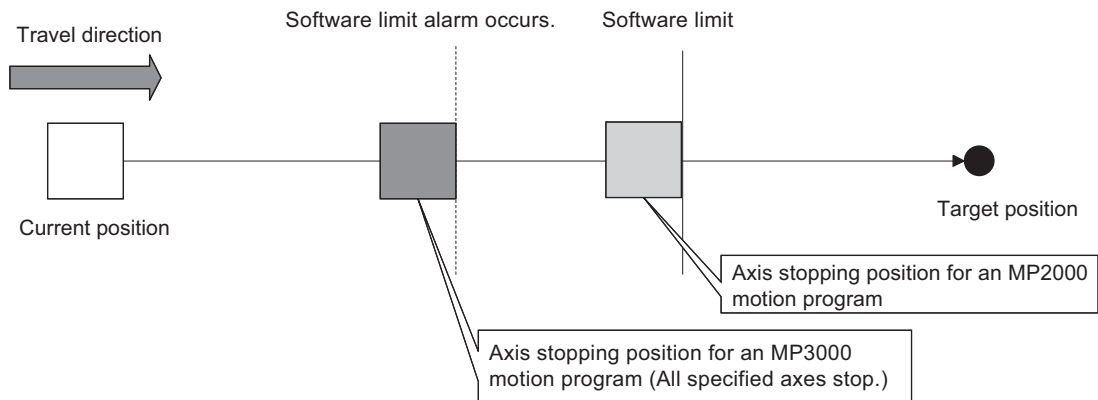
■ Operation When a Software Limit Alarm Occurs for an Axis Specified in an Interpolation Instruction

A software limit alarm occurs before the software limit so that the specified software limit is not exceeded.

For an MP3000-series Machine Controller, the axes stop when the axis alarm occurs, so all of the axes specified for the interpolation instructions stop when the axis with the alarm is before the software limit.

With an MP2000-series Machine Controller, the axis with the software limit alarm moves to the software limit and then stops. The axes without software limit alarms continue moving to their target positions.

Item	MP2000-series Machine Controller	MP3000-series Machine Controller
Axis with the Software Limit Alarm	The axis moves to the software limit.	The axis stops when the alarm occurs (it stops before the software limit).
Axes without a Software Limit Alarm	The axes move to their target positions.	The axes stop when the alarm occurs.
Motion Program Alarm Code	None	8F hex (axis alarm)



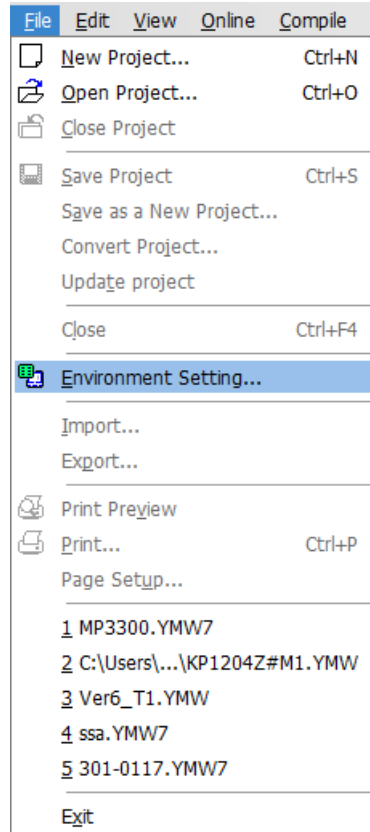
■ Procedure to Enable or Disable Axis Alarm Checks

This section describes how to enable and disable axis alarm checks.

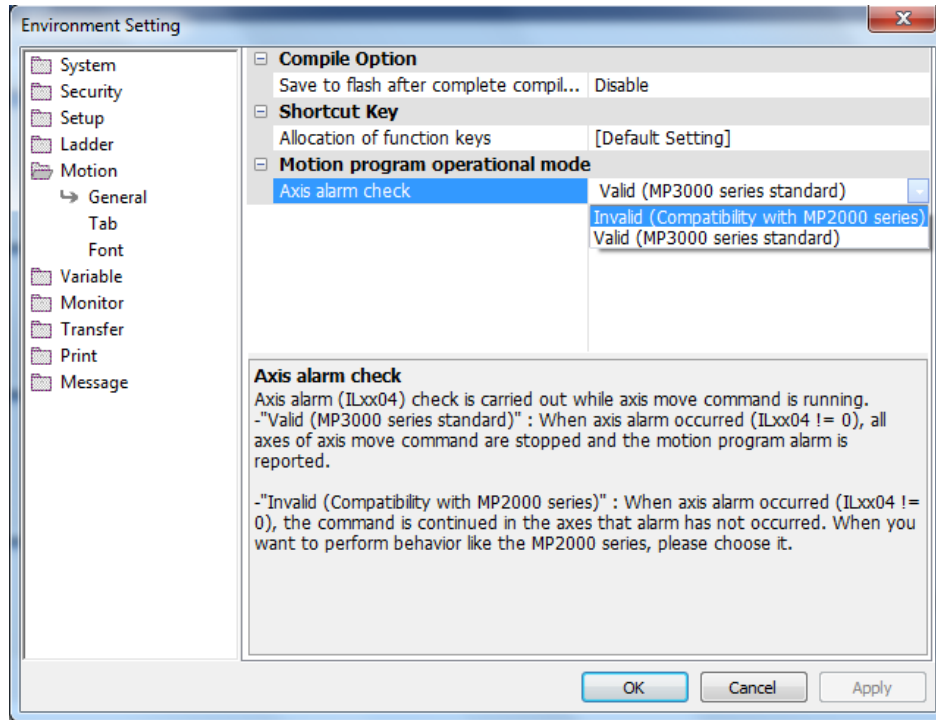
The setting for axis alarm checks is performed on the Environment Setting Dialog Box of MPE720 version 7.

Axis alarm checks are enabled by default.

1. Select **File – Environment Setting** from the menu bar.



2. Select the desired setting for **Motion Program Operation Mode** under **Motion** in the Environment Setting Dialog Box.



■ Timing at Which the Axis Alarm Check Setting Becomes Valid

The axis alarm setting becomes valid as soon as the **OK** Button is clicked in the Environment Setting Dialog Box.

Precautions

Appendix D

This appendix provides precautions for motion programs and sequence programs.

D.1	General Precautions	D-2
	Saving Data to Flash Memory when Changing Applications .	D-2
	Debugging a System in Operation	D-2
D.2	Precautions on Motion Parameters	D-3
	Performing Axis Movement Instructions on the Same Axis in Motion Programs	D-3
	Using a Subscript to Reference a Motion Register from an I/O Register	D-3
	Referencing the Motion Register of a Different Circuit	D-4
	OL□□□1C (Position Reference Setting) Setting Parameter .	D-5
	Axis Operation for Software Limit Alarms	D-5

D.1 General Precautions

This section provides general precautions for motion programs and sequence programs.

Saving Data to Flash Memory when Changing Applications

Always save the data to flash memory after you change motion programs, sequence programs, or other application data. If you do not, any changes that were made to the applications will be lost when the power supply to the Machine Controller is turned OFF.

Debugging a System in Operation

Never perform debugging on a system that is in operation. Debugging will cause changes in program operation, such as in instruction execution timing, resulting in malfunction or failure of the system. For debugging, use a special system for debugging.

D.2

Precautions on Motion Parameters

This section describes general precautions to consider when using motion parameters in a motion program.

Performing Axis Movement Instructions on the Same Axis in Motion Programs

If a movement instruction is executed by a motion program for an axis that is already in motion, the axis operation depends on the setting of bit 5 (Position Reference Type) in the OW□□□09 setting parameter.

The axis operation for each position reference type setting is described below.

Incremental Addition Method

This method adds the reference positions of both motion programs to perform positioning. The final position will be different from both original reference positions.

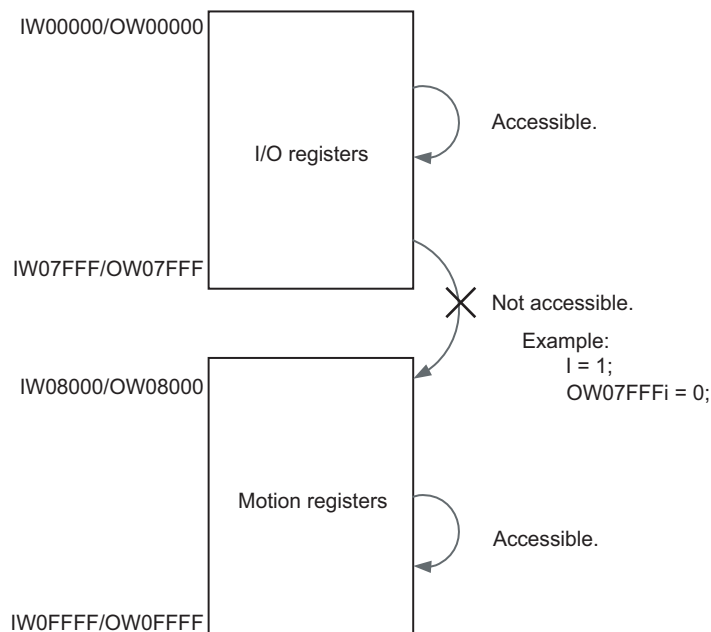
Absolute Value Set Method

This method performs positioning to the last-issued target position.

Using 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.



Referencing the Motion Register of a Different Circuit

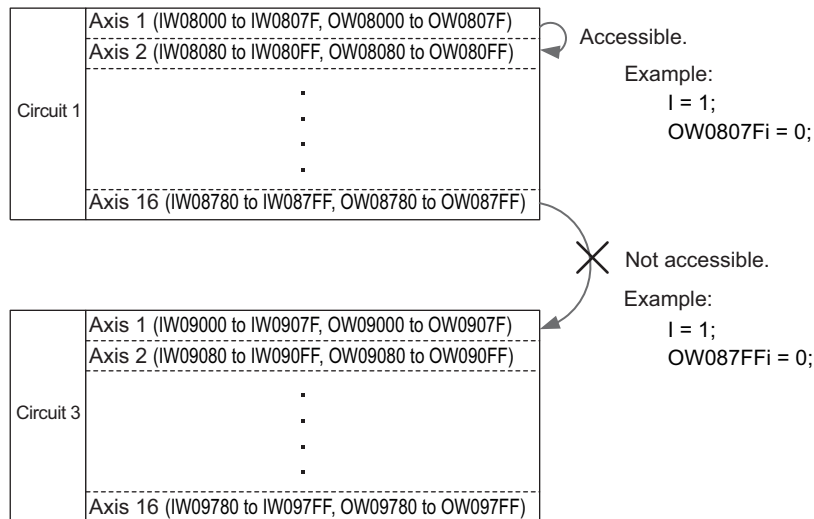
Motion registers for different circuits are not assigned to continuous memory location, just as is true for I/O registers and motion registers.

When using a subscript, access a register within the range of motion registers that is assigned to the circuit. If the circuit numbers are the same, it is even possible to access registers for different axes.

The following table lists the motion registers.

Circuit Number	Axis 1	Axis 2	...	Axis 16
1	OW08000 to OW0807F	OW08080 to OW080FF	...	OW08780 to OW087FF
3	OW09000 to OW0907F	OW09080 to OW090FF	...	OW09780 to OW097FF
5	OW0A000 to OW0A07F	OW0A080 to OW0A0FF	...	OW0A780 to OW0A7FF
7	OW0B000 to OW0B07F	OW0B080 to OW0B0FF	...	OW0B780 to OW0B7FF
9	OW0C000 to OW0C07F	OW0C080 to OW0C0FF	...	OW0C780 to OW0C7FF
11	OW0D000 to OW0D07F	OW0D080 to OW0D0FF	...	OW0D780 to OW0D7FF
13	OW0E000 to OW0E07F	OW0E080 to OW0E0FF	...	OW0E780 to OW0E7FF
15	OW0F000 to OW0F07F	OW0F080 to OW0F0FF	...	OW0F780 to OW0F7FF

Example Accessing the Motion Register of a Different Circuit



OL□□□1C (Position Reference Setting) Setting Parameter

If the OL□□□1C (Position Reference Setting) setting parameter is changed in a program (e.g., a ladder program) while axis motion is in progress for another motion program, the axes will move with the new value of the parameter. This will result in a difference between the actual axis position and the position specified in the motion program.

Example If the travel distance of axis A1 that is specified in OL□□□1C (Position Reference Setting) is changed from 1,000 to 1,500 from a ladder program while execution of ① in the following motion program is in progress, the axis will move to 1,500. This results in a difference between the actual axis position and the reference position (1,000) that is specified in the motion program. Line ② in the motion program is then executed. As a result, the actual final position of axis A1 will be at a different position from that specified in the motion program.

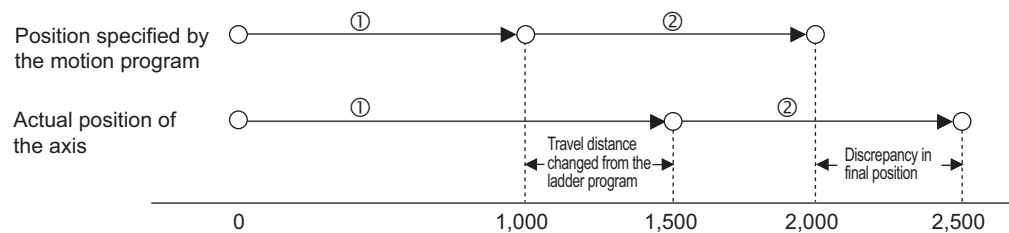
```

INC;
ZRN [A1]0;

MOV [A1]1000; . . . ①
MOV [A1]1000; . . . ②

END;

```



Axis Operation for Software Limit Alarms

When a software limit alarm (IL□□□04 bits 3 and 4) occurs during execution of an interpolation instruction, the axis may stop before the software limit depending on the speed setting. The stopping position depends on the speed setting.

Index

Symbols

-----	6-171
-----	6-174
^ -----	6-182
! -----	6-183
* -----	6-176
/ -----	6-177
&-----	6-181
# registers -----	4-4
+ -----	6-170
++ -----	6-172
< -----	6-186
<=-----	6-186
<> -----	6-186
= -----	6-169
== -----	6-186
> -----	6-186
>=-----	6-186
-----	6-180

Numerics

1-ms OFF-Delay Timer (TOF1MS) -----	6-219
1-ms ON-Delay Timer (TON1MS)-----	6-217

A

ABS -----	6-7
Absolute Mode -----	6-7
Absolute Mode (ABS) -----	6-7
ACC -----	6-15
acceleration times and deceleration times -----	6-34
Acceleration/Deceleration Mode with Continuous Process Control Signal Monitoring-----	6-65
Acceleration/Deceleration Mode with Interpolation Overlapping -----	6-69
acceleration/deceleration type -----	6-82, 6-88
ACCMODE -----	6-63
ACS -----	6-204
active program numbers -----	1-34
Add (+) -----	6-170
Add Quick Watch -----	7-17
Alarm Monitor -----	7-24
Alarm Reset Request -----	1-26
Allocation Register-----	7-10
AND (&) -----	6-181
Arc Cosine (ACS)-----	6-204
Arc Sine (ASIN)-----	6-203
Arc Tangent (ATN)-----	6-205
array registers-----	4-20
ASCII -----	6-196
ASCII Conversion 1 (ASCII)-----	6-196

ASCII text -----	6-196
ASN -----	6-203
assigned interlock contact -----	1-32
assigned registers -----	1-32
ATN -----	6-205
axis control instructions -----	6-115
Axis Monitor -----	1-13
axis movement instructions -----	6-77
axis number -----	5-10
axis setting instructions -----	6-4

B

basic functions -----	6-198
batch transfer -----	3-14
BCD -----	6-209
BCD data -----	6-208, 6-209
BCD to Binary (BIN) -----	6-208
BIN -----	6-208
binary data -----	6-208, 6-209
Binary to BCD (BCD) -----	6-209
Bit Shift Left (SFL) -----	6-191
Bit Shift Right (SFR) -----	6-189
BLK -----	6-192
block -----	5-2
Branching Instructions (IF, ELSE, and IEND)-----	6-131
Break-----	7-15
Breakpoint Enable/Disable -----	7-17
Breakpoint Set/Remove -----	7-14

C

C registers -----	4-3, 4-16
Call Motion Subprogram (MSEE) -----	6-148
Call Sequence Subprogram (SSEE) -----	6-149
Call User Function -----	6-150, 6-158
Call User Function from Motion Program (UFC) -----	6-150
Call User Function from Sequence Program (FUNC) -----	6-158
calling motion programs using the M-EXECUTOR program execution definitions-----	1-21
center point position -----	6-91
Change Acceleration Time (ACC) -----	6-15
Change Deceleration Time (DCC) -----	6-21
Change Interpolation Acceleration Time (IAC)-----	6-50
Change Interpolation Deceleration Time (IDC)-----	6-52
Change Interpolation Deceleration Time for Temporary Stop (IDH) -----	6-54
Change S-curve Time Constant (SCC) -----	6-27
character	
D-----	5-6
F-----	5-5
FW -----	5-5
M -----	5-5
MPS -----	5-6
N-----	5-6
P-----	5-5

R ----- 5-5
 SPS----- 5-6
 SS----- 5-6
 T----- 5-5
 TW----- 5-5
 U ----- 5-5
 V ----- 5-5
 W ----- 5-6
 circuit ----- 5-9
 circuit number ----- 5-9
 Circular Interpolation with Specified Center Point (MCW and MCC) ----- 6-90
 Circular Interpolation with Specified Radius (MCW and MCC) ----- 6-95
 Clear (CLR)- ----- 6-193
 Comment Check Box ----- 7-7
 comments ----- 5-2, 5-7
 compiling ----- 3-9
 composite travel distance ----- 6-86
 constant registers ----- 4-3
 control signals ----- 1-26
 conveyance device ----- 1-43
 Coordinate Plane Setting (PLN) ----- 6-128
 coordinate words ----- 5-2, 5-4
 COS ----- 6-201
 Cosine (COS) ----- 6-201
 creating a project ----- 3-4
 creating programs----- 3-8
 cross references ----- 7-27
 Current Position Set (POS) ----- 6-117
 current program line ----- 7-12

D

D registers ----- 4-3, 4-17
 data manipulations ----- 6-189
 data registers ----- 4-2, 4-12
 data types ----- 4-8, 6-151
 data types of registers used in user functions ----- 6-151
 DCC ----- 6-21
 Debug Operation Mode ----- 1-24, 7-11, 7-13
 Debugging----- 1-13, 2-4
 debugging programs ----- 3-16
 decimal integer ----- 5-8
 DEFAULT----- 6-143
 DEN ----- 6-107
 direct designation----- 2-8
 Disable Single-block Signal (SNGD) and Enable Single-block Signal (SNGE)- ----- 6-167
 Divide (/)- ----- 6-177
 Drive Control Panel ----- 7-18
 Dwell Time (TIM) ----- 6-161

E

easy programming functions ----- 1-13, 2-4
 electronic gear ----- 6-37
 encoder cable----- 3-3
 END ----- 6-159
 End ----- 7-15
 end of block----- 5-2, 5-6
 end position----- 6-91
 EOX ----- 6-166
 Error List Dialog Box----- 3-9
 Exclusive OR (^) ----- 6-182
 Execute ----- 7-15
 executing programs ----- 3-18
 Execution Scan Error ----- 1-24
 execution scans ----- 5-13
 EXM----- 6-113
 Extended Add (++) ----- 6-172
 Extended Subtract (--) ----- 6-174
 External Positioning (EXM) ----- 6-113
 external positioning signal ----- 6-114

F

F reference ----- 6-47, 6-87
 Falling-edge Pulse (NON) ----- 6-214
 Filter Time Constant----- 6-28
 filter type selection----- 6-32
 finite-length axis ----- 6-9
 FMX----- 6-39
 FUNC----- 6-158
 function keys ----- 7-12
 Function Selection Flags----- 6-9, 6-13

G

G registers ----- 4-2, 4-13
 global registers----- 4-5, 5-8
 group definition ----- 5-9
 group definition settings ----- 3-6
 group name ----- 5-9

H

Helical Interpolation with Specified Center Point (MCW and MCC) ----- 6-99
 Helical Interpolation with Specified Radius (MCW and MCC) ----- 6-102
 Help Button ----- 7-8
 hexadecimal integer ----- 5-8
 high-speed drawing ----- 1-15
 how to directly change the acceleration time settings ----- 6-20
 how to directly change the deceleration time settings ----- 6-26

I

I registers ----- 4-2, 4-14
 I/O services ----- 1-17
 I/O Variable Wait (IOW)- ----- 6-163
 IAC----- 6-50

- IDC ----- 6-52
 IDH ----- 6-54
 IF, ELSE, and IEND ----- 6-131
 IFMX ----- 6-42
 IFP ----- 6-47
 INC ----- 6-11
 Inclusive OR (I) ----- 6-180
 Incremental Mode (INC) ----- 6-7, 6-11
 index i ----- 4-18
 index j ----- 4-18
 indirect designation ----- 1-31
 indirect designation of a program number using a register ----- 1-31
 infinite-length axis ----- 6-9
 Infinite-length Axis Reset Position ----- 6-9, 6-13
 INP ----- 6-124
 In-position Check (PFN) ----- 6-122
 in-position range ----- 6-124
 In-Position Range (INP) ----- 6-124
 input registers ----- 4-2, 4-14
 installing MPE720 version 7 ----- 3-3
 Instruction Entry Assistance ----- 1-13, 2-4
 instruction format ----- 7-6
 instruction type table ----- 5-15
 instruction types ----- 5-13
 interpolation acceleration time ----- 6-51
 interpolation deceleration time ----- 6-53
 interpolation feed speed ----- 6-47, 6-87
 interpolation feed speed ratio ----- 6-48
 interpolation instructions ----- 6-31, 6-47
 interpolation override ----- 1-29
 Interpolation Override Setting ----- 1-27
 IOW ----- 6-163
- J**
- jogging ----- 7-22
 JOINTO ----- 6-140, 6-143
- L**
- labels ----- 5-2, 5-3
 ladder programs ----- 1-4
 language instructions ----- 2-3
 linear acceleration rate ----- 6-17
 linear deceleration rate ----- 6-23
 linear deceleration time constant ----- 6-22
 Linear Interpolation (MVS) ----- 6-85
 Linear Interpolation with Skip Function (SKP) ----- 6-109
 local registers ----- 4-6, 5-8
 logic operation instructions ----- 6-179
 logical axis name ----- 5-10
 logical axis names ----- 5-2, 5-3
- M**
- M registers ----- 4-2, 4-12
 Machine Controller specifications ----- A-3
 machine coordinate system ----- 6-117
 Main Program Number Limit Exceeded Error ----- 1-25
 main programs ----- 1-15, 2-5
 maximum interpolation feed speed ----- 6-40
 MCC ----- 6-90, 6-95, 6-99, 6-102
 MCW ----- 6-90, 6-95, 6-99, 6-102
 metal sheet pressing equipment ----- 1-44
 M-EXECUTOR control registers ----- 1-21
 M-EXECUTOR program execution definitions ----- 1-21
 MOD ----- 6-178
 Modulo (MOD) ----- 6-178
 Monitor Parameter Display ----- 7-25
 Monitor Parameter Selection Area ----- 7-25
 monitor parameters ----- 4-14
 monitoring motion program execution information using
 the S registers ----- 1-33
 Motion Control Function Module ----- 1-14
 Motion Editor ----- 1-13, 1-14, 3-8
 motion language ----- 1-3
 motion language instructions ----- 5-2, 6-1
 motion parameters ----- 1-14
 motion program execution timing figure ----- 1-17
 motion program numbers ----- 1-15
 motion programs ----- 1-3, 1-4
 application examples ----- 1-43
 data transfer to and from ladder programs ----- 1-6
 execution information ----- 1-34
 execution methods ----- 1-4
 execution processing methods ----- 1-19
 execution registration ----- 1-22
 execution timing ----- 1-17
 format ----- 5-2
 groups ----- 1-16
 motion control ----- 1-5
 online editing ----- 1-12
 parallel execution ----- 1-7
 system configuration ----- 1-14
 types ----- 1-15
 use of subprograms ----- 1-6
 motor cable ----- 3-3
 MOV ----- 6-81
 Move Block (BLK) ----- 6-192
 Move on Machine Coordinates (MVM) ----- 6-119
 Move Start Point ----- 7-13
 movement paths for interpolation instructions and
 S-curve acceleration/deceleration ----- 6-31
 Moving Average Filter ----- 6-32
 MPE720 version 7.0 ----- 1-13
 MSEE ----- 6-148
 M-type instructions ----- 5-14
 Multiply (*) ----- 6-176
 MVM ----- 6-119
 MVS ----- 6-85
 MVT ----- 6-111

N

NEAR Signal Output Width	6-122, 6-124
No System Work Available	1-24
NON	6-214
Normal Operation Mode	7-13
NOT (!)	6-183
notation for constants	5-8
number of controlled axes	5-9, 7-6
number of groups	5-9
number of turns	6-93
numeric comparison instructions	6-186
numeric operation instructions	6-168

O

O registers	4-3, 4-15
Off-delay Timer	
Measurement unit = 10 ms (TOF)	6-218
On-delay Timer	
Measurement unit = 10 ms (TON)	6-216
One Scan Wait (EOX)	6-166
online editing	1-12
Operation Control Panel	1-13
operation priority levels	5-11
operation with multiple groups	1-16
operation with one group	1-16
output registers	4-3, 4-15
overrides	6-36

P

panel processing machine	1-44
Parallel Execution Instructions (PFORK, JOINTO, and PJOINT)	6-140
parameter settings	7-7
part inserter	1-43
PFN	6-122
PFORK	6-140
PFP	6-126
PJOINT	6-140
PLD	6-120
PLN	6-128
PON	6-212
POS	6-117
Position after Distribution (DEN)	6-107
position reference value	6-7
Positioning (MOV)	6-81
Positioning Completed Check (PFP)	6-126
positioning instructions	6-33
positioning speed	6-16, 6-17, 6-22
POSMAX	6-9, 6-13
PP cable	3-3
precautions to consider when performing register operations	4-10
procedure to create the user function	6-155
Program Alarms	1-24
program control instructions	6-129

program current position	6-7
program development flow	3-2
Program End (END)	6-159
Program Executing	1-24
Program Paused	1-24
program properties	4-17
Program Single-block Execution Stopped	1-24
Program Single-block Mode Selection	1-26
Program Single-block Start Request	1-26
program status	1-35
Program Stopped at Breakpoint	1-24
Program Stopped for Request for Stop Request	1-24
Program Type	1-24
programming with variables	5-17

R

R{ }	6-211
radius	6-96
rated speed	6-16, 6-22
real number	5-8
reference position	6-7
Reference Unit Selection	6-34
register list	3-2
registering motion programs in the M-EXECUTOR program execution definitions	1-22
registering program execution	3-10
relationship between I/O registers and internal function registers	6-152
relative travel distances	6-11
Repetition Instructions (WHILE, WEND)	6-134
Repetition with One Scan Wait (WHILE and WENDX)	6-137
Request for Pause of Programs	1-26
Request for Start of Continuous Program Operation	1-26
Request for Start of Program Operation	1-26
Request for Stop of Program	1-26
Reset Bit (R{ })	6-211
RET	6-160
Rising-edge Pulse (PON)	6-212

S

S registers	4-2, 4-11
S{ }	6-210
sample programs	
motion program control program	B-2
parallel processing	B-3
performing speed control with a motion program	B-4
sequence programs	B-7
simple synchronized operation with a virtual axis	B-5
saving programs to flash memory	3-17
scan execution	1-4, 2-2, 2-3
SCC	6-27
S-curve acceleration/deceleration	6-83
S-curve time constant	6-27, 6-30
Select Command	7-6

- Selective Execution Instructions
(SFORK, JOINTO, SJOINT)----- 6-143
- self configuration ----- 3-6
- sequence programs
- execution ----- 2-6
 - execution methods ----- 2-3
 - execution processing method----- 2-6
 - execution timing----- 2-7
 - features ----- 2-3
 - M-EXECUTOR program execution definitions----- 2-7
 - registering program execution ----- 2-8
 - types ----- 2-5
 - use of subprograms----- 2-4
- sequential execution ----- 1-4
- SERVOPACK ----- 3-3
- Set Bit (S{ }) ----- 6-210
- Set Call Stack----- 7-16
- Set Interpolation Acceleration/Deceleration Mode
(ACCMODE)----- 6-63
- Set Interpolation Feed Speed Axes (+ and -)----- 6-60
- Set Interpolation Feed Speed Ratio (IFP) ----- 6-47
- Set Maximum Individual Axis Speeds for Interpolation
(IFMX) ----- 6-42
- Set Maximum Interpolation Feed Speed (FMX) ----- 6-39
- Set Motion Task ----- 7-16
- Set Speed (VEL)----- 6-33
- Set-time Positioning (MVT) ----- 6-111
- setting parameters----- 4-15
- setting up the system----- 3-6
- SETW ----- 6-194
- SFL ----- 6-191
- SFORK ----- 6-143
- SFR ----- 6-189
- SIN ----- 6-200
- Sine (SIN) ----- 6-200
- single-block operation mode ----- 6-167
- single-step linear acceleration/deceleration----- 6-82
- SJOINT ----- 6-143
- Skip 1 Information ----- 1-26
- Skip 2 Information ----- 1-26
- Skip Input Signal 1 (SS1) ----- 6-109
- Skip Input Signal 2 (SS2) ----- 6-109
- Skip Input Signal Selection ----- 6-109
- SKP----- 6-109
- SNGD ----- 6-167
- SNGE ----- 6-167
- software limit switches ----- 6-118
- specific characters ----- 5-2, 5-5
- specified center point ----- 6-99
- specified radius ----- 6-95, 6-102
- speed reference----- 7-22
- speed unit----- 6-34
- SQT----- 6-206
- Square Root ----- 6-198
- Square Root (SQT)----- 6-206
- SSEE----- 6-149
- Start Request History ----- 1-24
- Status Display ----- 7-24
- Status Flags ----- 1-24, 2-9
- step distance----- 7-22
- step execution ----- 7-22
- Step In ----- 7-14
- Step In execution ----- 3-16
- Step Over----- 7-14
- Stop/Start Monitor ----- 7-24, 7-26
- S-type instructions ----- 5-14
- Subprogram Return (RET) ----- 6-160
- subprograms----- 1-6, 1-15, 2-4, 2-5
- Substitute (=)----- 6-169
- Subtract (-)----- 6-171
- SVA-01 ----- 1-14
- SVB-01 ----- 1-14
- SVR ----- 1-14
- syntax error ----- 4-4
- system registers ----- 4-2, 4-11
- system work number----- 1-29
- System Work Number Setting ----- 1-27
- ## T
- Table Initialization (SETW)----- 6-194
- TAN ----- 6-202
- Tangent (TAN) ----- 6-202
- task assignments----- 7-9
- Test Runs ----- 1-13
- TIM----- 6-161
- TIM1MS ----- 6-162
- TOF----- 6-218
- TOF1MS ----- 6-219
- TON ----- 6-216
- TON1MS ----- 6-217
- toolbar icons ----- 7-12
- transferring programs ----- 3-13
- T-type instructions ----- 5-14
- types of registers ----- 4-2
- typical system configuration ----- 3-3
- ## U
- UFC ----- 6-150
- Update Current Position ----- 7-15
- Update Program Current Position (PLD) ----- 6-120
- user functions----- 6-151
- using registers ----- 4-11
- ## V
- VEL ----- 6-33
- virtual axes ----- 1-43

W

Warning Display	7-26
WHILE and WENDX.....	6-137
WHILE, WEND	6-134
work registers	1-23, 2-9
working coordinate system	6-104, 6-117

Z

Zero Point Return (ZRN)	6-104
Zero Point Return Method	6-105
zero point return speed	6-105
ZRN	6-104

Revision History

The revision dates and numbers of the revised manuals are given on the bottom of the back cover.

MANUAL NO. SIEP C880725 14A <0>-1
 Published in Japan July 2012

┌─── WEB revision number
 │
 └─── Revision number
 └─── Date of publication

Date of Publication	Rev. No.	WEB Rev. No.	Section	Revised Contents
March 2018	<3>	0	1.7	Addition: Supplemental information for timing chart
			5.5	Addition: Strings that Cannot Be Used in Variable Names
			6.4	Addition: Using parallel execution instructions with subprograms
July 2016	<2>	2	4.3	Addition: Setting range for indices i and j
			6.4	Revision: Radius of a circle drawn using the WHILE and WEND instruction
August 2015		1	Front cover	Revision: Format
			1.8	Revision: SW08192 to SW09215 → SL08192 to SL09214 SW03264 to SW03321 → SW03264 to SW03321 and SL08192 to SL08222 SW03380 to SW03437 → SW03380 to SW03437 and SL08256 to SL08286
			6.2	Revision: Interpolation feed speed figure (INC MVS[A1]1200 [B1]900 F500;)
			Appendix C	Revision: Information on motion program operation for MP2000-series Machine Controllers.
			Back cover	Revision: Address and format
June 2014		0	–	Based on Japanese user's manual, SIJP C880725 14D<3>-0, available on the Web in March 2014
			All chapters	Addition: Description of MP3300
			5.4	Addition: Description of FUT and IUT instructions
			6.1	Addition: Description of FUT, IUT, and ACCMODE instructions
September 2012	<1>	0	–	Based on Japanese user's manual, SIJP C880725 14B<1>-1, available on the Web in July 2012
			Back cover	Revision: Address
July 2012	<0>	1	6.2	Revision: Example of setting the interpolation feed speed in the MVS instruction
			Back cover	Revision: Address
March 2012	–	–	–	First edition

Machine Controller MP3000 Series

Motion Program

PROGRAMMING MANUAL

IRUMA BUSINESS CENTER (SOLUTION CENTER)

480, Kamifujisawa, Iruma, Saitama, 358-8555, Japan
Phone: +81-4-2962-5151 Fax: +81-4-2962-6138
<http://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
<http://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
<http://www.yaskawa.com.br>

YASKAWA EUROPE GmbH

Hauptstraße 185, 65760 Eschborn, Germany
Phone: +49-6196-569-300 Fax: +49-6196-569-398
<http://www.yaskawa.eu.com> E-mail: info@yaskawa.eu.com

YASKAWA ELECTRIC KOREA CORPORATION

35F, Three IFC, 10 Gukjegeumyung-ro, Yeongdeungpo-gu, Seoul, 07326, Korea
Phone: +82-2-784-7844 Fax: +82-2-784-8495
<http://www.yaskawa.co.kr>

YASKAWA ELECTRIC (SINGAPORE) PTE. LTD.

151, Lorong Chuan, #04-02A, New Tech Park, 556741, Singapore
Phone: +65-6282-3003 Fax: +65-6289-3003
<http://www.yaskawa.com.sg>

YASKAWA ELECTRIC (THAILAND) CO., LTD.

59, 1st-5th Floor, Flourish Building, Soi Ratchadapisek 18, Ratchadapisek Road, Huaykwang, Bangkok, 10310, Thailand
Phone: +66-2-017-0099 Fax: +66-2-017-0799
<http://www.yaskawa.co.th>

YASKAWA ELECTRIC (CHINA) CO., LTD.

22F, One Corporate Avenue, No.222, Hubin Road, Shanghai, 200021, China
Phone: +86-21-5385-2200 Fax: +86-21-5385-3299
<http://www.yaskawa.com.cn>

YASKAWA ELECTRIC (CHINA) CO., LTD. BEIJING OFFICE

Room 1011, Tower W3 Oriental Plaza, No.1, East Chang An Ave.,
Dong Cheng District, Beijing, 100738, China
Phone: +86-10-8518-4086 Fax: +86-10-8518-4082

YASKAWA ELECTRIC TAIWAN CORPORATION

12F, No. 207, Sec. 3, Beishin Rd., Shindian Dist., New Taipei City 23143, Taiwan
Phone: +886-2-8913-1333 Fax: +886-2-8913-1513 or +886-2-8913-1519
<http://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.

© 2012 YASKAWA ELECTRIC CORPORATION

MANUAL NO. SIEP C880725 14D <3>-0

Published in Japan March 2018

17-4-13