C200HW-MC402-E
Motion Control Unit

Operation Manual

Produced April 2001

Notice:

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

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

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

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

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

OMRON Product References

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

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

The abbreviation “PC” means Programmable Controller and is not used as an abbreviation for anything else.

Visual Aids

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

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

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

© OMRON, 2001

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 OMRON.

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

TABLE OF CONTENTS

PRECAUTIONS xi

1 Intended Audience xii
2 General Precautions xii
3 Safety Precautions xii
4 Operating Environment Precautions xiii
5 Application Precautions xiii
6 Conformance to EC Directives xv

SECTION 1
Features and System Configuration 1

1-1 Features 2
1-2 System Configuration 4
1-3 Motion Control Concepts 5
1-4 Control System 14
1-5 Specifications 18
1-6 Comparison with C200HW-MC402-UK 20

SECTION 2
Installation 23

2-1 Components and Unit Settings 24
2-2 Installation 25
2-3 Wiring 26
2-4 Servo System Precautions 36
2-5 Wiring Precautions 38

SECTION 3
PC Data Exchange 41

3-1 IR/CIO Area Allocation 42
3-2 Overview of Data Exchanges 46
3-3 Details of the Data Exchange Methods 48

SECTION 4
Multitasking BASIC Programming 57

4-1 Overview 58
4-2 BASIC Programming 58
4-3 Motion Control Application 61
4-4 Command Line Interface 65
4-5 BASIC Programs 65
4-6 Error Processing 68

TABLE OF CONTENTS

SECTION 5
BASIC Motion Control Programming Language 71

5-1 Notation Used in this Section 75
5-2 Classifications and Outlines 75
5-3 Command, function and parameter description 84

SECTION 6
Programming Environment 157

6-1 Motion Perfect Features 158
6-2 Motion Perfect Requirements 158
6-3 Going Online with the MC Unit 158
6-4 Motion Perfect Projects 159
6-5 Motion Perfect Desktop 161
6-6 Motion Perfect Tools 164
6-7 Suggestions and Precautions in Using Motion Perfect 177

SECTION 7
Troubleshooting 179

7-1 Problems and Countermeasures 180
7-2 Error Indicators 184
7-3 Error Handling 184
7-4 CPU Unit Error Flags and Control Bits 187

SECTION 8
Maintenance and Inspection 189

8-1 Routine Inspections 190
8-2 Handling Precautions 191

Appendices

Appendix A PC Interface Area Lists 193
Appendix B Programming Examples 195

Index 203

Revision History 209

About this Manual:

This manual describes the installation and operation of the C200HW-MC402-E Motion Control Unit (MC Unit) and includes the sections described below.

Please read this manual carefully and be sure you understand the information provided before attempting to install or operate the MC Unit. Be sure to read the precautions provided in the following section.

Precautions provides general precautions for using the MC Unit, Programmable Controller (PC), and related devices.

Section 1 describes the function of the C200HW-MC402-E Motion Control Unit and concepts related to its operation. Also the specifications and the comparison with previous C200HW-MC402-UK is shown.

Section 2 describes information required for hardware setup and installation.

Section 3 describes the IR/CIO area allocation and presents the different methods of data exchange between the MC Unit and the CPU Unit.

Section 4 gives an overview of the fundamentals of multitasking BASIC programs and the methods by which programs are managed for the MC Unit.

Section 5 describes the commands and parameters required for programing the motion control application using the MC Unit. All BASIC system, task and axis statements that determine the various aspects of program execution and MC Unit operation are presented.

Section 6 provides an overview of software package Motion Perfect, which is used to program, monitor and debug motion based applications for the MC Unit.

Section 7 provides procedures on troubleshooting problems that may arise with the MC Unit.

Section 8 explains the maintenance and inspection procedures that must be followed to keep the MC Unit operating in optimum condition. It also includes proper procedures when replacing an MC Unit or battery.

The ***Appendices*** provide the PC Interface Lists and gives convenient programming examples for the user.

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

PRECAUTIONS

This section provides general precautions for using the Motion Control Unit and related devices.

The information contained in this section is important for the safe and reliable application of the Motion Control Unit. You must read this section and understand the information contained before attempting to set up or operate a Motion Control Unit and PC system.

1 Intended Audience xii
2 General Precautions xii
3 Safety Precautions xii
4 Operating Environment Precautions xiii
5 Application Precautions xiii
6 Conformance to EC Directives xv
6-1-1 Concepts xv
6-1-2 Conformance to EC Directives xvi

1 Intended Audience

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

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

2 General Precautions

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

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

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

This manual provides information for installing and operating OMRON Motion Control Units. Be sure to read this manual before operation and keep this manual close at hand for reference during operation.

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

3 Safety Precautions

⚠WARNING Never attempt to disassemble any Units while power is being supplied. Doing so may result in serious electrical shock or electrocution.

⚠WARNING Never touch any of the terminals while power is being supplied. Doing so may result in serious electrical shock or electrocution.

⚠WARNING Provide safety measures in external circuits (i.e., not in the Programmable Controller or MC Unit) to ensure safety in the system if an abnormality occurs due to malfunction of the PC, malfunction of the MC Unit, or external factors affecting the operation of the PC or MC Unit. Not providing sufficient safety measures may result in serious accidents.

- Emergency stop circuits, interlock circuits, limit circuits, and similar safety measures must be provided in external control circuits.
- The PC or MC Unit outputs may remain ON or OFF due to deposits on or burning of the output relays, or destruction of the output transistors. As a counter-measure for such problems, external safety measures must be provided to ensure safety in the system.
- When the 24-VDC output (service power supply to the PC) is overloaded or short-circuited, the voltage may drop and result in the outputs being turned OFF. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.

- It is the nature of high speed motion control and motion control language programming and multi-tasking systems, that it is not always possible for the system to validate the inputs to the functions. It is the responsibility of the programmer to ensure that various BASIC statements are called correctly with the correct number of inputs and that the values are correctly validated prior to the actual calling of the various functions.

⚠ **Caution** Connect the ENABLE output (drivers enable signal) to the Servo Drivers. Otherwise, the motor may run when the power is turned ON or OFF or when an error occurs in the Unit.

⚠ **Caution** Do not save data into the flash memory during memory operation or while the motor is running. Otherwise, unexpected operation may be caused.

⚠ **Caution** Do not reverse the polarity of the 24-V power supply. The polarity must be correct. Otherwise, the motor may start running unexpectedly and may not stop.

4 Operating Environment Precautions

⚠ **Caution** Do not operate the control system in the following locations:

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidity outside the range specified in the specifications.
- Locations subject to condensation due to radical temperature changes.
- Locations subject to corrosive or inflammable gases.
- Locations subject to dust (especially iron dust) or salts.
- Locations subject to vibration or shock.
- Locations subject to exposure to water, oil or chemicals.

⚠ **Caution** Take appropriate and sufficient countermeasures when installing systems in the following locations:

- Locations subject to static electricity or other sources of noise.
- Locations subject to strong electromagnetic fields.
- Locations subject to possible exposure to radiation.
- Locations near power supply lines.

⚠ **Caution** The operating environment of the PC System can have a large effect on the longevity and reliability of the system. Improper operating environments can lead to malfunction, failure, and other unforeseeable problems with the PC System. Be sure that the operating environment is within the specified conditions at installation and remains within the specified conditions during the life of the system.

5 Application Precautions

Observe the following precautions when using the Motion Control Unit or the PC System.

⚠ **WARNING** Failure to abide by the following precautions could lead to serious or possibly fatal injury. Always heed these precautions.

- Always ground the system to 100 Ω or less when installing the system to protect against electrical shock.

- Always turn OFF the power supply to the PC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting the MC Unit or any other Units.
  - Assembling the Units.
  - Setting rotary switches.
  - Connecting cables or wiring the system.
  - Connecting or disconnecting the connectors.

⚠ **Caution** Failure to abide by the following precautions could lead to faulty operation of the PC, the MC Unit or the system, or could damage the PC or MC Unit. Always heed these precautions.

- Maximum 12 of the digital inputs (I0 to I15) should be switched on at any one time to ensure that the Unit remains within internal temperature specifications. Failure to meet this condition may lead to degradation of performance or damage of components.
- After development of the application programs, be sure to save the program data in flash memory within the MC Unit (using the EPROM command in BASIC). The program data will remain in the S-RAM during operation and power down, but considering possible battery failure it is advised to store the data in flash memory.
- It is strongly recommended to store dynamic application data, which can not be initiated in program, in the PC Unit's memory considering possible battery failure.
- Do not turn OFF the power supply to the Unit while data is being written to flash memory. Doing so may cause problems with the flash memory.
- Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in unexpected operation.
  - Changing the operating mode of the PC.
  - Changing the present value of any word or any set value in memory.
  - Force-setting/force-resetting any bit in memory
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.
- Be sure that all mounting screws, terminal screws, and cable connector screws are tightened securely. Incorrect tightening may result in malfunction.
- Before touching the Unit, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.
- Check the pin numbers before wiring the connectors.
- Be sure that the connectors, terminal blocks, I/O cables, cables between drivers, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Always use the power supply voltages specified in this manual. An incorrect voltage may result in malfunction or burning.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied. Be particularly careful in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals. Connection of bare stranded wires may result in burning.

- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction if foreign matter enters the Unit.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Do not apply voltages to the Input Units in excess of the rated input voltage. Excess voltages may result in burning.
- Do not apply voltages or connect loads to the Output Units in excess of the maximum switching capacity. Excess voltage or loads may result in burning.
- Disconnect the functional ground terminal when performing withstand voltage tests. Not disconnecting the functional ground terminal may result in burning.
- Double-check all wiring and switch settings before turning ON the power supply. Incorrect wiring may result in burning.
- Do not pull on the cables or bend the cables beyond their natural limit. Doing either of these may break the cables.
- Do not place objects on top of the cables or other wiring lines. Doing so may break the cables.
- Resume operation only after transferring to the new MC Unit the contents of the parameters, position data, and other data required for resuming operation. Not doing so may result in an unexpected operation.
- Resume operation only after transferring to the new CPU Unit the contents of the DM Area, HR Area, and other data required for resuming operation. Not doing so may result in an unexpected operation.
- Confirm that set parameters and data operate properly.
- Carefully check the user program before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Do not attempt to take any Units apart, to repair any Units, or to modify any Units in any way.
- Perform wiring according to specified procedures.

6 Conformance to EC Directives

6-1 Applicable Directives

- EMC Directives
- Low Voltage Directive

6-1-1 Concepts

EMC Directives

OMRON devices that comply with EC Directives also conform to the related EMC standards so that they can be more easily built into other devices or machines. The actual products have been checked for conformity to EMC standards (see the following note). Whether the products conform to the standards in the system used by the customer, however, must be checked by the customer. EMC-related performance of the OMRON devices that comply with EC Directives will vary depending on the configuration, wiring, and other conditions of the equipment or control panel in which the OMRON devices are installed. The customer must, therefore, perform final checks to confirm that devices and the over-all machine conform to EMC standards.

Note Applicable EMC (Electromagnetic Compatibility) standards are as follows:

EMS (Electromagnetic Susceptibility): EN61131-2
EMI (Electromagnetic Interference): EN50081-2
(Radiated emission: 10-m regulations)

Low Voltage Directive

Always ensure that devices operating at voltages of 50 to 1,000 VAC or 75 to 1,500 VDC meet the required safety standards for the PC (EN61131-2).

6-1-2 Conformance to EC Directives

The C200HX/HG/HE series and CS1 series PCs comply with EC Directives. To ensure that the machine or device in which a PC is used complies with EC directives, the PC must be installed as follows:

1,2,3...

1. The PC must be installed within a control panel.
2. Reinforced insulation or double insulation must be used for the DC power supplies used for the communications and I/O power supplies.
3. PCs complying with EC Directives also conform to the Common Emission Standard (EN50081-2). When a PC is built into a machine, however, noise can be generated by switching devices using relay outputs and cause the overall machine to fail to meet the Standards. If this occurs, surge killers must be connected or other measures taken external to the PC.

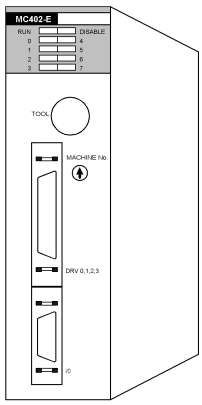
   The following methods represent typical methods for reducing noise, and may not be sufficient in all cases. Required countermeasures will vary depending on the devices connected to the control panel, wiring, the configuration of the system, and other conditions.

SECTION 1
Features and System Configuration

This section describes the features and system configuration of the C200HW-MC402-E Motion Control Unit and concepts related to its operation. It also indicates the difference with the previous C200HW-MC402-UK Unit.

1-1 Features 2
1-1-1 Overview 2
1-1-2 Description of Features 3
1-2 System Configuration 4
1-3 Motion Control Concepts 5
1-3-1 PTP-control 7
1-3-2 CP-control 9
1-3-3 EG-Control 11
1-3-4 Other Operations 13
1-4 Control System 14
1-4-1 Feedback Pulses 14
1-4-2 Servo System Principles 16
1-5 Specifications 18
1-6 Comparison with C200HW-MC402-UK 20

1-1 Features



1-1-1 Overview

The C200HW-MC402-E Motion Control (MC) Unit is a Special I/O Unit that can perform advanced MC operations on up to four axes simultaneously. The Unit's multi-tasking BASIC motion control language provides an easy to use tool for programming advanced motion control applications.

Three types of motion control are possible: point-to-point, continuous path and electronic gearing.

Point-to-point Control

Point-to-point (PTP) control enables positioning independently for each axis. Axis specific parameters and commands are used to determine the paths for the axes.

Continuous Path Control

Continuous path (CP) control enables the user not only to control the start and end positions, but also the path between those points. Possible multi-axis paths are linear interpolation, circular interpolation, helical interpolation. Also user defined paths can be realized with the CAM control.

Electronic Gearing

Electronic gearing (EG) enables controlling an axis as a direct link to another axis. The MC Units supports electronic gear boxing, linked moves and CAM movements and adding all movements of one axis to another.

The MC Unit can be used in many applications. The following areas have been identified as applicable areas for the MC Unit.

- Packaging
- Automotive welding
- Coil winding
- Web control
- Cut to length
- Drilling
- Electronic component assembly
- Glue laying
- Flying shears
- Laser guidance
- Milling
- Palletisation
- Tension control

There are many other types of machines that can be controlled by the MC Unit.

1-1-2 Description of Features

The MC Unit provides the following features.

Easy Programming with BASIC Motion Control Language

A multi-task BASIC motion control language is used to program the MC Unit. A total of 14 programs can be held in the Unit and up to 5 tasks can be run simultaneously. Programs can read and write to the PC memory areas using simple commands from BASIC or the IORD/IOWR instructions from the PC's ladder program.

Windows-based Programming Software

The MC Unit is programmed using a Windows-based application called [1]Motion Perfect. Motion Perfect allows extremely flexible programming and debugging.

Virtual Axes

The MC Unit contains a total of 8 axes, which consists of 4 servo axes and 4 virtual axes. The virtual axes acts as a perfect servo axes and are used for computational purposes for creating profiles. They can be linked directly to the servo axes.

PC Data Exchange

The coordination of the MC Unit with the CPU Unit is largely improved by modifying the PC Data Exchange interface. The PC Data Exchange interface now even more allows a centralized control from the PC. The MC Unit uses the full functionality of the C200HX/HG/HE or CS1 PC. It is now capable of both exchanging fast control bits via the IR/CIO area as exchanging large position profile data directly to the MC Unit's Table array.

Hardware-based Registration Inputs

There is a high-speed registration input for each axis. On the rising or falling edge of a registration input, the MC Unit will store the current position in a register. The registered position can then be used by the BASIC program as required. The registered positions are captured in hardware.

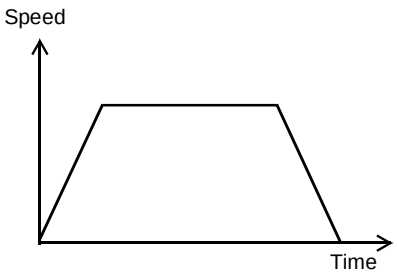
General-purpose Input and Output Signals

Starting, stopping, limit switching, origin searches and many other functions can be controlled without the use of PC I/O. The time required to switch an output or read an input is thus not dependant on the cycle time. The general I/O are freely allocable to the different functions.

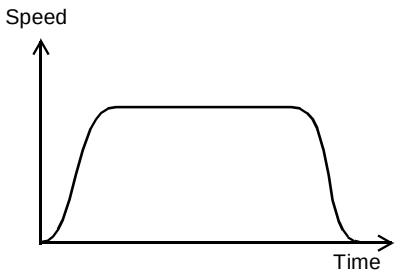
Reduced Machine Wear

The traditional trapezoidal speed profile is provided to generate smooth starting and stopping. The trapezoidal corners can be rounded off to S-curves.

Trapezoidal Speed Profile with Square Corners



Trapezoidal Speed Profile with S-curve Corners

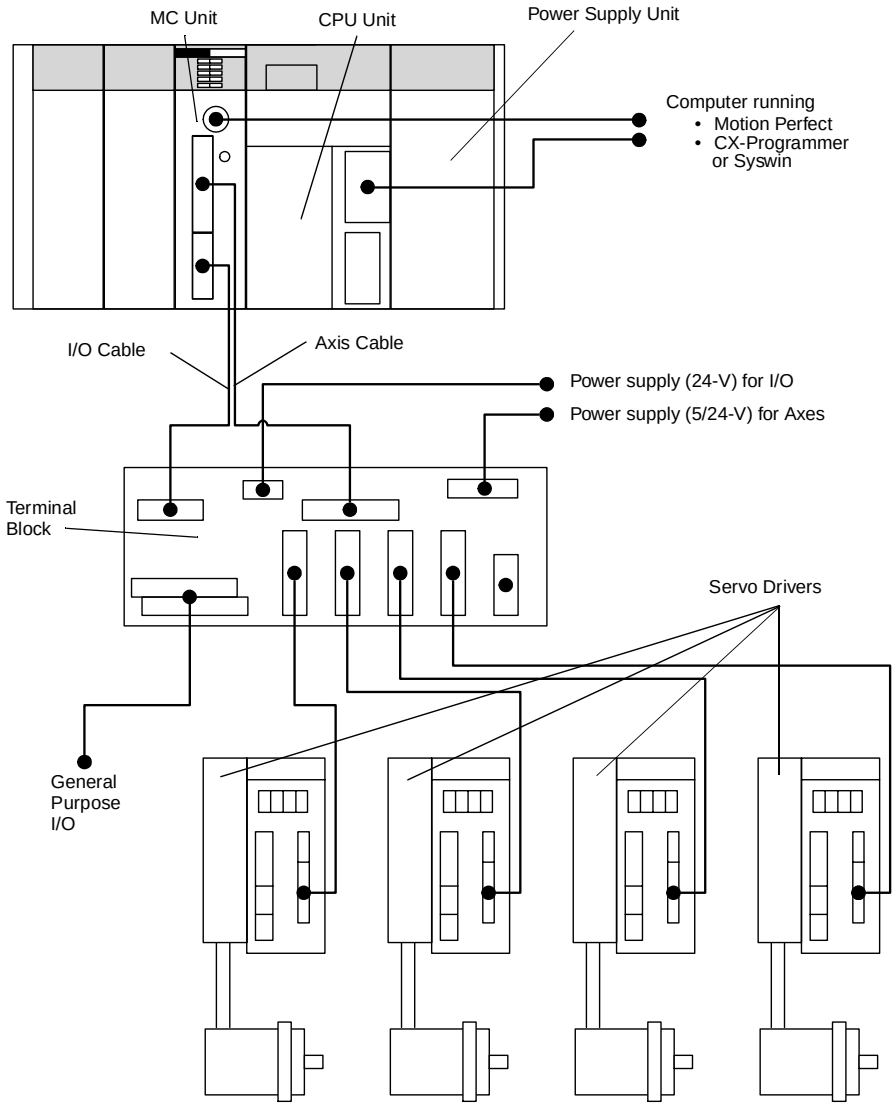


1.Motion Perfect is a product of Trio Motion Technology Limited.

1-2 System Configuration

Basic System Configuration

The basic system configuration of the MC Unit is shown below. The diagram shows the basic physical components of a coordinated motion control application.



The equipment and models which can be used in the system configuration are shown in the following table.

Device	Model
Motion Control Unit	C200HW-MC402-E
CPU Unit	Possible models: C200HX/HG/HE C200HS CS1H/CS1G
Power Supply Unit	Possible models: C200HW-PA204 C200HW-PA204S
CPU Backplane	Possible models: C200HW-BC031/BC051/BC81/BC101 CS1W-BC023/BC033/BC053/BC083/BC103
Terminal Block	R88A-TC04-E

Device	Model
Personal Computer (for Motion Perfect)	IBM Personal Computer or 100% compatible
Motion Perfect	Version 2.0 or later
Servo Driver	R88D-UA, -UT, -W series
Servomotor	R88M-UA, -UT, -W series
Inverter	3G3FV in Flux Vector Control

Note

1. The MC Unit cannot be mounted to a C200H PC.
2. The C200HS CPU Units do not support the IORD/IOWR instructions. The MC Unit can only communicate with a C200HS CPU Unit using the PLC_READ and PLC_WRITE commands.
3. The MC Unit cannot be mounted to a SYSMAC BUS Slave Rack.
4. The MC Unit can be mounted next to the CPU Unit on the CPU Rack, but care must be taken to first determine the mounting locations of certain Communications Unit and other Units that require bus connections to the CPU Unit.

Cables to be supplied by the user

The following standard cables are available. A cable can also be prepared by the user.

Item	Model
R88A-CMX001S-E	I/O Connection Cable from MC Unit to Terminal Block (1m)
R88A-CMS001J1-E	Axis Connection Cable from MC Unit to Terminal Block (1m)
R88A-CMU001J2-E	Connection from Terminal Block to UA Servo Driver (1m)
R88A-CMUK001J3-E	Connection from Terminal Block to UT Servo Driver (1m)
R88A-CMUK001J3-E2	Connection from Terminal Block to UT/W Servo Driver (1m)
R88A-CCM002P4-E	Connection Cable RS-232C from MC Unit to computer (2m)

1-3 Motion Control Concepts

The MC Unit offers the following types positioning control operations.

1. Point-to-point control
2. Continuous Path control
3. Electronic Gearing

This section will introduce some of the commands and parameters as used in the BASIC programming of the motion control application. Refer to *SECTION 5 BASIC Motion Control Programming Language* for details.

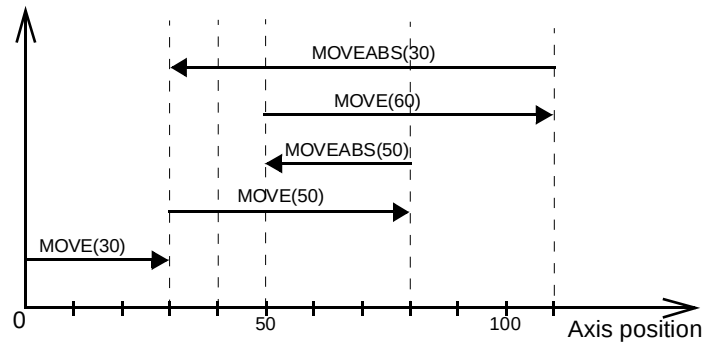
Coordinate System

Positioning operations performed by the MC Unit are based on an axis coordinate system. The MC Unit converts the encoder edges and pulses from the encoder into an internal absolute coordinate system.

The engineering unit which specifies the distances of travelling can be freely defined for each axis separately. The conversion is performed through the use of the unit conversion factor, which is defined by the UNITS axis parameter. The origin point of the coordinate system can be determined using the DEFPOS command. This command re-defines the current position to zero or any other value.

A move is defined in either absolute or relative terms. An absolute move takes the axis to a specific predefined position with respect to the origin point. A relative move takes the axis from the current position to a position that is defined relative to this current position. The following diagram shows gives an exam-

ple of relative (command MOVE) and absolute (command MOVEABS) linear moves.



Axis Types

The MC Unit has 8 axes in total, which can be used for different motion control purposes depending on the application. The type of each axis is determined by the ATYPE axis parameter. The following table lists the different available axis types.

Axis type	ATYPE value	Description
Virtual	0	A virtual axis is used for computational purposes to create a move profile without physical movement on any actual Servo Driver. All move commands and axis parameters available for the servo axis are also available for the virtual axis and the virtual axis behaves like a perfect servo axis (demanded position is equal to the actual position). Possible application for the virtual axis is having a virtual move profile added to a servo axis or to test a developed application before controlling the actual motors. Axis range: [0, 7]
Servo	2	The servo axis controls the connected Servo Driver. Based on the calculated movement profile and the measured position feedback of the Servomotor the proper speed reference is outputted to the Servo Driver. Axis range: [0, 3]
Encoder	3	The encoder axis defines an axis which provides an encoder input without the servo control speed reference output to the system. An encoder can be connected for measurement, registration and/or synchronization functions. Axis range: [0, 3]

Refer to *1-4 Control System* for details on the servo system and encoder feedback signals. Axes 0 to 3 are servo axes by default and axes 4 to 7 are fixed as virtual axes.

1-3-1 PTP-control

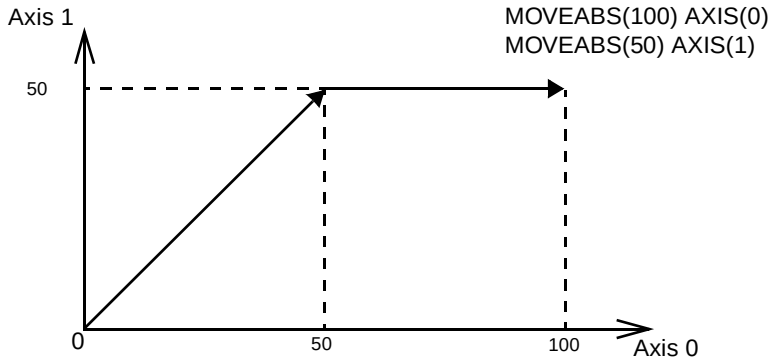
In point-to-point positioning, each axis is moved independently of the other axis. The MC Unit supports the following operations.

- Relative move
- Absolute move
- Continuous move forward
- Continuous move reverse

Relative and Absolute Moves

To move a single axis either the command MOVE for a relative move or the command MOVEABS for an absolute move is used. Each axis has its own move characteristics, which are defined by the axis parameters.

Suppose a control program is executed to move from the origin to an axis no. 0 coordinate of 100 and axis no. 1 coordinate of 50. If the speed parameter is set to be the same for both axes and the acceleration and deceleration rate are set sufficiently high, the movements for axis 0 and axis 1 will be as illustrated below.



At start, both the axis 0 and axis 1 will move to a coordinate of 50 over the same duration of time. At this point, axis 1 will stop and the axis 0 will continue to move to a coordinate of 100.

Relevant Axis Parameters

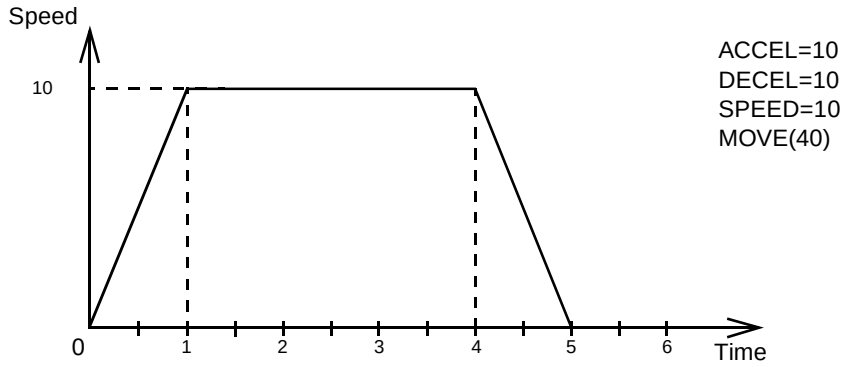
As mentioned before the move of a certain axis is determined by the axis parameters. Some relevant parameters are given in the next table.

Parameter	Description
UNITS	Unit conversion factor
ACCEL	Acceleration rate of an axis in units/s^2.
DECEL	Deceleration rate of an axis in units/s^2.
SPEED	Demand speed of an axis in units/s.

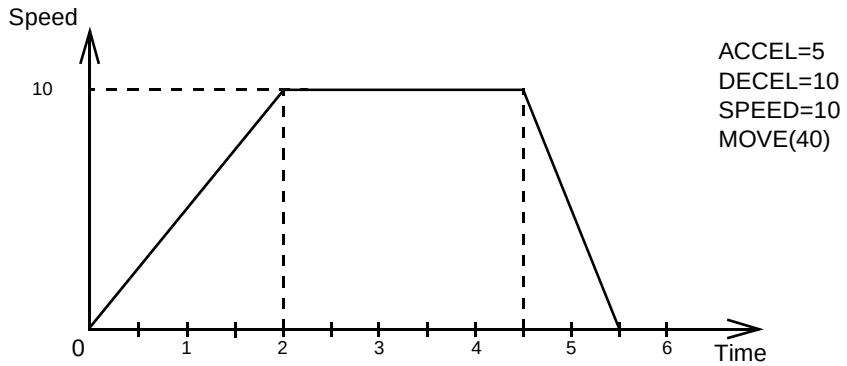
Defining moves

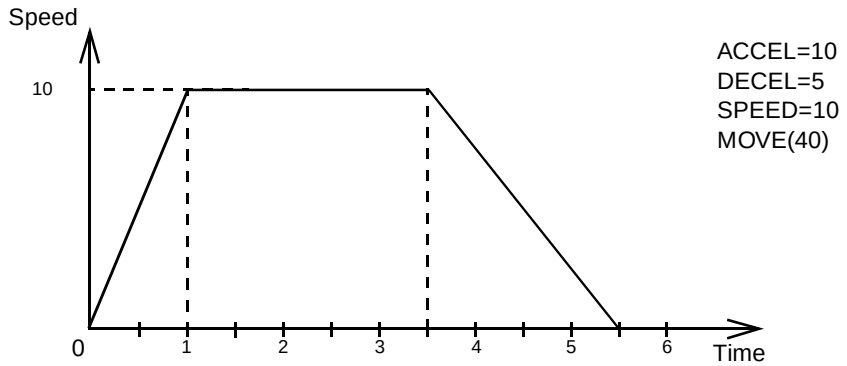
The speed profile below shows a simple MOVE operation. The UNITS parameter for this axis has been defined for example as meters. The required maximum speed has been set to 10 m/s. In order to reach this speed in one second and also to decelerate to zero speed again in one second, both the acceleration as the deceleration rate have been set to 10 m/s^2. The total distance travelled is the sum of distances travelled during the acceleration, constant speed and deceleration segments. Suppose the distance moved by the

MOVE command is 40 m, the speed profile will be given by the following graph.



The following two speed profiles show the same movement with an acceleration time respectively a deceleration time of 2 seconds.





Move Calculations

The following equations are used to calculate the total time for the motion of the axes. Consider the moved distance for the MOVE command as D, the demand speed as V, the acceleration rate a and deceleration rate d.

Acceleration time $= \frac{V}{a}$

Acceleration distance $= \frac{V^2}{2a}$

Deceleration time $= \frac{V}{d}$

Deceleration distance $= \frac{V^2}{2d}$

Constant speed distance $= D - \frac{V^2(a+d)}{2ad}$

Total time $= \frac{D}{V} + \frac{V(a+d)}{2ad}$

Continuous Moves

The FORWARD and REVERSE commands can be used to start a continuous movement with constant speed on a certain axis. The FORWARD command will move the axis in positive direction and the REVERSE command in negative direction. For these commands also the axis parameters ACCEL and SPEED apply to specify the acceleration rate and demand speed.

Both movements can be canceled by using either the CANCEL or RAPIDSTOP command. The CANCEL command will cancel the move for one axis and RAPIDSTOP will cancel moves on all axes.

1-3-2 CP-control

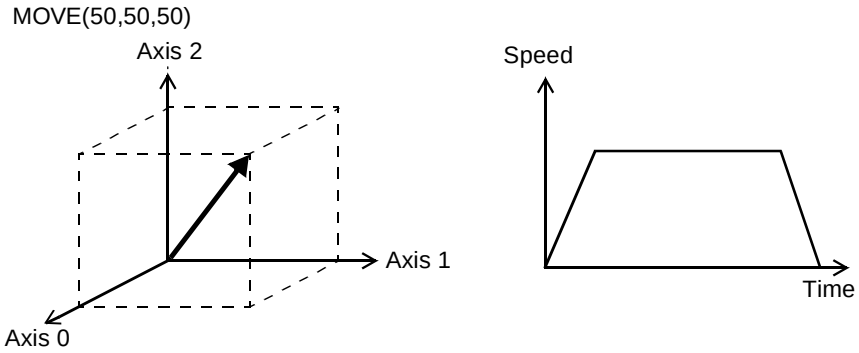
Continuous Path control enables to control a specified path between the start and end position of a movement for one or multiple axes. The MC Unit supports the following operations.

- Linear interpolation
- Circular interpolation
- Helical interpolation
- CAM control

Linear Interpolation

In applications it can be required for a set of motors to perform a move operation from one position to another in a straight line. Linearly interpolated moves can take place among several axes. The commands MOVE and MOVEABS are also used for the linear interpolation. In this case the commands will have

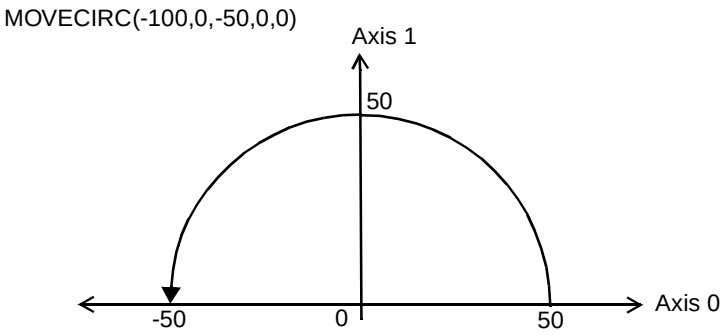
multiple arguments to specify the relative or absolute move for each axis. Consider the following three axis move in a 3-dimensional plane.



The speed profile of the motion along the path is given in the diagram. The three parameters SPEED, ACCEL and DECEL which determine the multi axis movement are taken from the corresponding parameters of the base axis. The MOVE command computes the various components of speed demand per axis.

Circular Interpolation

It may be required that a tool travels from the starting point to the end point in an arc of a circle. In this instance the motion of two axes is related via a circular interpolated move using the MOVECIRC command. Consider the following diagram.

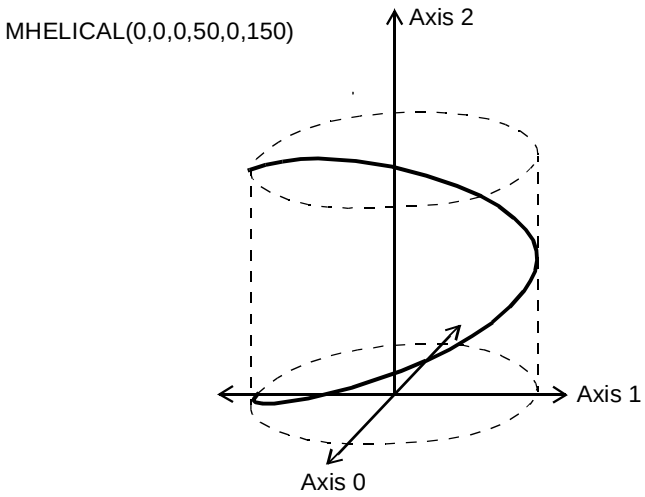


The centre point and desired end point of the trajectory relative to the start point and the direction of movement are specified. The MOVECIRC command computes the radius and the angle of rotation. Like the linearly interpolated MOVE command, the ACCEL, DECEL and SPEED variables associated with the base axis determine the speed profile along the circular move.

Helical Interpolation

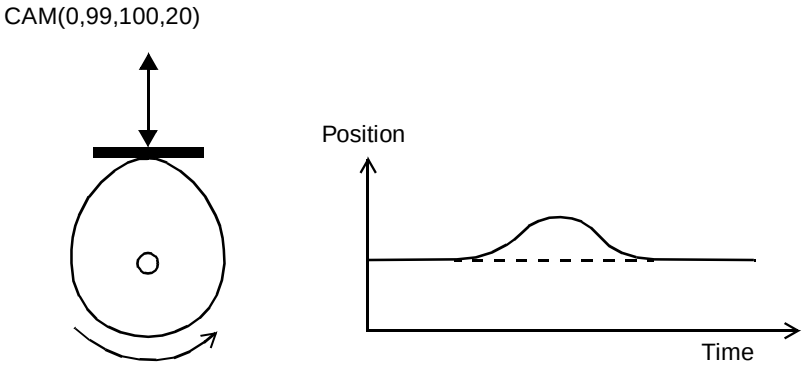
Helical interpolation performs a helical movement on three axes. The motion control command MHELICAL will perform a circular interpolation to two axis and will add a linear move to the third axis. Positioning is performed by again specifying the centre point, end point and direction for the circular distance

and the distance for the third axis. The diagram shows helical interpolation in a three dimensional plane for axes 0 to 2.



CAM Control

Additional to the standard move profiles the MC Unit also provides a way to define a position profile for the axis to move. The CAM command will move an axis according to position values stored in the MC Unit Table array. The speed of travelling through the profile is determined by the axis parameters of the axis.



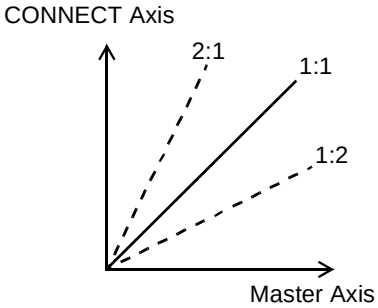
1-3-3 EG-Control

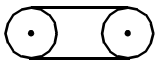
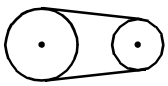
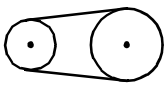
Electronic Gearing control allows you to create a direct gearbox link or a linked move between two axes. The MC Unit supports the following operations.

1. Electronic gearbox
2. Linked CAM
3. Linked move
4. Adding axes

Electronic Gearbox

The MC Unit is able to have a gearbox link from one axis to another as if there is a physical gearbox connecting them. This can be done using the CONNECT command in the program. In the command the ratio and the axis to link to are specified.

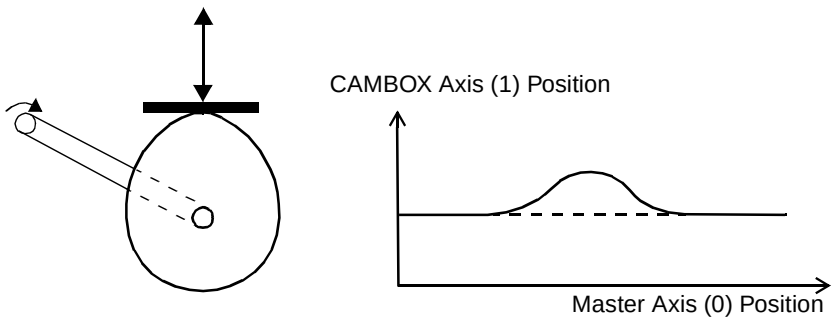


Axes		Ratio	CONNECT command
0	1		
		1:1	CONNECT(1,0) AXIS(1)
		2:1	CONNECT(2,0) AXIS(1)
		1:2	CONNECT(0.5,0) AXIS(1)

Linked CAM control

Next to the standard CAM profiling tool the MC Unit also provides a tool to link the CAM profile to another axis. The command to create the link is called CAMBOX. The travelling speed through the profile is not determined by the axis parameters of the axis but by the position of the linked axis. This is like connecting two axes through a cam.

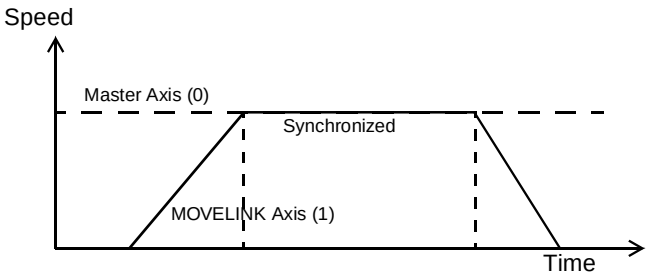
CAMBOX(0,99,100,20,0) AXIS(1)



Linked Move

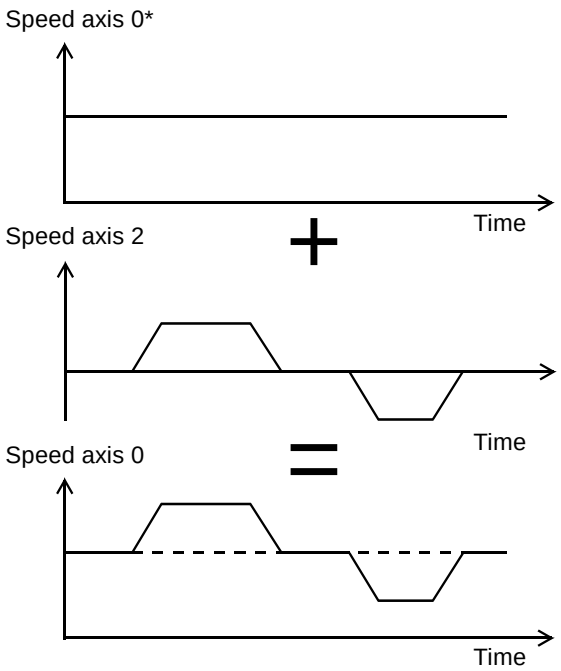
The MOVELINK command provides a way to link a specified move to a master axis. The move is divided into an acceleration, deceleration and constant speed part and they are specified in master link distances. This can be particularly useful for synchronizing two axes for a fixed period.

MOVELINK(50,60,10,10,0) AXIS(1)



Adding Axes

It is very useful to be able to add all movements of one axis to another. One possible application is for instance changing the offset between two axes linked by an electronic gearbox. The MC Unit provides this possibility by using the ADDAX command. The movements of the linked axis will consists of all movements of the actual axis plus the additional movements of the master axis.




```
BASE(0)
ADDAX(2)
FORWARD
MOVE(100) AXIS(2)
MOVE(-60) AXIS(2)
```

1-3-4 Other Operations

Canceling Moves

In normal operation or in case of emergency it can be necessary to cancel the current movement from the buffers. When the CANCEL or RAPIDSTOP commands are given, the selected axis respectively all axes will cancel their current move.

Origin Search

The encoder feedback for controlling the position of the motor is incremental. This means that all movement must be defined with respect to an origin point. The DATUM command is used to set up a procedure whereby the MC Unit

goes through a sequence and searches for the origin based on digital inputs and/or Z-marker from the encoder signal.

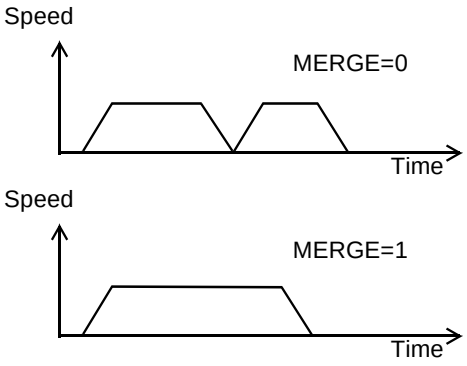
Print Registration

The MC Unit can capture the position of an axis in a register when an event occurs. The event is referred to as the print registration input. On the rising or falling edge of an input signal, which is either the Z-marker or an input, the MC Unit captures the position of an axis in hardware. This position can then be used to correct possible error between the actual position and the desired position. The print registration is set up by using the REGIST command.

The position is captured in hardware, and therefore there is no software overhead and no interrupt service routines, eliminating the need to deal with the associated timing issues. Each servo axis has one registration input.

Merging Moves

If the MERGE axis parameter is set to 1, a movement will always be followed by a subsequent movement without stopping. The following illustrations will show the transitions of two moves with MERGE value 0 and value 1.



Jogging

Jogging moves the axes at a constant speed forward or reverse by manual operation of the digital inputs. Different speeds are also selectable by input. Refer to the FWD_JOG, REV_JOG and FAST_JOG axis parameters.

1-4 Control System

1-4-1 Feedback Pulses

The MC Unit is designed to comply with the standard OMRON Servomotors which have an incremental encoder output. In this section, the signals produced by an incremental optical quadrature encoder are discussed. Incremental encoders are available in linear as well as the more common rotary types.

Incremental Encoders

The incremental encoder are encoders for which the output position information is relative to a starting position and only the distance moved is measured.

The main components of the rotary incremental encoder are an encoder disk, light source and photodetectors, plus an amplification circuitry to “square-up” the photodetector output. The encoder disk is imprinted with marks or slots evenly spaced around its perimeter. As the disk rotates, light strikes the photodetector at the passing of each slot or mark. Amplifiers then convert the photodetector output to square wave form.

Quadrature signals are produced by using two photodetectors, one positioned precisely one half a slot, or marker width, from the other. So quadrature refers to two periodic functions separated by a quarter cycle or 90°.

With this arrangement, the direction of rotation can be easily detected by monitoring the relative phase of both signals. For example, if channel A leads channel B, then counterclockwise (CCW) movement could be indicated. Con-

versely, if channel B leads channel A, then clockwise (CW) movement would be indicated.

Typically, rotary encoders also provide an additional Z-marker or slot on the disk used to produce a reference pulse. By properly decoding and counting these signals, the direction of motion, speed, and relative position of the encoder can be determined.

The number of output pulses produced per revolution per channel is equivalent to the number of marks around the disk. This position information is decoded in encoder edges, which is actually the number of pulses multiplied by four. The resolution is multiplied because the circuit generates a pulse at any rising or falling edge of either of the two phase signals.

Decoding

Understanding how the signals generated by a quadrature encoder are decoded will help considerably when applying the quadrature decoder feature in an actual situation.

The basic task of the decoder is to provide two counter input lines: one that produces clock pulses when CCW motion is detected and another that produces clock pulses when CW motion is detected. These clock pulses are supplied to counters in the MC Unit, one for CW counts and one for CCW counts. The contents of the counters can be compared with each other by software, and the relative position of the rotary device can be determined from the difference.

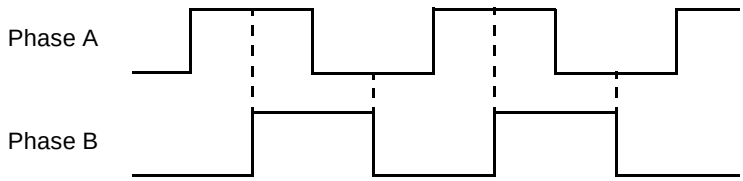
One advantage of this approach is that the actual counting is done by hardware devices, freeing the MC Unit for other operations. The MC Unit has only to periodically read the counter values and to make a quick subtraction.

Decoder Theory of Operation

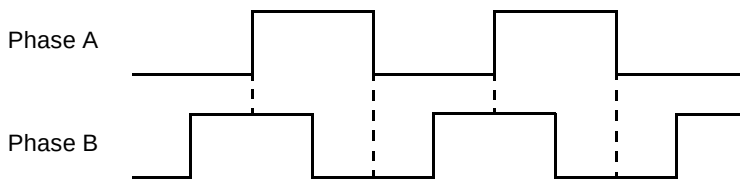
A closer look at the quadrature signals will be helpful. In this example, the direction of rotation is CCW if phase A leads phase B, and CW if phase A leads phase B.

The decoder circuit detects a transition and generates a pulse on the appropriate counter input channel depending on whether the transition is in the CW or CCW direction. Although time is plotted on the horizontal axis, it is not necessarily linear. The mechanical device may be changing speed as well as direction.

Forward Rotation



Reverse Rotation



Standard OMRON Servomotors are designed for an advanced A-phase for forward rotation and an advanced B-phase for reverse rotation. The MC Unit is designed to comply with this phase advancement, allowing OMRON Servo Driver Connecting Cables to be used without modification.

For typical OMRON Servo Drivers, there are 1,000 pulses per revolution. This implies that there are 4,000 edges per revolution. So there will be a Z pulse every 4,000 edges.

The signals A, B and Z appear physically as A and /A, B and /B and Z and /Z. These appear as differential signals on twisted-pair wire inputs, ensuring that common modes are rejected and that the noise level is kept to a minimum.

When using Servomotors by other makers, check carefully the encoder specification for phase advancement. If the definition differs from the ones given above, reverse the B-phase wiring between the MC Unit and the Servo Driver. In most case, this should resolve the problem.

1-4-2 Servo System Principles

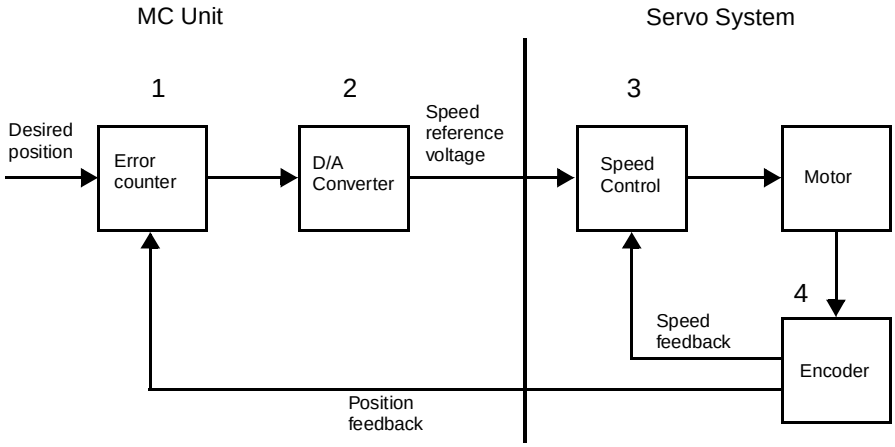
The servo system used by and the internal operation of the MC Unit are briefly described below. Refer to *2-4 Servo System Precautions* for precautions related to servo system operation.

Inferred Closed Loop System or Semi-closed Loop System

The servo system of the MC Unit uses an inferred closed loop system. This system detects actual machine movements by the rotation of the motor in relation to a target value. It calculates the error between the target value and actual movement, and reduces the error through feedback.

Internal Operation of the MC Unit

Inferred closed loop systems occupy the mainstream in modern servo systems applied to positioning devices for industrial applications. Commands to the MC Unit, speed control voltages to the Servo Drivers, and feedback signals from the encoder are described in the next few pages.



1,2,3...

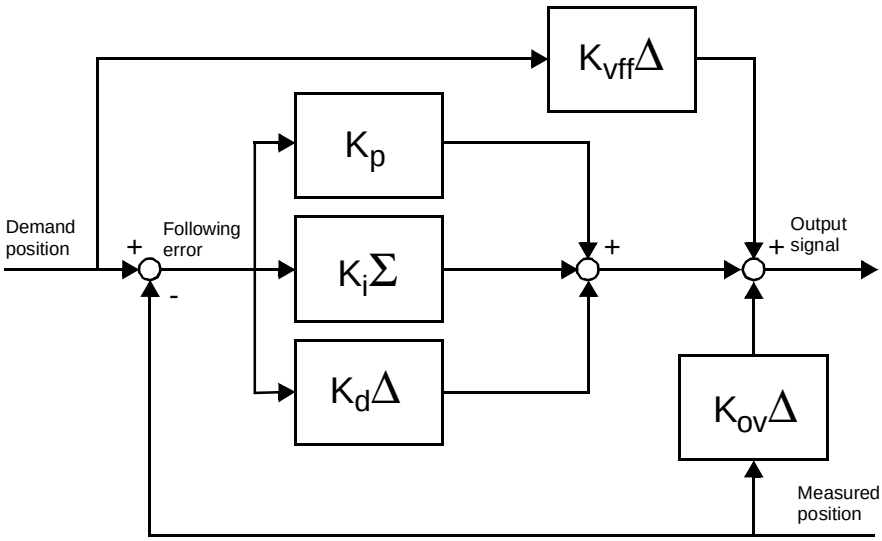
1. The MC Unit performs actual position control. It receives encoder pulses and calculates the required speed reference from the difference between the actual position and the desired position.
2. The calculated desired speed is directly converted by the D/A converter into an analogue speed reference voltage, which is provided to the Servo Driver.
3. The Servo Driver controls the rotational speed of the Servomotor corresponding to the speed reference input.
4. The rotary encoder will generate the feedback pulses for both the speed feedback within the Servo Driver speed loop and the position feedback within the MC Unit position loop.

Motion Control Algorithm

The servo system controls the motor by continuously adjusting the voltage output that serves as a speed reference to the Servo Driver. The speed reference is calculated by comparing the measured position of the axis from the encoder with the demand position generated by the MC Unit.

The axis parameters MPOS, DPOS and FE contain the value of respectively the measured position, demand position and the following error. The following error is the difference between the demanded and measured position. MC Unit uses five gain values to control how the servo function generates the voltage output from the following error.

The control algorithm for the motion control system of the MC Unit is shown in the diagram below. The five gains are described below.



Proportional Gain

The proportional gain K_p creates an output O_p that is proportional to the following error E.

$$O_p = K_p \cdot E$$

All practical systems use proportional gain. For many just using this gain parameter alone is sufficient. The proportional gain axis parameter is called P_GAIN.

Integral Gain

The integral gain K_i creates an output O_i that is proportional to the sum of the following errors E that have occurred during the system operation.

$$O_i = K_i \cdot \sum E$$

Integral gain can cause overshoot and so is usually used only on systems working at constant speed or with slow accelerations. The integral gain axis parameter is called I_GAIN.

Derivative Gain

The derivative gain K_d produces an output O_d that is proportional to the change in the following error E and speeds up the response to changes in error while maintaining the same relative stability.

$$O_d = K_d \cdot \Delta E$$

Derivative gain may create a smoother response. High values may lead to oscillation. The derivative gain axis parameter is called D_GAIN.

Output Speed Gain

The output speed gain K_{ov} produces an output O_{ov} that is proportional to the change in the measured position P_m and increases system damping.

$$O_{ov} = K_{ov} \cdot \Delta P_m$$

The output speed gain can be useful for smoothing motions but will generate high following errors. The output speed gain axis parameter is called OV_GAIN.

Speed Feedforward Gain

The speed feedforward gain K_{vff} produces an output O_{vff} that is proportional to the change in demand position P_d and minimizes the following error at high speed.

$$O_{vff} = K_{vff} \cdot \Delta P_d$$

The parameter can be set to minimise the following error at a constant machine speed after other gains have been set. The speed feed forward gain axis parameter is called VFF_GAIN.

Default Values

The default settings are given below along with the resulting profiles. Fractional values are allowed for gain settings.

Gain	Default
Proportional Gain	1.0
Integral Gain	0.0
Derivative Gain	0.0
Output Speed Gain	0.0
Speed Feedforward Gain	0.0

1-5 Specifications

General Specifications

General specifications other than those shown below conform to those for the SYSMAC C200HS/C200HX/C200HG/C200HE PCs.

Item	Specifications
Power supply voltage	5 VDC (from Backplane)
	24 VDC (from external power supply)
Voltage fluctuation tolerance	4.75 - 5.25 VDC (from Backplane)
	21.6 - 26.4 VDC (from external power supply)
Internal current consumption	600 mA or less for 5 VDC
	50 mA or less for 24 VDC
Weight (Connectors excluded)	500 g max.
External Dimensions	130.0 x 35 x 100.5 mm (H x W x D)

Functional Specifications

Item		Contents
Type of Unit		C200H Special I/O Unit
Applicable PC		C200HX/HG/HE and CS1
Backplanes on which MC Unit can be mounted		CPU Backplane
Method for data transfer to CPU Unit	Words allocated in IR/CIO area	10 words per unit (See note 1.)
	PC and MC Unit instructions	Any number of words modified by ladder program or BASIC program instruction
External connected devices		Personal computer with Motion Perfect Programming Software
Controlled Servo Drivers		Analogue (speed) input Servo Drivers
Control	Control method	Inferred closed loop with incremental encoder and with PID, output speed and speed feed forward gains
	Maximum No. of axes	8
	Maximum No. of interpolated axes	8
	Maximum No. of servo axes	4
	Maximum No. of virtual axes	8

Item		Contents
Speed control		Speed control of up to 4 axes
Measurement Units		User definable
Positioning operations	Linear interpolation	Linear interpolation for any number of axes
	Circular interpolation	Circular interpolation for any two axes
	Helical interpolation	Helical interpolation for any three axes
	CAM profile	CAM profile movement for any axis
	Electronic gearbox	Electronic gearbox link between any two axes
	Linked CAM	Linked CAM profile movement for any two axes
	Linked move	Linked move for any two axes
	Adding axes	Adding any two axes
Encoder interface		Line receiver input; maximum response frequency: 250 kp/s (before multiplication) 1 M counts/s (after multiplication)
Acceleration/deceleration curves		Trapezoidal or S-curve
External I/O	Serial Communication ports	One RS-232C port for connection to computer with the Motion Perfect software. One RS-232C port for general purpose.
	Encoder	Line receive inputs: For four axes (250 kp/s, before multiplication)
	Servo Driver relationship	The following signals are provided. **Inputs:** Driver Alarm Signal (each axis) **Outputs:** Driver Enable (all axes) Speed Reference Voltage (each axis) Driver Alarm Reset (all axes)
	General Purpose I/O	Up to 16 digital inputs and 8 outputs can be wired to control MC Unit functions. These can include limit switches, emergency stop switches and proximity inputs.
	Registration inputs	Each servo axis has a registration input which capture the position in hardware. Timing specification (see note 2): Digital Input (rising edge): 10 µs (max.) Digital Input (falling edge): 200 µs (max.) Z-marker (rising edge): 2 µs (max.) Z-marker (falling edge): 2 µs (max.)
Power supply for general and axis I/O		Provided externally
Task program management	Programming language	BASIC
	Number of tasks	Up to 5 tasks running simultaneously plus the Command Line Interface task
	Max. number of programs	14
	Data storage capacity	251 (VR) + 16000 (Table) max.
	Data transfer to PC Unit	PLC_READ and PLC_WRITE command in BASIC program, IORD and IOWR instructions in ladder program in C200HX/HG/HE PCs

Item		Contents
Saving program data	MC Unit	Battery-backed RAM with flash memory backup. (See note 3.)
	External devices	Motion Perfect software manages a backup on the hard disk of the personal computer.
Self diagnostic functions		Detection of memory corruption via checksum Detection of error counter overrun

Note
1. The number of MC Units that can be mounted under one CPU Unit must be determined based on the maximum number of Special I/O Units that can be allocated words in the CPU Units, the power supply capacity on the CPU or Expansion Rack, and the current consumption of the Units mounted to the Rack. Refer to the CPU Unit's operation manual for details on calculation methods.
2. This specification is the time between the edge in the input signal and the capture of the position data.
3. The service life for the flash memory is 100,000 writing operations.

1-6 Comparison with C200HW-MC402-UK

The following table shows a comparison between the C200HW-MC402-E Unit and the previously released C200HW-MC402-UK Unit.

Item		C200HW-MC402-UK	C200HW-MC402-E
Applicable PCs		C200HS,C200HX/HG/HE (HX up to CPU64)	C200HS,C200HX/HG/HE and CS1
Supported axes		4 (4 servo)	8 (4 servo and 4 virtual)
Allocated IR (CIO) area words		6 words (6 input)	10 words (8 input, 2 output). See note.
Compatible software		Motion Perfect 1.24 and 2.0	Motion Perfect 2.0
Serial Port A		Motion Perfect connection and freely programmable	Dedicated for Motion Perfect connection
Cyclic Servo Period		Set by SERVO_PERIOD parameter (default 1 ms)	Fixed to 1 ms
Commands and instructions	PLC_READ/ PLC_WRITE	Yes	Yes Also reads/writes allocated IR(CIO) area words
	IORD/IOWR	Yes Read/write to MC Unit's VR array in one-word format	Yes Read/write to MC Unit's VR and Table array and one-word and three-word format supported
	CLEAR_BIT/ SET_BIT/ READ_BIT	No	Yes Enables bit operation for VR variables
	INPUT/KEY/ LINPUT	No	Yes Added functionality for serial communications
	PROC	No	Yes. Allows a process parameter of a particular task to be read/written
	INDEVICE/ OUTDEVICE	Yes	No Port 0 is default port for serial communication
	CLEAR/RESET	No	Yes Commands to clear memory

Item		C200HW-MC402-UK	C200HW-MC402-E
Commands and instructions	APPENDPROG/ AXISVALUES/ EX/ INPUTS0/ INPUTS1/ LOADSYSTEM/ MPE/ STORE	Yes	Commands reserved for Motion Perfect: descriptions have been removed from manual
	WAIT LOADED/ LIST	No	Yes Added functionality

Note The allocation of the IR/CIO area bits has been modified in comparison with the C200HW-MC402-UK. Please refer to *3-1 IR/CIO Area Allocation* for more information.

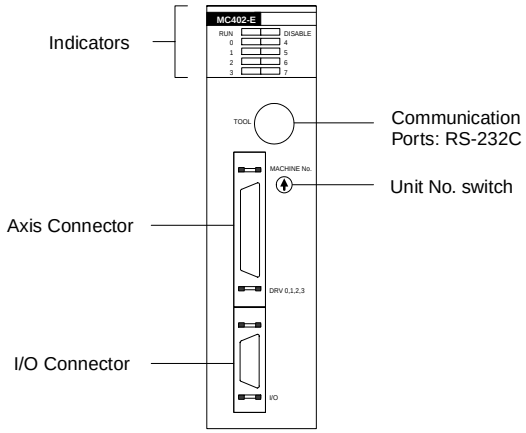
SECTION 2
Installation

This section describes the MC Unit components and provides the information required for installing the MC Unit.

2-1 Components and Unit Settings 24
2-2-1 Installation Method 25
2-2-2 Dimensions 26
2-2 Installation 25
2-3-1 Connector Pin Assignments 26
2-3-2 I/O Specifications 29
2-3-3 Serial Port Connections 31
2-3-4 Terminal Block 32
2-3-5 Connection Examples 34
2-3 Wiring 26
2-4 Servo System Precautions 36
2-5 Wiring Precautions 38

2-1 Components and Unit Settings

The following diagram shows the main components of the MC Unit.



Indicators

The following table describes the indicators on the front of the MC Unit.

Indicator	Color	Status	Meaning
RUN	Green	ON	The MC Unit is operating normally.
		OFF	The MC Unit is not recognized by the PC at initialization or is malfunctioning.
		Flashing alone	The battery voltage is low.
		Flashing with DISABLE	An error occurred in the communication between MC Unit and CPU Unit.
DISABLE	Red	ON	The axes have been disabled. The Servo Enable Output is not ON.
		OFF	The axes are enabled.
		Flashing alone	The following error has exceeded the limit. The Servo Drives have been disabled.
		Flashing with RUN	An error occurred in the communication between MC Unit and CPU Unit.
0 to 7	Orange	ON	These indicators can be controlled from the programs. Refer to *5-3-53 DISPLAY* for details.
		OFF	

Unit No. Switch

Set the number the unit number between 0 and F.

CPU Unit	Unit No. setting range
C200HS-CPU01-E/21-E/31-E/03-E/23-E/33-E C200HE-CPU11-E/32-E/42-E/11-ZE/32-ZE/42-ZE, C200HG-CPU33-E/43-E/33-ZE/43-ZE, C200HX-CPU34-E/44-E/34-ZE/44-ZE	0 to 9
C200HG-CPU53-E/63-E/53-ZE/63-ZE, C200HX-CPU54-E/64-E/54-ZE/64-ZE/85-ZE CS1H-CPU66-E/65-E/64-E/63-E CS1G-CPU45-E/44-E/43-E/42-E	0 to F

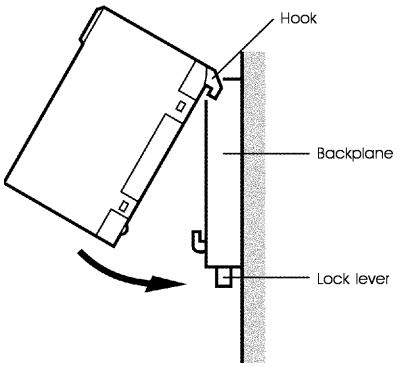
⚠ **Caution** Do not change the unit number while power is being supplied to the Unit.

2-2 Installation

2-2-1 Installation Method

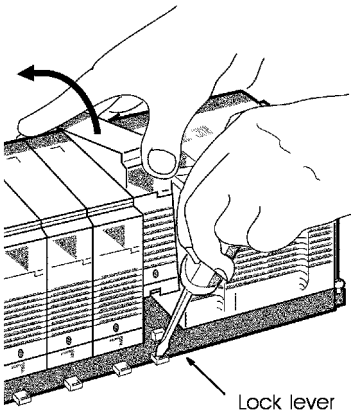
1,2,3...

1. Attach the hooks on the upper section of the MC Unit onto the Backplane.
2. Insert the MC Unit connector into the Backplane connector.



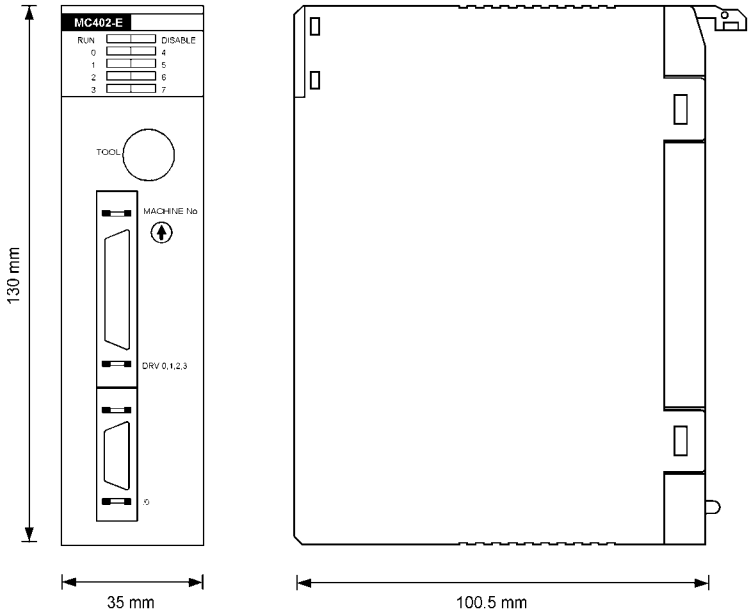
⚠ **Caution** Do not mount the MC Unit while the power is turned ON to the Rack.

When removing the MC Unit, lift it out while pressing down on the lock lever with a screwdriver, as shown in the following illustration.



2-2-2 Dimensions

The basic dimensions of the MC Unit are shown below.



2-3 Wiring

2-3-1 Connector Pin Assignments

I/O Connector

The I/O Connector is used for wiring to external I/O. All I/O are general purpose and functions like limit inputs and origin proximity inputs can be allocated. Inputs I0 / R0 to I3 / R3 can also be used as the Registration Inputs for axis 0 to 3. Refer to *2-3-2 I/O Specifications* for electrical specifications.

Recommended Connector and Cable

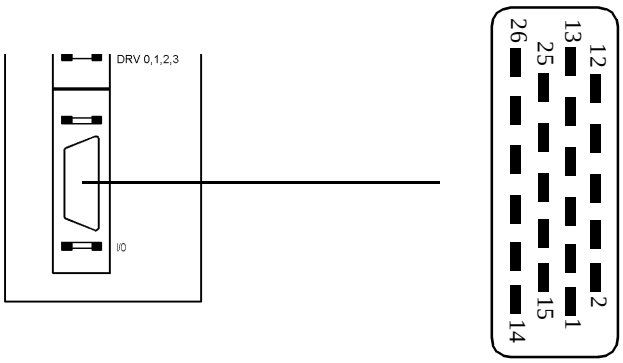
The 3M model numbers are listed below.

Connector	IDC plug connectors with metal backshells	Soldered connector with plastic shells
26-pin MDR	10126-6000EC or 10126-6000EL 10326-A200-00	10126-3000VE or 10126-3000VC 10326-52F0-0086

The IDC or soldered connectors can be used with various types of cables. The following 3M cable is recommended for the MC Unit.

- Round-Jacketed, Shielded, Discrete Wire Cable 3444C-series, 28 AWG Stranded, Twisted-pair, PVC/PVC.

Connector pin arrangement



I/O Connector Pin Functions

Pin	Signal	
	Name	Function
1	24V_IO	24V supply for I/O circuits
2	O0	Output 0
3	O1	Output 1
4	O2	Output 2
5	O3	Output 3
6	O4	Output 4
7	O5	Output 5
8	O6	Output 6
9	O7	Output 7
10	I0 / R0	Input 0 or Registration input for axis 0
11	I1 / R1	Input 1 or Registration input for axis 1
12	I2 / R2	Input 2 or Registration input for axis 2
13	0V_IO	0V common for I/O circuits
14	I3 / R3	Input 3 or Registration input for axis 3
15	I4	Input 4
16	I5	Input 5
17	I6	Input 6
18	I7	Input 7
19	I8	Input 8
20	I9	Input 9
21	I10	Input 10
22	I11	Input 11
23	I12	Input 12
24	I13	Input 13
25	I14	Input 14
26	I15	Input 15

Axis Connector

The Axis Connector is used to connect the Servo Drivers for axes 0 to 3. Refer to *2-3-2 I/O Specifications* for electrical specifications.

Recommended Connector and Cable

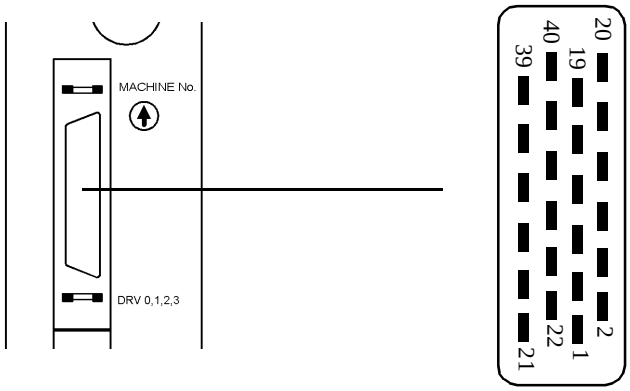
The 3M model numbers are listed below.

Connector	IDC plug connectors with metal backshells	Soldered connector with plastic shells
40-pin MDR	10140-6000EC or 10140-6000EL 10340-A200-00	10140-3000VE or 10140-3000VC 10340-5500-008

The IDC or soldered connectors can be used with various types of cables. The following 3M cable is recommended for the MC Unit.

- Round-Jacketed, Shielded, Discrete Wire Cable 3444C-series, 28 AWG Stranded, Twisted-pair, PVC/PVC.

Connector pin arrangement



Axis Connector Pin Functions

Pin	Signal	
	Name	Function
1	0V_DRV	0V common for control signals
2	/ALARM_0	Alarm input for axis 0
3	/ALARM_1	Alarm input for axis 1
4	/ALARM_2	Alarm input for axis 2
5	A_0	Encoder phase A axis 0
6	/A_0	Encoder phase /A axis 0
7	B_0	Encoder phase B axis 0
8	/B_0	Encoder phase /B axis 0
9	Z_0	Encoder phase Z axis 0
10	/Z_0	Encoder phase /Z axis 0
11	VREF_0	Speed reference signal axis 0
12	0V_ENC	0V common for encoder signals
13	A_1	Encoder phase A axis 1
14	/A_1	Encoder phase /A axis 1
15	B_1	Encoder phase B axis 1
16	/B_1	Encoder phase /B axis 1
17	Z_1	Encoder phase Z axis 1
18	/Z_1	Encoder phase /Z axis 1
19	VREF_1	Speed reference signal axis 1
20	0V_REF	0V common for reference signals
21	/ALARM_3	Alarm input for axis 3
22	ALARMRST	Drivers alarm reset signal
23	ENABLE	Drivers enable signal
24	24V_DRV	24V supply for driver control signals
25	A_2	Encoder phase A axis 2
26	/A_2	Encoder phase /A axis 2
27	B_2	Encoder phase B axis 2
28	/B_2	Encoder phase /B axis 2
29	Z_2	Encoder phase Z axis 2
30	/Z_2	Encoder phase /Z axis 2
31	VREF_2	Speed reference signal axis 2
32	0V_ENC	Ground encoder signals
33	A_3	Encoder phase A axis 3
34	/A_3	Encoder phase /A axis 3
35	B_3	Encoder phase B axis 3
36	/B_3	Encoder phase /B axis 3

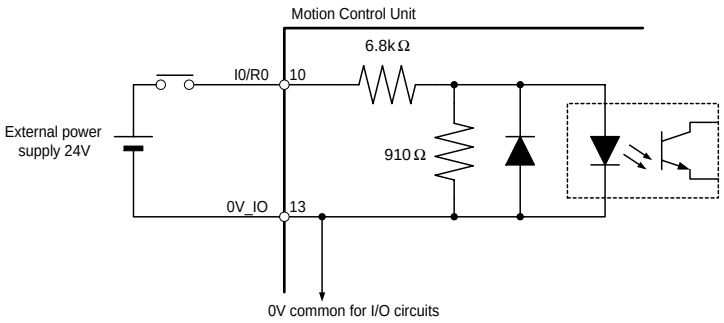
Pin	Signal	
	Name	**Function**
37	Z_3	Encoder phase Z axis 3
38	/Z_3	Encoder phase /Z axis 3
39	VREF_3	Speed reference signal axis 3
40	0V_REF	0V common for speed reference signals

Note The 0V_REF and 0V_ENC pins are connected inside the MC Unit.

2-3-2 I/O Specifications

The following tables provide specifications and circuits for the Axis and I/O connections.

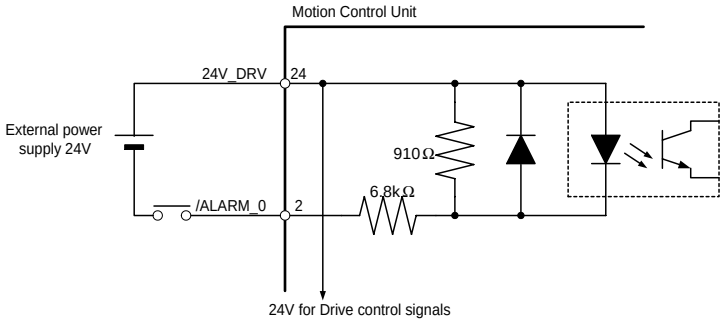
Digital Inputs

I/O inputs: I0 to I15		
Item	**Specification**	**Circuit Configuration**
Type	PNP	
Maximum voltage	24 VDC + 10%	
Input current	3.2 mA at 24 VDC	
ON voltage	12 V min.	
OFF voltage	5 V max.	
ON response time (see note)	1.8 ms (max.)	
OFF response time (see note)	2.1 ms (max.)	

Note The given response time is the time between the change in the input voltage and the corresponding change in the IN variable. This time includes the physical delays in the input circuit.

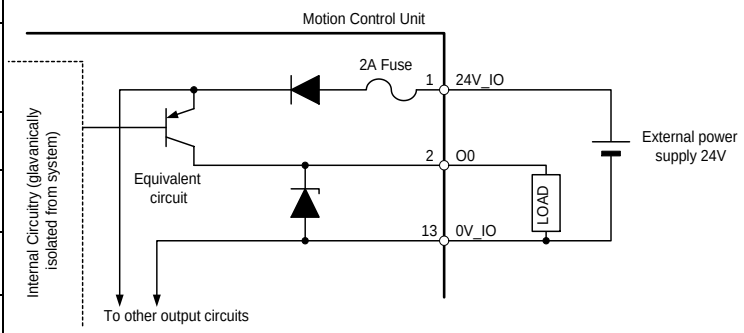
⚠ Caution Maximum 12 of the digital inputs (I0 to I15) should be switched on at any one time to ensure that the Unit remains within internal temperature specifications. Failure to meet this condition may lead to degradation of performance or damage of components.

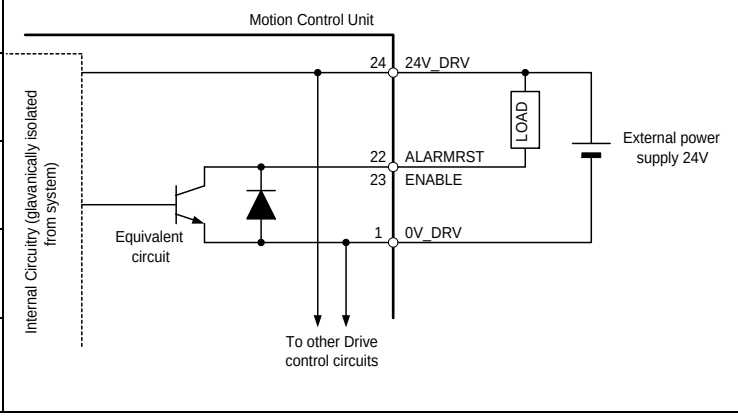
Please refer to *1-5 Specifications* for timing specification on print registration using inputs I0/R0 to I3/R3.

Axis inputs: ALARM (axis 0 to 3)		
Item	**Specification**	**Circuit Configuration**
Type	NPN	
Maximum voltage	24 VDC + 10%	
Input current	3.2 mA at 24 VDC	
ON voltage	12 V min.	
OFF voltage	5 V max.	
ON response time (see note)	1.8 ms (max.)	
OFF response time (see note)	2.1 ms (max.)	

Note The given response time is the time between the change in the input voltage and the corresponding change in the IN variable. This time includes the physical delays in the input circuit.

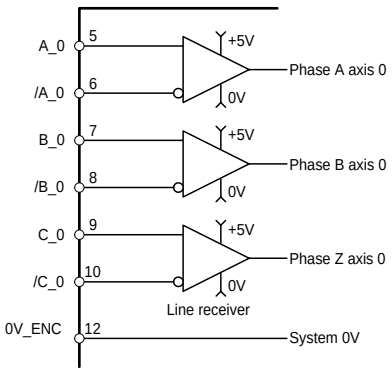
Digital Outputs.

I/O outputs: O0 to O7		
Item	**Specification**	**Circuit Configuration**
Type	PNP	
Current capacity	100 mA each output (800 mA total for group of 8)	
Maximum voltage	24 V + 10%	
ON response time (see note)	1.3 ms (max.)	
OFF response time (see note)	1.4 ms (max.)	
Protection	Over current, over temperature and 2 A fuse on common	

Axis outputs: ENABLE, ALARMRST		
Item	**Specification**	**Circuit Configuration**
Type	NPN	
Current capacity	80 mA each output	
Maximum voltage	24 V + 10%	
ON response time (see note)	1.3 ms (max.)	
OFF response time (see note)	1.4 ms (max.)	

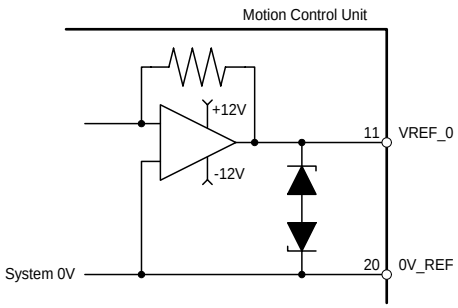
Note The given response time is the time between a change in the OP or WDOG variable and the corresponding change in the digital output signal. This time includes the physical delays in the output circuit.

Encoder Input

Item	Specification	Circuit Configuration
Signal level	EIA RS-422-A Standards	
Input impedance	48 kΩ min.	
Response frequency	250 kp/s	
Termination	None (see note)	

Note Termination resistors can be mounted on the Terminal Block if required (see section *2-5 Wiring Precautions*).

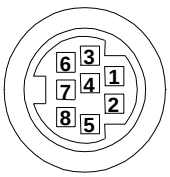
Analogue Output

Item	Specification	Circuit Configuration
Output Voltage	0 to ±10 V	
Resolution	12-bit	
Output impedance	100 Ω	
Load impedance	10 kΩ min.	

2-3-3 Serial Port Connections

The MC Unit has two serial RS-232C ports for communication with external devices. Port A is the programming port of the unit, connect this port to the computer to configure the Unit using the Motion Perfect software package. Port B can be used for connection to other external devices.

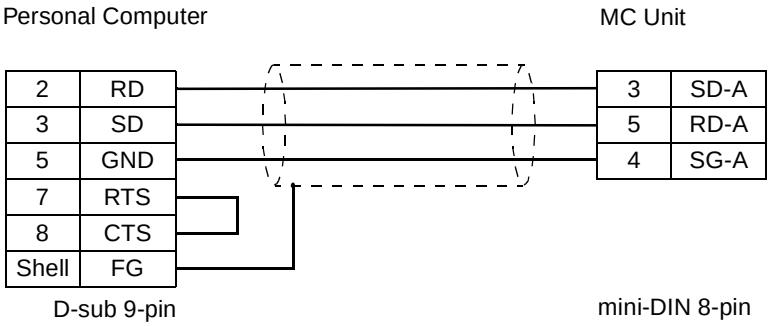
The table below shows the connector on the MC Unit (8-pin mini-DIN) and the pin allocation for both RS-232C ports.

Pin Layout	Pin	Symbol	Name	Port
	1	-	Not used	-
	2	-	Not used	-
	3	SD-A	Send data	A
	4	SG-A	Signal ground	A
	5	RD-A	Receive data	A
	6	SD-B	Send data	B
	7	SG-B	Signal ground	B
	8	RD-B	Receive data	B

You can use the following connection cable for connection to the computer.

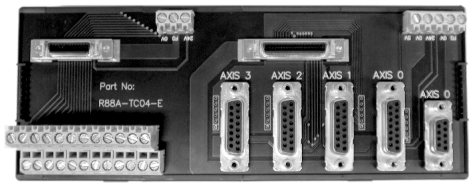
Product	Description
R88A-CCM002P4-E	Connection cable RS-232C (2m)

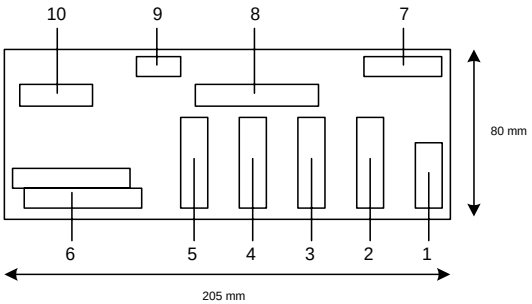
The connections to the computer are shown below.



2-3-4 Terminal Block

The Terminal Block (Quick Connect Kit) can be used to facilitate the connections to the Servo Drivers and other devices. The Terminal Block can be mounted on a DIN rail.





The table below shows the various items on the unit.

Item	Description	Connection Type
1	Axis 0 encoder output	9-pin D-sub (female)
2	Axis 0 Servo Driver connection	15-pin D-sub (female)
3	Axis 1 Servo Driver connection	15-pin D-sub (female)
4	Axis 2 Servo Driver connection	15-pin D-sub (female)
5	Axis 3 Servo Driver connection	15-pin D-sub (female)
6	I/O connections	Screw terminals
7	24V and 5V supply for Axis connections	Screw terminals
8	Axis connection to MC Unit	40-pin MDR socket (female)
9	24V supply for I/O connections	Screw terminals
10	I/O connection to MC Unit	26-pin MDR socket (female)

Dimensions

The unit's dimensions are 205mm x 80mm x 57 mm (L x H x D) without the cables connected.

Cable and Connector Parts

The available ready-made cables together with the Terminal Block are shown in the next table.

Product	Description
R88A-TC04-E	Terminal Block
R88A-CMX001S-E	I/O connection cable from MC Unit to Terminal Block (1m)
R88A-CMS001J1-E	Axis connection cable from MC Unit to Terminal Block (1m)
R88A-CMU001J2-E	Connection from Terminal Block to UA Servo Driver (1m)
R88A-CMUK001J3-E	Connection from Terminal Block to UT Servo Driver (1m)
R88A-CMUK001J3-E2	Connection from Terminal Block to UT/W Servo Driver (1m)

Pin Allocations

Axis D-sub 15-Pin

The pin layout of the15-pin D-sub connectors, which are used for item 2 to 5, is shown in the next table.

Pin	Signal	
	Name	Function
1	0V_DRV	0V common for control signals
2	/ALARM	Alarm input for axis
3	ALARMRST	Drive alarm reset signal
4	0V_ENC	0V common for encoder signals
5	A	Encoder phase A
6	B	Encoder phase B
7	Z	Encoder phase Z
8	VREF	Speed reference signal
9	24V_DRV	24V power supply for control signals
10	ENABLE	Driver enable signal
11	5V_ENC	5V power supply for encoder
12	/A	Encoder phase /A
13	/B	Encoder phase /B
14	/Z	Encoder phase /Z
15	0V_REF	0V common for reference signal

Axis D-sub 9-Pin

The Terminal Block has a second 9-pin D-sub connection for Axis 0 (item 1) to enable the encoder signals of this axis to be outputted. This can be used to cascade the signals through to another MC Unit with Terminal Block.

Pin	Signal	
	Name	Function
1	0V_DRV	0V common for control signals
2	A	Encoder phase A
3	B	Encoder phase B
4	Z	Encoder phase Z
5	-	-
6	/A	Encoder phase /A
7	/B	Encoder phase /B
8	/Z	Encoder phase /Z
9	-	-

I/O Connections

The order of the pins for the I/O connections (item 6) is as follows. Refer to *2-3-1 Connector Pin Assignments* for the pin descriptions.

I15	I13	I11	I9	I7	I5	R3	R1	O7	O5	O3	O1
I14	I12	I10	I8	I6	I4	R2	R0	O6	O4	O2	O0

Power Supplies

There are 2 sets of terminals for supplying power to the interface unit

1. The 24V and optional 5V supply for the Axes part (item 7).
2. The 24V supply for the I/O connections to the unit (item 9).

The 24V power supply to the Axis connection and the I/O connection should in principle be separate. This will ensure 500V RMS galvanic isolation between the two circuits.

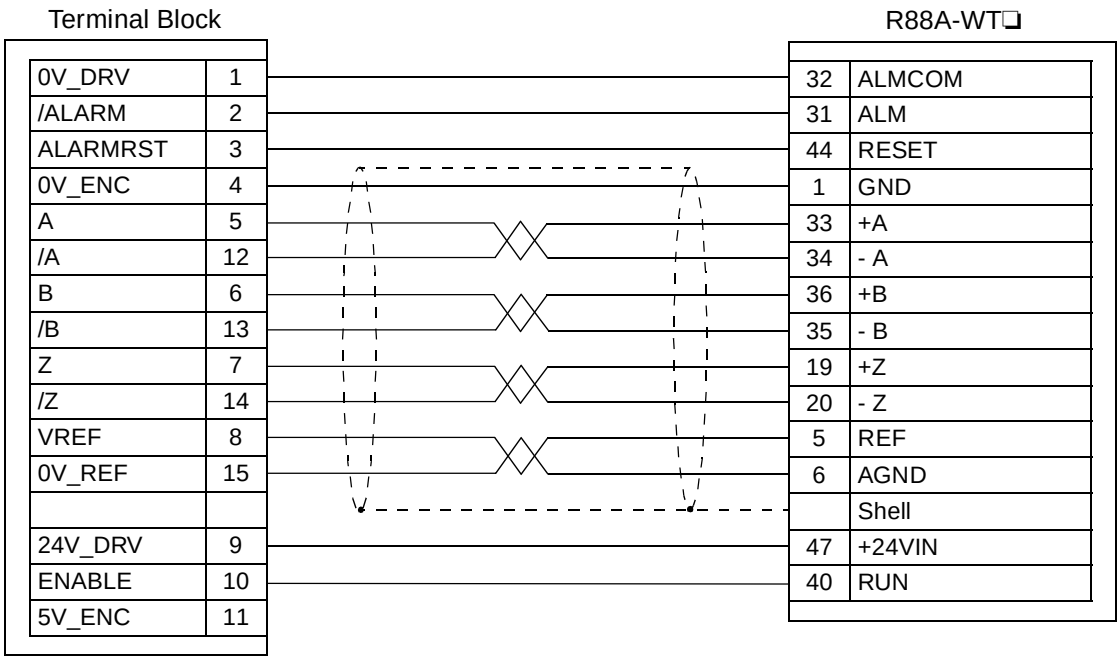
The 5V power supply is used to supply power an Omron FV Driver or a standalone line driver encoder feedback is used.

Terminating resistors

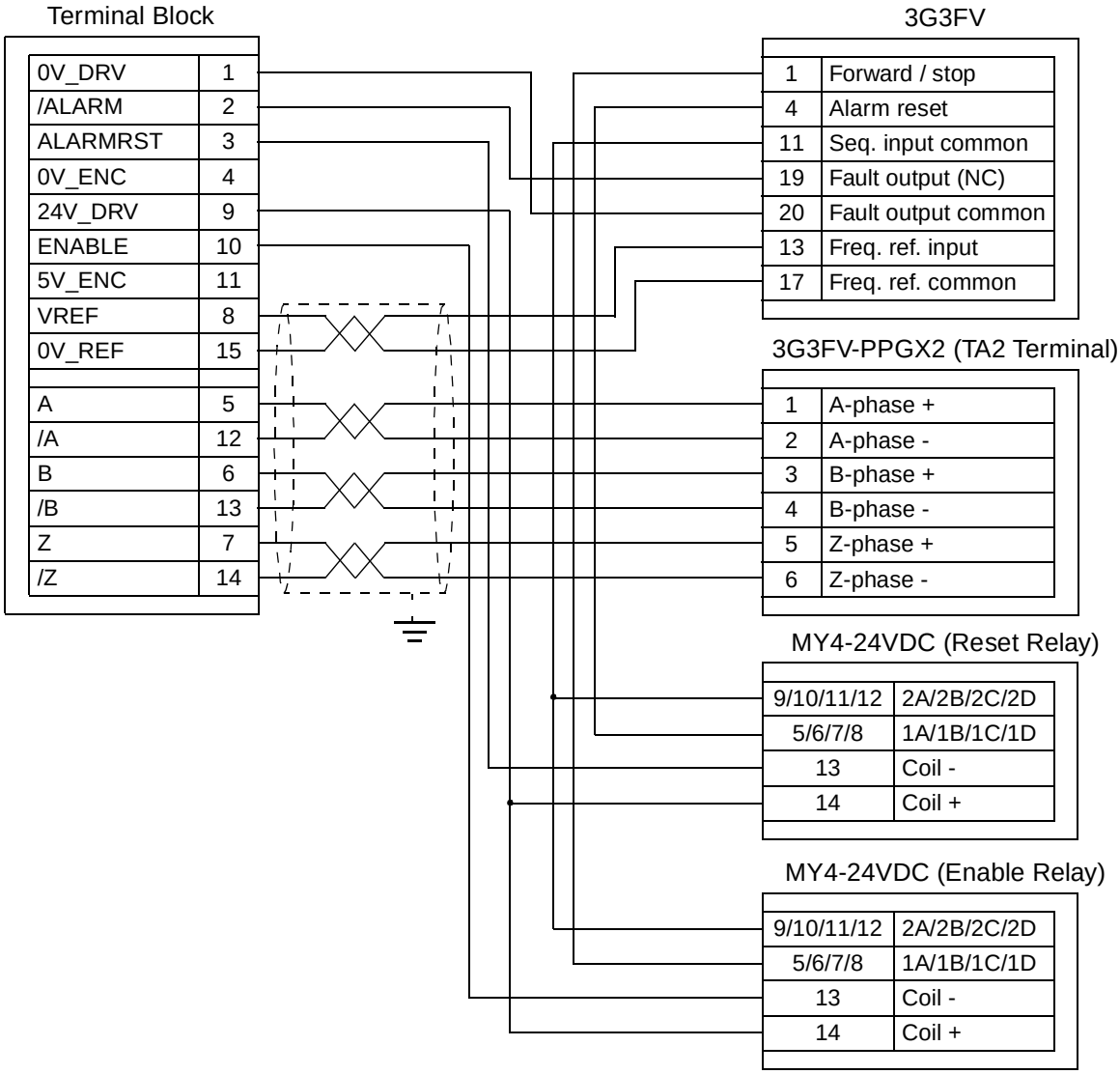
Immediately next to each item 1-5 there is a 6 pin through-hole connection that allows placement of terminating resistors on the encoder A, B and Z signals. These resistors will have to be soldered onto the sites by competent personnel. The resistor pack recommended for this operation is the 220 Ω / 0.2 W resistors e.g. Bourns 4306R-102-221, which contains 6 isolated resistors in one package.

2-3-5 Connection Examples

W Driver



3G3FV Inverter



2-4 Servo System Precautions

The following precautions are directly related to the operation of the servo system. Refer to *1-4-2 Servo System Principles* for a description of servo system operation.

Motor Runaway

In a servo system employing a Servomotor, faulty or disconnected wiring may cause the Servomotor to run out of control. Therefore, careful attention must be paid to preventing faulty or disconnected wiring.

When the wiring is correct, the Servomotor will maintain the stopped position through corrective operations as long as a position loop is formed and servolock is in effect.

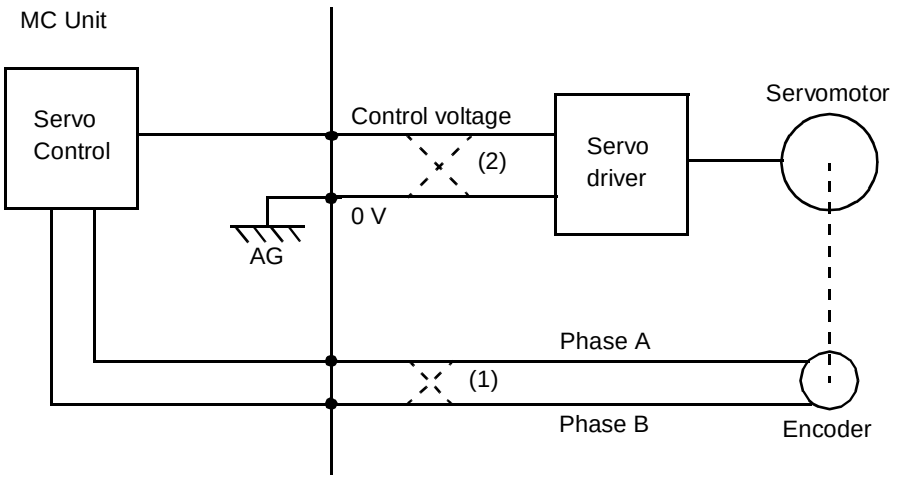
If the motor rotates in the CW direction due to a factor such as temperature drift, it is detected by the encoder and the internal error counter of the MC Unit is notified of the direction and amount of rotation by means of feedback signals output by the encoder.

The count of the error counter is ordinarily zero unless otherwise designated. When the motor moves in the CW direction, the feedback signal transfers the direction and amount of movement as a count to the error counter. In response, the MC Unit outputs a control voltage to rotate the motor in the CCW direction to zero the error count.

The control voltage is output to the Servo Driver, and the Servomotor rotates in the CCW direction. If the motor rotates in this CCW direction, the encoder detects the direction and amount of movement and notifies the error counter in the MC Unit with feedback signals to subtract and zero the count again.

The position loop subtracts the count in the error counter to zero it.

The analogue ground is common among all axes, preventing the reversal of axes by swapping the wires. The reversal can easily be achieved in software inside the Servo Driver using the PP_STEP command.



Runaway Caused by Faulty Wiring

If the phase-A and phase-B feedback input lines are wired in reverse (crossed dotted lines at 1 in the figure), the servolock will not be effective and the motor will run out of control.

1,2,3...

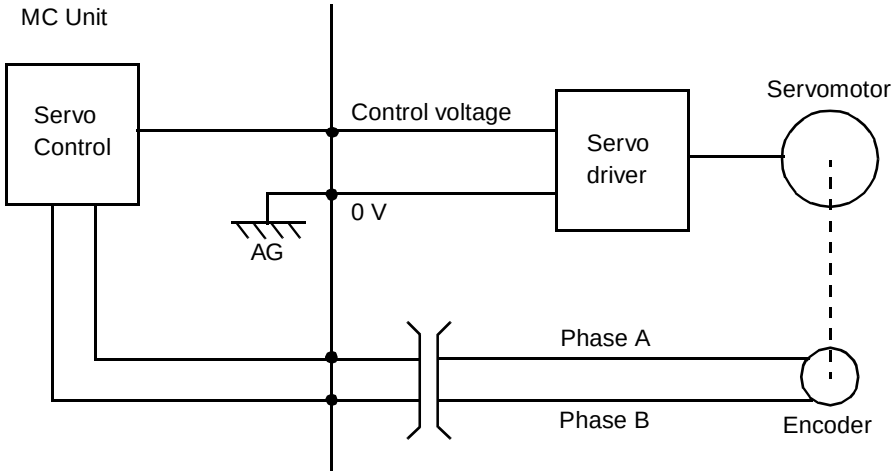
1. If the phase-A and phase-B feedback input lines are wired in reverse, the error counter will receive the information as a rotation in the CCW direction.
2. If the motor rotates in the CW direction due to drift or some other cause, the encoder will detect the direction and amount of movement and transmit feedback signals to the error counter in the MC Unit.

3. As a result, the error counter having a count in the CCW direction will attempt to zero the count by outputting a control voltage to the Servo Driver in the CW direction.
4. The Servomotor will rotate in the CW direction, repeating the above steps 1 to 3, causing the motor to run out of control.

Runaway can occur not only from reversed wiring of phases A and B of the feedback inputs, but also from reversed wiring of the speed control voltage and the ground lines (crossed dotted lines at 2 in the figure above).

Runaway Caused by Disconnected Wiring

The Servomotor will run out of control not only when the position loop is not correctly formed, but also when the position loop is interrupted due to disconnected wiring.



1,2,3...

1. Wire Breakage with Servomotor Rotating:
   While the Servomotor is rotating, the speed control voltage is not 0 V because of the signal from the error counter. If the feedback line is broken, no feedback signals will be given to the error counter and the speed control voltage remains unchanged from the value that existed before the line breakage, causing motor runaway.
2. Wire Breakage with Servomotor Stopped:
   If the feedback line is broken while the Servomotor is stopped and correct feedback signals cannot be returned, the speed control voltage will remain at zero without changing. Therefore, the Servomotor will also remain stopped. In fact, however, the motor may move in one direction without stopping.

This is caused by a discrepancy between the 0 V of the MC Unit's control voltage and the 0 V of the Servo Driver's voltage input. When the two 0 voltages do not match, an electric potential difference is generated, resulting in a false control voltage. This in turn causes the Servomotor to move in one direction without stopping.

To prevent this, repair the wiring or adjust the 0 V of either the MC Unit or the Servo Driver so that the 0 V levels match.

Following Error Limit Setting

While following a motion profile, the servo system will generally follow the set profile but not exactly. There will be a following error. The following error limit can be set according to operating conditions using the axis parameter FE_LIMIT. If for any reason the following error exceeds this limit, the servo enable output will reset and the Servo Driver will be disabled, causing the motor to come to a sudden halt. The user must make sure that this does not have an adverse effect on the machine. See *5-3-66 FE_LIMIT* for details.

External Limit Switches

Another fail-safe condition must normally be set up using monitoring sensors installed at the edges of the workpiece's range of movement to detect abnormal workpiece movement and stop operation if a runaway occurs. This can be done by mapping the address of FWD_IN and REV_IN to the relevant digital inputs. See *5-3-75 FWD_IN* and *5-3-143 REV_IN* for details.

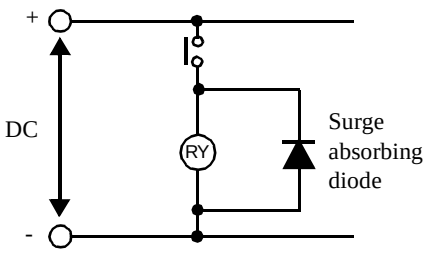
Monitoring sensors are installed outside of the limit inputs. If the workpiece reaches one of the sensors, the appropriate bit in the axis status will be turned ON. The enable signal to the Servo Driver will be turned OFF and then the dynamic brake will be applied to stop the motor.

2-5 Wiring Precautions

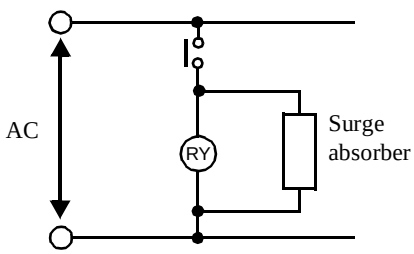
Electronically controlled equipment may malfunction because of noise generated by power supply lines or external loads. Such malfunctions are difficult to reproduce, and determining the cause often requires a great deal of time. The following precautions will aid in avoiding noise malfunctions and improving system reliability.

- Use electrical wires and cables of the designated sizes as specified in the operation manual for the Servo Driver. Use larger size cables for FG lines of the PC or the Servo Driver and ground them over the shortest possible distances.
- Separate power cables (AC power supply lines and motor power supply lines) from control cables (pulse output lines and external input signal lines). Do not group power cables and control cables together or place them in the same conduit.
- Use shielded cables for control lines.
- Use the Terminal Block and the ready-made cables designed for MC Unit to reduce connectivity problems.
- Connect a surge absorbing diode or surge absorber close to relays. Use a surge-absorbing diode with a voltage tolerance of at least five times greater than the circuit voltage.

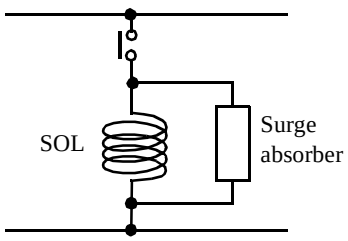
DC relay



AC relay



Solenoid



- Noise may be generated on the power supply line if the same power supply line is used for an electric welder or electrical discharge unit. Connect

an insulating transformer and a line filter in the power supply section to remove such noise.

- Use twisted-pair cables for power supply lines. Use adequate grounds (i.e., to 100 Ω or less) with wire cross sections of 1.25 mm^2 or greater.
- Use twisted-pair shielded cables for control voltage output signals, input signals and feedback signals.
- Use wires of maximum 2 m between the MC Unit and the Servo Driver for control voltage output signals.
- If the distance of the encoder from the MC Unit is more than 10 m, terminating resistors should be placed on the Terminal Block. The maximum distance for the encoder position signal from the encoder to the MC Unit must not exceed 20 m.
- The input terminals that operate the 24 V system are isolated with optical couplers to reduce external noise effects on the control system. Do not connect the analogue voltage ground and the 24 V system ground.

SECTION 3
PC Data Exchange

This section describes the IR/CIO area allocation and presents the different methods of data exchange between the MC Unit and the CPU Unit.

3-1 IR/CIO Area Allocation 42
3-1-1 Overview 42
3-1-2 Overview of IR/CIO Area Allocations 43
3-2 Overview of Data Exchanges 46
3-2-1 Data Exchange Methods 46
3-2-2 Data formats 47
3-3 Details of the Data Exchange Methods 48
3-3-1 Data Words in IR/CIO Area 48
3-3-2 Data Transfer by CPU Unit 49
3-3-3 Data Transfer by MC Unit 55

3-1 IR/CIO Area Allocation

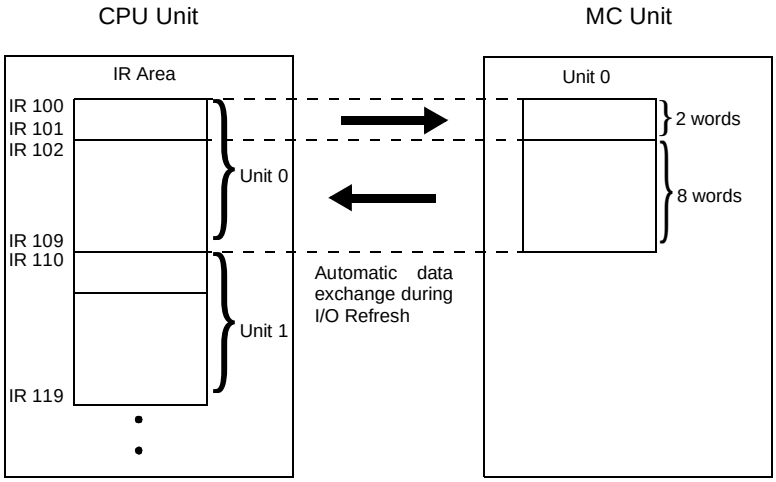
3-1-1 Overview

Each MC Unit is allocated 10 words in the Special I/O Unit Areas of the CPU Unit's IR or CIO area. The words that are allocated depend on the Unit No. set on the rotary switch on the front panel of the MC Unit. The contents of the allocated 10 words is exchanged automatically between the CPU Unit and the MC Unit every time the CPU Unit refreshes I/O.

Input and Output Words

The words allocated to the MC Unit are classified as input and output words. The input and output directions are defined from the CPU Unit's perspective.

Data Exchange for the C200HX/HG/HE, C200HS

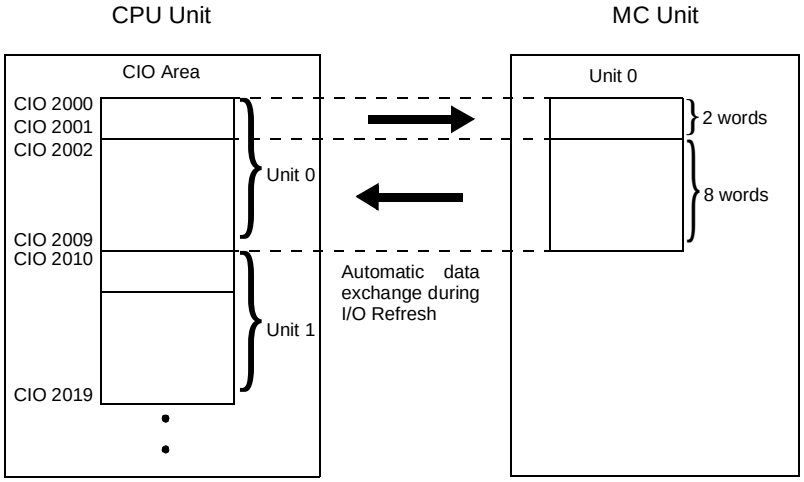


Special I/O Area Allocation for C200HX/HG/HE, C200HS

Unit no.	Allocated words	Unit no.	Allocated words	Unit no.	Allocated words	Unit no.	Allocated words
0	IR 100 to IR 109	4	IR 140 to IR 149	8	IR 180 to IR 189	C	IR 420 to IR 429 (See note.)
1	IR 110 to IR 119	5	IR 150 to IR 159	9	IR 190 to IR 199	D	IR 430 to IR 439 (See note.)
2	IR 120 to IR 129	6	IR 160 to IR 169	A	IR 400 to IR 409 (See note.)	E	IR 440 to IR 449 (See note.)
3	IR 130 to IR 139	7	IR 170 to IR 179	B	IR 410 to IR 419 (See note.)	F	IR 450 to IR 459 (See note.)

Note For C200HG-CPU53-E/63-E/53-ZE/63-ZE and C200HX-CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE CPU Units only.

Data Exchange for the CS1 Series



Special I/O Area Allocation for CS1 Series

Unit no.	Allocated words	Unit no.	Allocated words	Unit no.	Allocated words	Unit no.	Allocated words
0	CIO 2000 to CIO 2009	4	CIO 2040 to CIO 2049	8	CIO 2080 to CIO 2089	C	CIO 2120 to CIO 2129
1	CIO 2010 to CIO 2019	5	CIO 2050 to CIO 2059	9	CIO 2090 to CIO 2099	D	CIO 2130 to CIO 2139
2	CIO 2020 to CIO 2029	6	CIO 2060 to CIO 2069	A	CIO 2100 to CIO 2109	E	CIO 2140 to CIO 2149
3	CIO 2030 to CIO 2039	7	CIO 2070 to CIO 2079	B	CIO 2110 to CIO 2119	F	CIO 2150 to CIO 2159

3-1-2 Overview of IR/CIO Area Allocations

The following tables show the data which is automatically exchanged during the I/O refresh period. The first word allocated to the MC Unit in the IR/CIO area is specified as “n” (refer to the previous section). The value of “n” can be calculated from the unit number using the following equation.

- C200HX/HG/HE, C200HS

  Unit numbers 0 to 9: n = 100 + 10 x Unit number

  Unit numbers A to F: n = 400 + 10 x (Unit number - 10)
- CS1 Series

  Unit numbers 0 to F: n = 2000 + 10 x Unit number

Outputs

The outputs consists of two transfer output words, which are used for data exchange from the CPU Unit to the MC Unit. More details can be found in the next section.

Output word	Bit	Name	Function
n	00 to 15	Output Word 1	First transfer output word.
n + 1	00 to 15	Output Word 2	Second transfer output word.

Inputs

The inputs consists of the status flags of the MC Unit and transfer input words. The transfer input data are two words, which are used for data exchange from the MC Unit to the CPU Unit. More details can be found in the next section.

Input word	Bit	Name	Function
n + 2	00	Unit Operating Flag	OFF: MC Unit is not operating. ON: MC Unit is operating.
	01	Servo Error Flag	OFF: No error. ON: A motion error has occurred.
	02	Task 1 Flag	OFF: Task1 is inactive. ON: Task 1 is active.
	03	Task 2 Flag	OFF: Task 2 is inactive. ON: Task 2 is active.
	04	Task 3 Flag	OFF: Task 3 is inactive. ON: Task 3 is active.
	05	Task 4 Flag	OFF: Task 4 is inactive. ON: Task 4 is active.
	06	Task 5 Flag	OFF: Task 5 is inactive. ON: Task 5 is active.
	07	PC Transfer Busy Flag	ON: The MC Unit is exchanging data with the PC Unit.
	08 to 15	Digital Input Status Flags	Indicate the status of digital inputs 0 to 7.
n + 3	00 to 15	Digital Input Status Flags	Indicate the status of the digital inputs 8 to 23.
n + 4	00 to 15	Digital Output Status Flags	Indicate the status of digital outputs 8 to 23.
n + 5	00	Axis 0 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 0.
	01	Axis 0 Forward Limit Flag	ON: A forward limit is set for axis 0.
	02	Axis 0 Reverse Limit Flag	ON: A reverse limit is set for axis 0.
	03	Axis 0 Origin Search Flag	ON: An origin search is in progress for axis 0.
	04	Axis 0 Feedhold Flag	ON: A feedhold is set for axis 0.
	05	Axis 0 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 0.
	06	Axis 0 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 0.
	07	Axis 0 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 0.
	08	Axis 1 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 1.
	09	Axis 1 Forward Limit Flag	ON: A forward limit is set for axis 1.
	10	Axis 1 Reverse Limit Flag	ON: A reverse limit is set for axis 1.
	11	Axis 1 Origin Search Flag	ON: An origin search is in progress for axis 1.
	12	Axis 1 Feedhold Flag	ON: A feedhold is set for axis 1.
	13	Axis 1 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 1.
	14	Axis 1 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 1.
	15	Axis 1 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 1.

Input word	Bit	Name	Function
n + 6	00	Axis 2 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 2.
	01	Axis 2 Forward Limit Flag	ON: A forward limit is set for axis 2.
	02	Axis 2 Reverse Limit Flag	ON: A reverse limit is set for axis 2.
	03	Axis 2 Origin Search Flag	ON: An origin search is in progress for axis 2.
	04	Axis 2 Feedhold Flag	ON: A feedhold is set for axis 2.
	05	Axis 2 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 2.
	06	Axis 2 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 2.
	07	Axis 2 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 2.
	08	Axis 3 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 3.
	09	Axis 3 Forward Limit Flag	ON: A forward limit is set for axis 3.
	10	Axis 3 Reverse Limit Flag	ON: A reverse limit is set for axis 3.
	11	Axis 3 Origin Search Flag	ON: An origin search is in progress for axis 3.
	12	Axis 3 Feedhold Flag	ON: A feedhold is set for axis 3.
	13	Axis 3 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 3.
	14	Axis 3 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 3.
	15	Axis 3 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 3.
n + 7	00	Task 1 BASIC Error Flag	ON: An error occurred in the BASIC program in task 1.
	01	Task 2 BASIC Error Flag	ON: An error occurred in the BASIC program in task 2.
	02	Task 3 BASIC Error Flag	ON: An error occurred in the BASIC program in task 3.
	03	Task 4 BASIC Error Flag	ON: An error occurred in the BASIC program in task 4.
	04	Task 5 BASIC Error Flag	ON: An error occurred in the BASIC program in task 5.
	05	Low Battery Flag	ON: The voltage of the backup battery is low.
	06	Not used	---
	07	PC Transfer Error Flag	ON: An error has occurred during data transfer between MC Unit and PC Unit.
	08 to 15	Indicator Mode	Contains the value of the DISPLAY system parameter, which determines the display mode of the bank of the 8 LED indicators on the front panel. Refer to *5-3-53 DISPLAY* for details.
n + 8	00 to 15	Input Word 1	First transfer input word.
n + 9	00 to 15	Input Word 2	Second transfer input word.

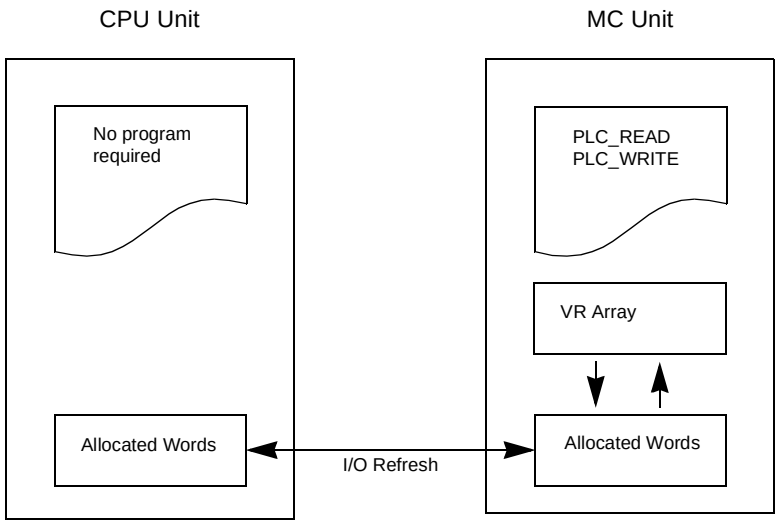
3-2 Overview of Data Exchanges

3-2-1 Data Exchange Methods

The MC Unit is able to exchange data with the CPU Unit the following three ways.

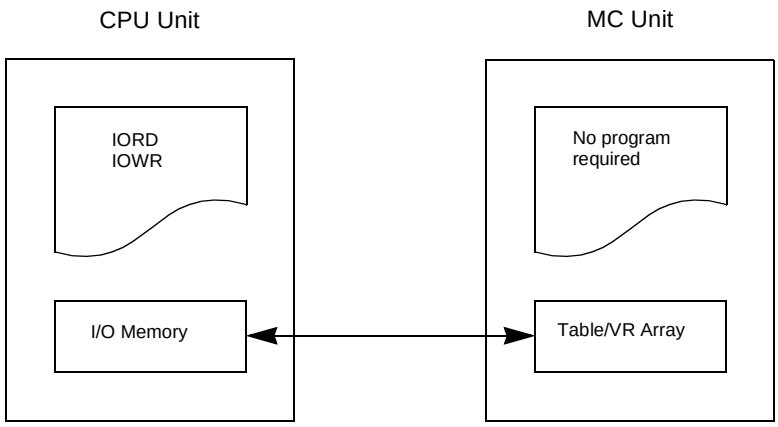
Data Words in IR/CIO Area

The MC Unit and the CPU Unit both have access to their own allocated words in memory. These transfer I/O words allocated in the MC Unit memory and in the CPU Unit's IR/CIO Area are exchanged during the I/O refresh period. The MC Unit accesses the words by using the BASIC commands PLC_READ and PLC_WRITE.



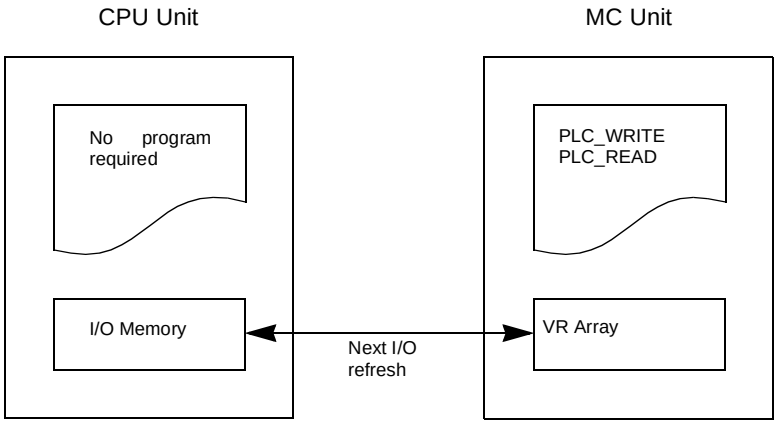
Data Transfer by CPU Unit

The CPU Unit is able to read/write directly into both the VR and Table memory areas of the MC Unit by using the ladder instructions IORD and IOWR. BASIC programming in the MC Unit is not required.



Data Transfer by MC Unit

The MC Unit can initiate data transfer to and from the CPU Unit using the PLC_READ and PLC_WRITE commands. The CPU Unit's user program is not required and the transfer will be executed at the next I/O refresh.



3-2-2 Data formats

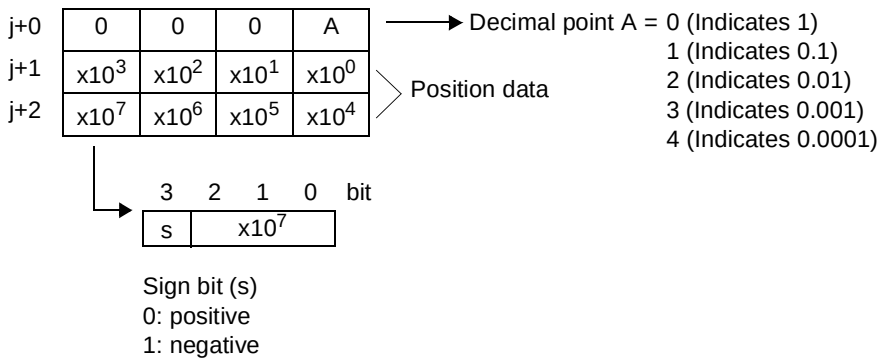
The data transfers commands of the PC Unit and the MC Unit support two data format types.

One-word format

The data is transferred word by word from each PC memory location to each variable in the MC unit and vice versa. The value in the MC Unit is always the integer equivalent of the hexadecimal value in the PC (no 2's complement). From the floating-point data in the MC unit only the integer part will be transferred. The valid range is [0,65535].

Three-word format

The data in the PC is represented by three memory elements, in total three words. The following is the configuration of a BCD position data item.



Example 1: The three-word format of value 56143 is given by

j+0	0	0	0	0
j+1	6	1	4	3
j+2	0	0	0	5

Example 2: The three-word format of value -48.89 is given by

j+0	0	0	0	2
j+1	4	8	8	9
j+2	8	0	0	0

One data item uses three words. Therefore the total words for data transfers should be the amount of data transferred multiplied by three.

3-3 Details of the Data Exchange Methods

3-3-1 Data Words in IR/CIO Area

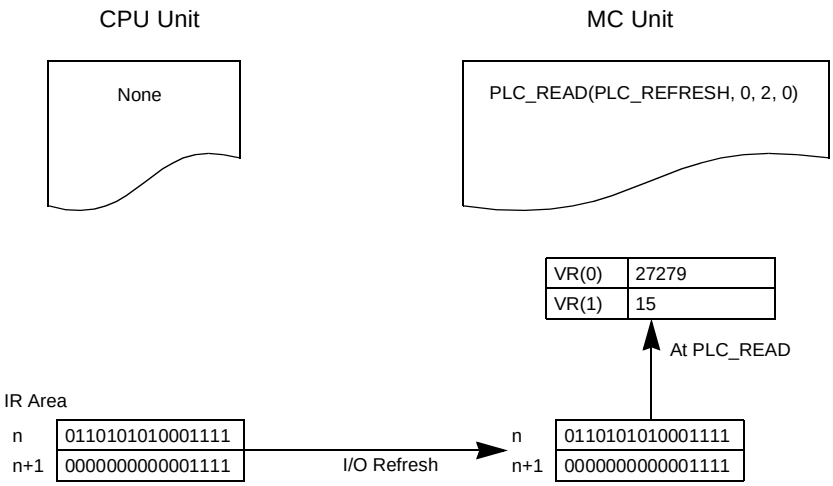
Data is read/written during the I/O refresh from either the MC Unit or the CPU Unit.

- The amount of data is two words output and two words input.
- The data is copied between the VR area and the allocated words in the MC Unit by using the PLC_READ and PLC_WRITE commands.

Transfer Output

The two words output data present in the CPU Unit is copied at every I/O refresh to the allocated words within the MC Unit. When a PLC_READ command is given, the contents of the allocated words (n, n+1) will be copied to the specified VR variables in the MC Unit.

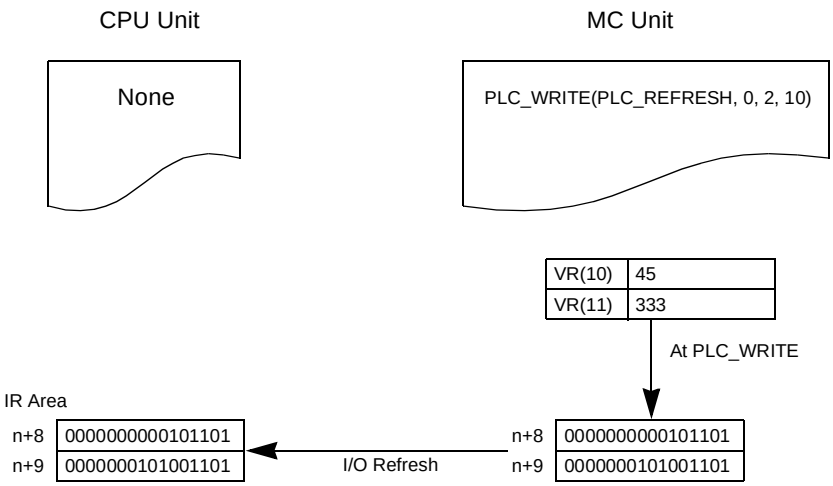
The following example is for a C200HX/HG/HE PC.



Transfer Input

After the PLC_WRITE command is given, the content of the specified VR variables will be copied to the allocated words in the MC Unit. On the next I/O refresh, this data will be copied to the allocated words (n+8, n+9) in the IR/CIO area of the CPU Unit.

The following example is for a C200HX/HG/HE PC.



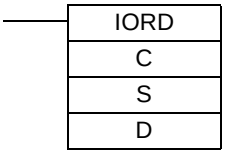
3-3-2 Data Transfer by CPU Unit

The CPU Unit is able to independently read and write to the MC Unit's Table and VR array.

- The maximum amount of data transferred is 128 words.
- The data transfer is synchronous with the ladder program.

IORD Instruction

The IORD instruction can be used in the following way to read data from the MC Unit. Refer to the Operation Manual of your PC Unit for further details on using this instruction.



C:

Value (hex)	Details
#A❑❑❑	VR (one-word format): Data is transferred from the first VR variable as specified by ❑❑❑. Range (BCD): 000 to 250.
#B❑❑❑	VR (three-word format): Data is transferred from the first VR variable as specified by ❑❑❑. Range (BCD): 000 to 250.
#❑❑❑❑	Table (three-word format): Data is transferred from the first Table variable as specified by ❑❑❑❑ multiplied by factor 10. Range (BCD): 0000 to 1599.

C200HX/HG/HE

S:

Value (hex)	Details
#n❑❑❑	Value n is the unit number of the MC Unit. Range: 0 to F. The amount of data is specified by ❑❑❑. Range: (BCD): 001 to 128.

CS1 Series

S: left most 4 digits
S+1: right most 4 digits

S:

Value (hex)	Details
#000n	Value n is the unit number of the MC Unit. Range: 0 to F.

S+1:

Value (hex)	Details
#❑❑❑❑	The no. of transfer words is specified by ❑❑❑❑. Range: 0000 to 0080 Hex.

D: First destination word in CPU Unit's memory.

Data

Item	Detail
PC	C200HX/HG/HE
Operand	D
IR Area 1	IR 000 to IR 235
SR Area 1	SR 236 to SR 252
SR Area 2	SR 256 to SR 299
IR Area 2	IR 300 to IR 511
HR Area	HR 00 to HR 99
AR Area	AR 00 to AR 27
LR Area	LR 00 to LR 63
TC Area	TC 000 to TC 511
TR Area	---
DM Area	DM 0000 to DM 6143
EM Area	---
Indirect DM addresses	*DM 0000 to DM 6655
Constants	---

Data

Item	Detail
PC	CS1
Operand	D
CIO Area	CIO 0000 to CIO 6143
Work Area	W000 to W511
Holding Bit Area	H000 to H511
Auxiliary Bit Area	A000 to A959
Timer Area	T0000 to T4095
Counter Area	C0000 to C4095
DM Area	D00000 to D32767
EM Area without bank	E00000 to E32767
EM Area with bank	En_00000 to En_32767 (n = 0 to C)
Indirect DM/EM addresses in binary	@D00000 to @D32767 @E00000 to @E32767 @En_00000 to @En_32767 (n = 0 to C)
Indirect DM/EM addresses in BCD	*D00000 to *D32767 *E00000 to *E32767 *En_00000 to *En_32767 (n = 0 to C)
Constants	---
Data Registers	---
Directly addressing Index Registers	---
Indirect addressing using Index Registers	,IR0 to ,IR15 -2048 to +2047 ,IR0 to ,IR15 DR0 to 15, IR0 to IR15 ,IR0 to ,IR15+(++) -(--) IR0 to IR15

Flags

Value (see note)	ON	OFF
Instruction Execution Error Flag (ER) (25503)	• The number of transfer words is not in BCD, or it is 0 words or is greater than 128 words. • The indirectly addressed DM address is greater than 6656 or not BCD. • The destination Unit No. is outside the range 0 to F, or is on a SYSMAC BUS slave rack. • The number of transfer words is not BCD.	C., S., and D. settings are correct.
Carry Flag (CY) (25504)	---	---
Greater Than Flag (GR) (25505)	---	---
Equals Flag (EQ) (25506)	Reading was correctly completed.	Reading was not correctly completed.
Less Than Flag (LE) (25507)	---	---
Overflow Flag (OF) (25404)	---	---
Underflow Flag (UF) (25405)	---	---
Negative Flag (N) (25402)	---	---

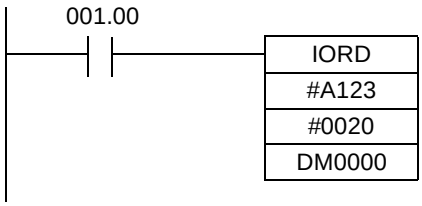
Note SR Area addresses for the C200HX/HG/HE, C200H, and C200HS are given in parentheses.

Value	ON	OFF
PC Transfer Error Flag (IR n+7 bit 07)	• The control code is not valid for the MC Unit. • The amount of words is not a multiple of three for three-word format transfer. • The MC Unit's Table or VR address in combination with the amount of data is invalid. • There is an overflow of IORD/IOWR and PLC_READ/PLC_WRITE transfers.	None of the errors has occurred.

Note The user should be aware that the MC Unit does not check if the MC Unit data is in the range of the three-word format.

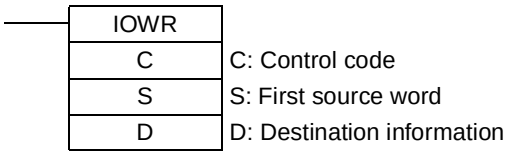
Transfer example for C200HX/HG/HE PC

In the following example, 20 words from VR(123) to VR(142) is transferred from the MC Unit with unit number set to 0 to addresses DM0000 to DM0019 in one-word format.



IOWR Instruction

The IOWR instruction can be used in the following way to write data to the MC Unit. Refer to the Operation Manual of your PC Unit for further details on using this instruction.



C:

Value (hex)	Details
#A❑❑❑	VR (one-word format): Data is transferred to the first VR variable as specified by ❑❑❑. Range (BCD): 000 to 250.
#B❑❑❑	VR (three-word format): Data is transferred to the first VR variable as specified by ❑❑❑. Range (BCD): 000 to 250.
#❑❑❑❑	Table (three-word format): Data is transferred to the first Table variable as specified by ❑❑❑❑ multiplied by factor 10. Range (BCD): 0000 to 1599.

S: First source word in CPU Unit's memory.

C200HX/HG/HE

D:

Value (hex)	Details
#n❑❑❑	Value n is the unit number of the MC Unit. Range: 0 to F. The amount of data is specified by ❑❑❑. Range: (BCD): 001 to 128.

CS1 Series

D: left most 4 digits
D+1: right most 4 digits

D:

Value (hex)	Details
#000n	Value n is the unit number of the MC Unit. Range: 0 to F.

D+1:

Value (hex)	Details
#❑❑❑❑	The no. of transfer words is specified by ❑❑❑❑. Range: 0000 to 0080 Hex.

Data

Item	Detail
PC	C200HX/HG/HE
Operand	S
IR Area 1	IR 000 to IR 235
SR Area 1	SR 236 to SR 255
SR Area 2	SR 256 to SR 299
IR Area 2	IR 300 to IR 511
HR Area	HR 00 to HR 99
AR Area	AR 00 to AR 27
LR Area	LR 00 to LR 63
TC Area	TC 000 to TC 511
TR Area	---
DM Area	DM 0000 to DM 6655
EM Area	---
Indirect DM addresses	*DM 0000 to DM 6655
Constants	---

Data

Item	Detail
PC	CS1
Operand	S
CIO Area	CIO 0000 to CIO 6143
Work Area	W000 to W511
Holding Bit Area	H000 to H511
Auxiliary Bit Area	A000 to A959
Timer Area	T0000 to T4095
Counter Area	C0000 to C4095
DM Area	D00000 to D32767
EM Area without bank	E00000 to E32767
EM Area with bank	En_00000 to En_32767 (n = 0 to C)
Indirect DM/EM addresses in binary	@D00000 to @D32767 @E00000 to @E32767 @En_00000 to @En_32767 (n = 0 to C)
Indirect DM/EM addresses in BCD	*D00000 to *D32767 *E00000 to *E32767 *En_00000 to *En_32767 (n = 0 to C)
Constants	#0000 to #FFFF (binary)
Data Registers	---
Directly addressing Index Registers	---
Indirect addressing using Index Registers	,IR0 to ,IR15 -2048 to +2047 ,IR0 to ,IR15 DR0 to 15, IR0 to IR15 ,IR0 to ,IR15+(++) -(--) IR0 to IR15

Clearing PC Transfer Error in MC Unit

C:

Value (hex)	Details
#EC00	The PC transfer error flag in IR/CIO area is cleared.

S: Use any source address

C200HX/HG/HE

D:

Value (hex)	Details
#n001	Value n is the unit number of the MC Unit. The amount of data should always be 001.

CS1 Series

D:

Value (hex)	Details
#000n	Value n is the unit number of the MC Unit. Range: 0 to F.

D+1:

Value (hex)	Details
#0001	The amount of data should always be 0001.

Flags

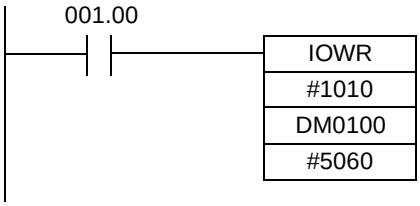
Value (see note)	ON	OFF
Instruction Execution Error Flag (ER) (25503)	• The number of transfer words is not in BCD, or it is 0 words or is greater than 128 words. • The indirectly addressed DM address is greater than 6656 or not BCD. • The Unit No. of the MC Unit is outside the range 0 to F, or is on a SYSMAC BUS slave rack. • The number of transfer words is not BCD. • The instruction was not correctly completed.	C., S., and D. settings are correct.
Carry Flag (CY) (25504)	---	---
Greater Than Flag (GR) (25505)	---	---
Equals Flag (EQ) (25506)	Writing was correctly completed.	Writing was not correctly completed.
Less Than Flag (LE) (25507)	---	---
Overflow Flag (OF) (25404)	---	---
Underflow Flag (UF) (25405)	---	---
Negative Flag (N) (25402)	---	---

Note SR Area addresses for the C200HX/HG/HE, C200H, and C200HS are given in parentheses.

Value	ON	OFF
PC Transfer Error Flag (IR n+7 bit 07)	• The control code is not valid for the MC Unit. • The amount of words is not a multiple of three for three-word format transfer. • The MC Unit's Table or VR address in combination with the amount of data is invalid. • There is an overflow of IORD/IOWR and PLC_READ/PLC_WRITE transfers.	None of the errors has occurred.

Note The user should be aware that the MC Unit does not check if the PC memory data complies to the three-word format.

Transfer example In the following example, 60 words from DM0100 to DM0159 is transferred in three words format from the PLC Unit to TABLE(10100) to TABLE(10119) of the MC Unit with unit number set to 5.



3-3-3 Data Transfer by MC Unit

The MC Unit is able to independently read and write from the MC Unit's VR array to the CPU Unit's I/O memory by using the PLC_READ and PLC_WRITE commands.

- The maximum amount of data transferred is 127 words.
- The commands only support the one-word format.
- The data is transferred at the end of the CPU scan cycle during I/O refresh.
- The BASIC program will be paused until completion of the transfer.

Refer to the *5-3-122 PLC_READ* and *5-3-124 PLC_WRITE* for more details on the commands.

PLC_READ Command

The following BASIC program will read 50 words from the CPU Unit's DM 0 to DM 49 to VR variables 100 to 149 in one-word format. A PC user program is not required.

BASIC program:

```
PLC_READ(PLC_DM,0,50,100)
```

PLC_WRITE Command

The following BASIC program will write 50 words from the MC Unit's VR variables 0 to 99 from the CPU Unit's EM 200 to EM 299 to in one-word format. A PC user program is not required.

BASIC program:

```
PLC_WRITE(PLC_EM,200,100,0)
```

Restrictions for CS1-series PCs

When the MC Unit is used with a CS1-series PC, there are some addressing restrictions for the data transfer using the PLC_READ and PLC_WRITE commands. Note that the other two methods do not have these restrictions and should be used for accessing data outside this range.

The addresses shown in the following table can be specified for PLC_READ and PLC_WRITE.

Area	CS1 address	Designation method	
		PC_area	First word address
DM area	D00000 to D06655	PLC_DM	0 to 6655
CIO area	CIO 0000 to CIO 0511	PLC_IR	0 to 511
Data Link area (in CIO area)	CIO 1000 to CIO 1063	PLC_LR	0 to 63
Holding Bit area, part 1	H000 to H099	PLC_HR	0 to 99
Holding Bit area, part 2	H100 to H127 (except H101)	PLC_AR	0 to 27
Timer area	T0000 to T0511	PLC_TC	0 to 511
EM area, bank 0	E0_00000 to E0_06143	PLC_EM	0 to 6143

SECTION 4
Multitasking BASIC Programming

This section gives an overview of the fundamentals of multitasking BASIC programs and the methods by which programs are managed for the MC Unit.

4-1 Overview 58
4-2 BASIC Programming 58
4-2-1 Axis, System and Task Statements 58
4-2-2 Data Structures and Variables 59
4-2-3 Mathematical Specifications 60
4-3 Motion Control Application 61
4-4 Command Line Interface 65
4-5 BASIC Programs 65
4-5-1 Managing Programs 65
4-5-2 Program Compilation 66
4-5-3 Program Execution 66
4-6 Error Processing 68

4-1 Overview

The C200HW-MC402 Motion Control Unit features a multitasking version of the BASIC programming language. The motion control language is largely based upon a tokenised BASIC and the programs are compiled into the tokenised form prior to their execution.

Multitasking is simple to set up and use and allows very complex machines to be programmed. Multitasking gives the MC Unit a significant advantage over equivalent single task systems. It allows modular applications where the logically connected processes can be grouped together in the same task program, thus simplifying the code architecture and design.

The MC Unit can hold up to 14 programs if memory size permits. A total of 5 tasks can be allocated to the programs. The execution of the programs is user controlled using BASIC.

The BASIC commands, functions and parameters presented here can be found in *SECTION 5 BASIC Motion Control Programming Language*.

4-2 BASIC Programming

The BASIC language consists among others of commands, functions and parameters. These BASIC statements are the building blocks provided to control the MC Unit operation.

Commands

Commands are words recognized by the processor that perform a certain action but do not return a value. For example, PRINT is a recognized word that will cause the value of the following functions or variables to be printed on a certain output device.

Functions

Functions are words recognized by the processor that perform a certain action and return a value related to that action. For example, ABS will take the value of its parameter and return the absolute value of it to be used by some other function or command. For example ABS(-1) will return the value 1, which can be used by the PRINT command, for example, to generate a string to be output to a certain device.

Parameters

Parameters are words recognized by the processor that contain a certain value. This value can be read and, if not read only, written. Parameters are used to determine and monitor the behavior of the system. For example, ACCEL determines the acceleration rate of a movement for a certain axis.

4-2-1 Axis, System and Task Statements

The commands, functions and parameters apply either to (one of) the axes, the tasks running or the general system.

Axis Statements

The motion control commands and the axis parameters apply to one or more axes. Axis parameters determine and monitor how an axis reacts on commands given and how it reacts to the outside world. Every axis has a set of parameters, so that all axes can work independently of each other. The motion control commands are able to control one or more of the axes simultaneously, while every axis has its own behavior.

The axis parameters are reset to their default values either when the power to the MC Unit is turned ON, the MC Unit is restarted from Motion Perfect, the MC Unit is restarted using the Restart Bit in the CPU Unit or the INITIALISE command is executed.

The commands and parameters work on some base axis or group of axes, specified by the BASE command. The BASE command is used to change this base axis group and every task has its own group which can be changed at any time. The default base axis is 0.

Individual axis dependent commands or parameters can also be programmed to work on a temporary base axis by including the AXIS function as a modifier in the axis dependent command. A temporary base axis is effective only for the command or parameter after which AXIS appears.

Task Statements The task parameters apply to a single task. The task parameters monitor the task for example for error handling. The PROC modifier allows the user to access a parameter of a certain task. Without PROC the current task is assumed. The BASE command (see above) is task specific and can be used with the PROC modifier.

System Statements These statements govern the overall system features, which are basically all statements which do not belong to the first two groups.

4-2-2 Data Structures and Variables

BASIC programs can store numerical data in various types of variables. Some variables have predefined functions, such as the axis parameters and system parameters; other variables are available for the programmer to define as required in programming. The MC Unit's Table, global and local variables are explained in this section. Furthermore also the use of labels will be specified.

Table

The Table is an array structure that contains a series of numbers. These numbers are used for instance to specify positions in the profile for a CAM or CAMBOX command. They can also be used to store data for later use, for example to store the parameters used to define a workpiece to be processed.

The Table is common to all tasks on the MC Unit, i.e., the values written to the Table from one task can be read from other tasks. The Table is backed up by a battery and will maintain its contents when power is turned OFF.

Table values can be written and read using the TABLE command. The maximum length of the array is 16000 elements, from TABLE(0) to TABLE(15999). The Table array is initialized up to the highest defined element.

Global Variables

The global variables, also called VR variables, are common to all tasks on the MC Unit. This means that if a program running on task 2 sets VR(25) to a certain value, then any other program running on a different task can read that same value from VR(25). This is very useful for synchronizing two or more tasks, but care must be taken to avoid more than one program writing to the same variable at the same time. The controller has 251 global variables, VR(0) to VR(250). The variables are read and written using the VR command. The VR variables maintain their values when power is turned OFF to the MC Unit. They are stored in RAM backed up by battery in the MC Unit.

⚠ **Caution** If the voltage of the backup battery drops, Table and global data will be lost. This can happen when the power to the MC Unit is turned OFF for a long period of time. The user should be very aware of this and should take the following precautions:

- Initialize variables from a program at power up as much as possible.
- Store dynamic application data, which can not be defined in programs, in the PC Unit's memory as much as possible.
- Update the data from the PC Unit at each power up before operation.

The Low Battery flag will turn ON when the voltage of the backup battery has dropped. Also the BATTERY_LOW system parameter will become TRUE. Refer to *3-1-2 Overview of IR/CIO Area Allocations* and *5-3-28 BATTERY_LOW* for detailed information.

Local Variables

Named variables or local variables can be declared in programming and are local to the task. This means that two or more programs running on different tasks can use the same variable name, but their values can be different. Local variables cannot be read from any task except for the one in which they are declared. Local variables are always cleared when a program is started. The local variables can be cleared by using either the CLEAR or the RESET command. Undefined local variables will return zero. Local variables cannot be declared on the command line.

A maximum of 255 local variables can be declared. Only the first 16 characters of the name are significant.

Labels

BASIC programs are normally executed in descending order through the lines. Labels can be used to alter this execution flow using the BASIC commands GOTO and GOSUB. To define a label it must appear as the first statement on a line and it must be ended by a colon (:). Labels can be character strings of any length, but only the first 15 characters are significant.

Using Variables and Labels

Each task has its own local labels and local variables. For example, consider the two programs shown below:

```
start:
   FOR a = 1 to 100
      MOVE(a)
      WAIT IDLE
   NEXT a
   GOTO start
```

```
start:
   a=0
   REPEAT
      a = a + 1
      PRINT a
   UNTIL a = 300
   GOTO start
```

These two programs when run simultaneously in different tasks and have their own version of variable “a” and label “start”. Note that undefined local variables will also return zero and not generate an error message.

If you need to hold data in common between two or more programs, VR variables should be used, or alternatively, if a large amount of data is to be held, the Table can be used.

To make a program more readable when using a VR variable, a named local variable can be used as a constant in the VR variable. The constant, however, must be declared in each program using the variable. In the example below, VR(3) is used to hold a length parameter.

```
start:
   GOSUB initial
   VR(length) = x


   ...Body of program

initial:
   length = 3
   RETURN
```

```
start:
   GOSUB initial
   MOVE(VR(length))
   PRINT VR(length)


   ...Body of program

initial:
   length = 3
   RETURN
```

4-2-3 Mathematical Specifications

Number format

The MC Unit has two main formats for numeric values: single precision floating point and single precision integer.

The single precision floating point format is internally a 32 bit value. It has an 8 bit exponent field, a sign bit and 23 bit fraction field with an implicit 1 as the 24th bit. Floating point numbers have a valid range of $\pm 5.9 \cdot 10^{-39}$ to $\pm 3.4 \cdot 10^{38}$.

Integers are essentially floating point numbers with a zero exponent. This implies that the integers are 24 bits wide. The integer range is therefore given from -16777216 to 16777215. Numeric values outside this range will be floating point.

⚠WARNING All mathematical calculations are done in floating point format. This implies that for calculations of/with larger values the results may have limited accuracy. The user should be aware of this when developing the motion control application.

Positioning

For positioning, the Unit will round up if the fractional encoder edge distance calculated exceeds 0.9. Otherwise the fractional value will be rounded down.

Floating point comparison

The comparison functions considers small difference between values as equal to avoid unexpected comparison results. Therefore any two values for which the difference is less than $1.19 \cdot 10^{-6}$ are considered equal.

Precedence

The precedence of the operators is given below:

Unary Minus, NOT
^
/ *
MOD
+ -
= <> > >= <= <
AND OR XOR
Left to Right

The best way to ensure the precedence of various operators is through the use of parentheses.

4-3 Motion Control Application

Initialisation

For setting up a motion application with the MC Unit, the following parameters need to be considered.

Parameter	Description
WDOG	The WDOG parameter contains the software switch used to control the enable relay contact, which enables all drivers.
SERVO	The SERVO parameter determines whether the base axis runs under servo control (ON) or open loop (OFF). When in open loop the output speed reference voltage is determined by the DAC parameter.
DAC	The DAC parameter contains the voltage value which is applied directly to the Servo Driver when the base axis is in open loop.
P_GAIN	The P_GAIN parameter contains the proportional gain for the axis.
I_GAIN	The I_GAIN parameter contains the integral gain for the axis.
D_GAIN	The D_GAIN parameter contains the derivative gain for the axis.
VFF_GAIN	The VFF_GAIN parameter contains the speed feed forward gain for the axis.
OV_GAIN	The OV_GAIN parameter contains the output speed gain for the axis.

In the following example a simple motion application including initialisation for a single axis is shown.

```
init:
      BASE(0)
      P_GAIN=.5: I_GAIN=0: D_GAIN=0
      VFF_GAIN=0: OV_GAIN=0
      ACCEL=1000
      DECEL=1000
      SPEED=500
      WDOG=ON
      SERVO=ON
loop:
      MOVE(500)
      WAIT IDLE
      WA(250)
      MOVE(-500)
      WAIT IDLE
      WA(250)
      GOTO loop
```

Move Execution

Every task on the MC Unit has a set of buffers that holds the information from the motion commands given. The motion commands include MOVE, MOVEABS, MOVEMODIFY, MOVECIRC, MHELICAL, FORWARD, REVERSE, MOVELINK, CONNECT, CAM and CAMBOX. Refer to *5-2-1 Motion Control Commands* for details on specific commands.

Motion Generator

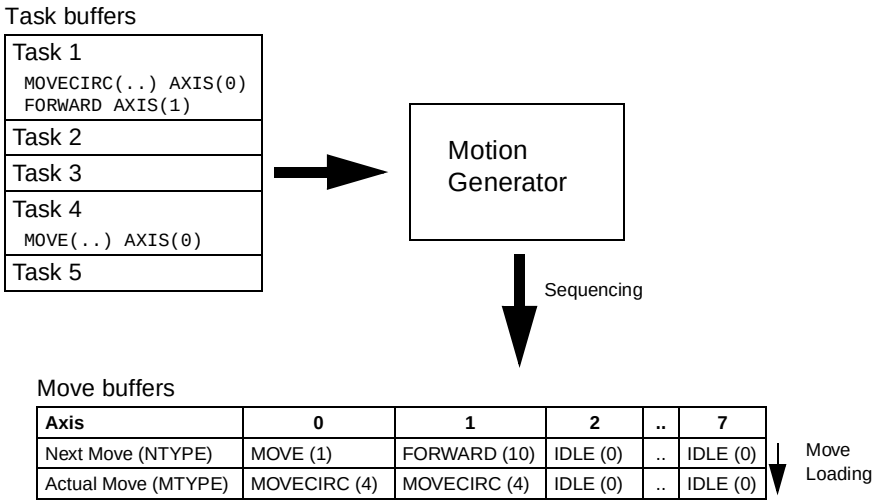
The motion generator, a background process that prepares and runs moves, has a set of two motion buffers for each axis. One buffer holds the Actual Move, which is the move currently executing on the axis. The MTYPE axis parameter contains the identity number of this move. For example the MTYPE will have value 10 if currently the FORWARD move is executed. The other buffer holds the Next Move, which is executed after the Actual Move has finished. The NTYPE axis parameter contains the identity number of this next move.

The BASIC programs are separate from the motion generator program, which controls moves for the axes. The motion generator has separate functions for each axis, so each axis is capable of being programmed with its own axis parameters (for example speed, acceleration) and moving independently and simultaneously or they can be linked together using special commands.

When a move command is processed, the motion generator waits until the move buffers for the required axes are empty and then loads these buffers with the move information.

Note If the task buffers are full, the program execution is paused until buffers are available again. This also applies to the command line task and no commands can be given for that period. Motion Perfect will disconnect in such a

case. The PMOVE task parameter will be set to TRUE when the task buffers are full and will be reset to FALSE when the task buffers are available again.



Sequencing

On each servo interrupt every millisecond (see *4-5-3 Program Execution*), the motion generator examines the NTYPE buffers to see if any of them are available. If there are any available then it checks the task buffers to see if there is a move waiting to be loaded. If a move can be loaded, then the data for all the specified axes is loaded from the task buffers into the NTYPE buffers and the corresponding task buffers are marked as idle. This process is called sequencing.

Move Loading

Once sequencing has been completed, the MTYPE buffers are checked to see if any moves can be loaded. If the required MTYPE buffers are available, then the move is loaded from the NTYPE buffers to the MTYPE buffers and the NTYPE buffers are marked as idle. This process is called move loading.

If there is a valid move in the MTYPE buffers, then it is processed. When the move has been completed, the MTYPE buffers are marked as idle.

Accessing I/O

The MC Unit has three different types of I/O. These are the physical I/O, the driver I/O and the virtual I/O. The inputs and outputs are accessible by using the IN and OP commands in BASIC and within Motion Perfect using the I/O status window. Refer to *5-3-83 IN*, *5-3-115 OP* and *6-6-6 I/O Status Window* for further details.

The different types of inputs are explained here.

Input type	Range (amount)	Description
Physical	0 - 15 (16)	The physical inputs are freely allocable to the different functions. Some of the functions are origin search, limit switches, jog inputs and so on. The MC Unit uses axis parameters to allocate a certain function to an input. The first four inputs R0 to R3 are used as registration inputs for axis 0 to 3. These inputs can also be used for any other purpose. The related BASIC command and axis parameters are REGIST Registration Command DATUM_IN Selection of origin switch input FAST_JOG Selection of fast jog input FHOLD_IN Selection of feedhold input FWD_IN Selection of forward limit input FWD_JOG Selection of forward jog input REV_IN Selection of reverse limit input REV_JOG Selection of reverse jog input
Driver	16 - 19 (4)	The four driver inputs nr. 16 to 19 correspond to the four alarm inputs from the drivers of axis 0 to 3.
Virtual	20 - 31 (12)	The virtual inputs are only present inside the MC Unit and are used for computational purposes only. The virtual inputs and outputs are bi-directional. The inputs are controlled by the outputs. All functions which can be used on physical inputs can also by these used for these virtual inputs.

The different types of outputs are explained here.

Output type	Range (amount)	Description
Physical	8 - 15 (8)	The physical outputs are freely allocable to any user defined functions. An output can be set and reset depending on the current axis position by using the command PSWITCH. Note that the physical output connections O0 to O7 are corresponding to the internal outputs 8 to 15.
Driver	16 (1)	The single driver output is the driver alarm reset for all drivers.
Virtual	20 - 31 (12)	The virtual outputs are only present inside the MC Unit and are used for computational purposes only. The virtual inputs and outputs are bi-directional. The inputs are controlled by the outputs.

4-4 Command Line Interface

The Command Line Interface provides a direct interface for the user to give commands and access parameters on the system. There are two options to use the command line interface:

- Use the Terminal Window within Motion Perfect and the MC Unit connected. See *SECTION 6 Programming Environment* for details.
- Use a VT100 Terminal to connect to the MC Unit. This is similar to using the Terminal Window within Motion Perfect when the MC Unit is disconnected.

The MC Unit puts the last 10 commands given on the command line in a buffer. Pressing the Up and Down Cursor Key will cycle through the buffer to execute the command again.

4-5 BASIC Programs

The MC Unit can store up to 14 programs in memory, provided the capacity of memory is not exceeded. The MC Unit supports simple file-handling instructions for managing these program files rather like the DOS filing system on a computer.

The Motion Perfect software package is used to store and load programs to and from a computer for archiving, printing and editing. It also has several controller monitor and debugging facilities. Refer to *SECTION 6 Programming Environment* for details on Motion Perfect.

4-5-1 Managing Programs

Motion Perfect automatically creates a project which contains the programs to be used for an application. The programs of the project are kept both in the controller as on the computer. Whenever a program is created or edited, Motion Perfect edits both copies in order to always have an accurate backup outside the controller at any time. Motion Perfect checks that the two versions of the project are identical using a cyclic redundancy check. If the two differ, Motion Perfect allows copying the MC Unit version to disk or vice versa.

Programs on the computer are stored in ASCII text files. They may therefore be printed, edited and copied using a simple text editor. The source programs are held in the MC Unit in a tokenised form and as a result, the sizes of the programs will be less on the MC Unit compared to the same programs on the computer.

Storing Programs

Programs on the MC Unit are held in battery-backed RAM or flash EPROM when power is turned OFF. At start-up before operation either the programs present in RAM are used or the programs in flash EPROM will be copied first to RAM. These two options are selectable by using the POWER_UP system parameter.

The current programs in RAM can be copied to flash EPROM by using the EPROM command. Both the POWER_UP parameter as the EPROM command are also provided by Motion Perfect with buttons on the control panel and commands under the Program and Controller menus.

Note After development of the application programs, be sure to save the data in flash memory within the MC Unit. The data will remain in the S-RAM during operation and power down, but considering possible battery failure it is advised to store the data in flash memory.

Program Commands

The MC Unit has a number of BASIC commands to allow creation, manipulation and deletion of programs. Motion Perfect provides buttons which also perform these operations.

Command	Function
SELECT	Selects a program for editing, deleting etc.
NEW	Deletes the current selected program, a specified program or all programs.
DIR	Lists the directory of all programs.
COPY	Duplicates a specified program.
RENAME	Renames a specified program.
DEL	Deletes the current selected program or a specified program.
LIST	Lists the current selected program or a specified program.

4-5-2 Program Compilation

The MC Unit system compiles programs automatically when required. It is not normally required to force the MC Unit to compile programs, but programs can be compiled under the Program Menu in Motion Perfect.

The MC Unit automatically compiles programs at the following times.

- The selected program is compiled before it is executed if it has been edited.
- The selected program is compiled if it has been edited before switching the selected program to another program.
- The selected program is compiled by using the COMPILE command.

The program syntax and structure are checked during compilation. If compilation is unsuccessful, a message will be provided and no program code will be generated. A red cross will appear in the Motion Perfect directory box.

Programs cannot be run when compilation errors occur. The errors should be corrected and the program recompiled.

The compilation process also includes the following:

- Removing comments.
- Compiling numbers into the internal processor format.
- Converting expressions into reverse Polish Notation format for execution.
- Precalculating variable locations.
- Calculating and embedding loop structure destinations.

4-5-3 Program Execution

The timing of the execution for the different tasks and the refreshing of the I/O of the Unit revolves around the servo period of the system. For the MC Unit the servo period is set to 1 ms. The servo period is not synchronised with the PC scan time.

I/O Refresh

The I/O status of the MC Unit is refreshed at the beginning of every servo cycle.

- The captured status of the digital inputs is transferred to the IN system input variable. Note that this is the status captured in the previous servo cycle.
- The analogue outputs for the speed references are updated.
- The digital outputs are updated conform the status of the OP system output variable.
- The status of the digital inputs is captured.

Note that no automatic processing of the I/O signals is taking place, except for registration. This implies that all actions must be programmed in the BASIC programs.

Program Tasks

This servo period is split into three equal segments. These three slots are partly taken up by respectively servo control, the background communications and the basic house keeping tasks. The remaining period in each of the time slots is available for the BASIC tasks.

The multi-tasking executive operates as follows. There are 6 tasks available for execution. Tasks 1 to 5 are user tasks, which are used to run BASIC programs simultaneously, one program per task. Each program can be allocated to a specific task or priority, which implies more or less execution time (see Program Execution Priority below). It is important to know that the BASIC program execution is not synchronised with the servo cycle. The command line always uses the system task 0. This task is used for the raw terminal communications and Motion Perfect communications.

Relevant commands

Motion Perfect provides several ways of executing, pausing and stopping the programs using buttons on the control panel and the editing windows. The following commands can be given on the command line to control the execution.

Command	Function
RUN	Run the current selected program or a specified program, optionally on a specified task number.
STOP	Stop the current selected program or a specified program.
HALT	Stop all programs on the system
PROCESS	Displays all running tasks.

The user can explicitly allocate the task priority on which the BASIC program is expected to run. When a user program is run without explicit task allocation, it is assigned the highest available task priority. Tasks 5 and 4 have high priority and tasks 3,2 and 1 have low priority.

Setting Programs to Run at Start-up

Programs can be set to run automatically at different priorities when power is turned ON. If required, the computer can be left connected as an operator interface or may be removed and the programs run “stand-alone”.

Programs are set in Motion Perfect to run automatically at start-up using the Set Power Up Mode… selection under the Program Menu. This operation sets which program to run automatically and at which priority. This can also be accomplished by the RUNTYPE parameter. The current status can be seen with the DIR command.

Program Execution Priority

The programs given task no. 5 and 4 have high priority and programs given task 3, 2 and 1 have low priority. Task 0, which is the command line task, has also low priority. When both high and low priority tasks are running, the high priority tasks are divided over two slot and the low priority task are divided over one slot. If all tasks have the same priority the tasks will be divided equally over the three slots. Tasks with the same priority will be allocated to the slots in such a way that over multiple servo periods that they will have equal execution time.

Consider the following examples:

Example 1: Tasks 1 & 2 and command line running.

Servo Cycle (n)		
Servo Control	House Keeping	Communications
Task 0	Task 1	Task 2

The real-time executive operates in a round-robin schedule.

Example 2: Tasks 1, 2 & 3 and command line task 0 running

Servo Cycle (n)			Servo Cycle (n+1)		
Servo Control	House Keeping	Comms	Servo Control	House Keeping	Comms
Task 0	Task 1	Task 2	Task 3	Task 0	Task 1

The real-time executive allocates the time slots to tasks 0, 1, 2 & 3 in turn.

Example 3: Tasks 1 & 4 and command line task 0 running

Servo Cycle (n)			Servo Cycle (n+1)		
Servo Control	House Keeping	Comms	Servo Control	House Keeping	Comms
Task 4	Task 4	Task 0	Task 4	Task 4	Task 1

The real-time executive invokes task 4 as a high priority task every servo cycle. The remaining tasks fit in the remaining slot in turn.

Example 4: Tasks 1, 2, 4 & 5 and command line task 0 running

Servo Cycle (n)			Servo Cycle (n+1)		
Servo Control	House Keeping	Comms	Servo Control	House Keeping	Comms
Task 5	Task 4	Task 0	Task 5	Task 4	Task 1

Servo Cycle (n+2)		
Servo Control	House Keeping	Comms
Task 5	Task 4	Task 2

The real-time executive invokes task 5 and task 4 as a high priority tasks every servo cycle. Note that the high priority tasks take up both high priority slots. The remaining tasks fit in the remaining slot in turn. If task 3 was also added to above scenario it will be executed in the third slot alongside tasks 0, 1 & 2.

4-6 Error Processing

For the safety of the application it is very important that proper safety measures are taken for the different problems which may occur in the system. For safe operation at all times the user must make use of the several options to check for these errors in both the MC Unit or in the PC Unit.

It is advisable to have the master control of the application within the PC Unit, not the MC Unit. The PC Unit can monitor the status of the MC Unit and of the other Units, manage emergency shut-downs for the application, control the data flows from and to the MC Unit, and so on.

As for the MC Unit, the BASIC programming language provide the programmer with the freedom to include a lot of safety measures or not. This requires a sensible solution, which covers all possible behaviour of the system.

This section will present the possible errors that may occur and suggest the way to detect them. For a full description of the error handling refer to section *7-3 Error Handling* and for the IR/CIO area description section *3-1-2 Overview of IR/CIO Area Allocations*.

⚠ **Caution** The PC or MC Unit outputs may have undefined status due to deposits on or burning of the output relays, or destruction of the output transistors. As a counter-measure for such problems, external safety measures must be provided to ensure safety in the system.

BASIC Errors

If a BASIC error is generated during the execution of a BASIC command in some task, the program will be halted immediately or the user can define a specific error routine structure to stop the system. The error routine can stop the driver, put the digital I/O in a safe status and notify the PC Unit of the error. Please refer to section *5-3-27 BASICERROR* on the way to include an error subroutine in a BASIC program. The BASIC error is also indicated in the PC's IR/CIO area.

Motion Error

In case a motion error occurs, the MC Unit will disable the control of the driver automatically. The user has the ability to decide for each axis which motion errors will disable the driver by using the ERRORMASK parameter (see section *5-3-61 ERRORMASK*). After detection of the motion error the user is free to program the necessary countermeasures for the other axes and the complete system. The PC's IR/CIO area also indicates the axis on which the error has occurred and the type of error.

PC Interface Error

In case of an error in the PC transfer using IORD and IOWR instruction, the PC Interface Error flag (n+7 bit 07) in the IR/CIO area will be set. This bit can be checked in the PC program to deal with any unforeseen event for PC transfers.

Please refer to *Appendix B* to find an example of implementing the control of the application including error checking. Example no. 12 shows a master program which is dedicated to the control of the application by running the appropriate programs and is continuously checking for any error event which may occur. This program should be set to run at start-up of the MC Unit. It is strongly recommended to use this program or a similar program within every application.

SECTION 5
BASIC Motion Control Programming Language

This section describes the commands and parameters required for programing the motion control application using the MC Unit. All BASIC system, task and axis statements that determine the various aspects of program execution and MC Unit operation are presented.

5-1 Notation Used in this Section 75
5-2 Classifications and Outlines 75
5-2-1 Motion Control Commands 76
5-2-2 I/O Commands and Functions 77
5-2-3 Loop and Conditional Structures 77
5-2-4 Program Commands and Functions 78
5-2-5 System Commands and Parameters 78
5-2-6 Mathematical and Logical Functions 80
5-2-7 Constants. 81
5-2-8 Motion Perfect Commands, Functions and Parameters 81
5-2-9 Axis Parameters 81
5-2-10 Task Commands and Parameters 83
5-2-11 PC Data Exchange Commands. 83
5-3 Command, function and parameter description 84
5-3-1 Multiply: * 84
5-3-2 Power: ^ 84
5-3-3 Add: + ... 84
5-3-4 Subtract: –. 85
5-3-5 Divide: / 85
5-3-6 Is Less Than: < 85
5-3-7 Is Less Than Or Equal To: <=. 85
5-3-8 Is Not Equal To: <> 86
5-3-9 Is Equal To: = ... 86
5-3-10 Is Greater Than: >. 86
5-3-11 Is Greater Than or Equal To: >=. 86
5-3-12 Statement separator: “:” 87
5-3-13 Comment field: ‘. 87
5-3-14 ABS. 87
5-3-15 ACCEL 87
5-3-16 ACOS 87
5-3-17 ADDAX 88
5-3-18 AND 88
5-3-19 ASIN. 89
5-3-20 ATAN 89
5-3-21 ATAN2 89
5-3-22 ATYPE 90
5-3-23 AUTORUN. 90
5-3-24 AXIS. 90
5-3-25 AXISSTATUS. 91
5-3-26 BASE 91
5-3-27 BASICERROR 92
5-3-28 BATTERY_LOW 93
5-3-29 CAM. 93
5-3-30 CAMBOX. 94
5-3-31 CANCEL 95

5-3-32 CHECKSUM 96
5-3-33 CLEAR 96
5-3-34 CLEAR_BIT 96
5-3-35 CLOSE_WIN 96
5-3-36 COMMSERROR 97
5-3-37 COMPILE 97
5-3-38 CONNECT 97
5-3-39 CONTROL 98
5-3-40 COPY 98
5-3-41 COS 98
5-3-42 CREEP 99
5-3-43 D_GAIN 99
5-3-44 DAC 99
5-3-45 DAC_OUT 100
5-3-46 DATUM 100
5-3-47 DATUM_IN 101
5-3-48 DECEL 101
5-3-49 DEFPOS 101
5-3-50 DEL 102
5-3-51 DEMAND_EDGES 102
5-3-52 DIR 103
5-3-53 DISPLAY 103
5-3-54 DPOS 103
5-3-55 EDIT 104
5-3-56 ENCODER 104
5-3-57 ENDMOVE 104
5-3-58 EPROM 104
5-3-59 ERROR_AXIS 104
5-3-60 ERROR_LINE 105
5-3-61 ERRORMASK 105
5-3-62 EXP 105
5-3-63 FALSE 106
5-3-64 FAST_JOG 106
5-3-65 FE 106
5-3-66 FE_LIMIT 106
5-3-67 FE_RANGE 106
5-3-68 FHOLD_IN 107
5-3-69 FHSPEED 107
5-3-70 FOR TO STEP NEXT 107
5-3-71 FORWARD 108
5-3-72 FRAC 108
5-3-73 FREE 109
5-3-74 FS_LIMIT 109
5-3-75 FWD_IN 109
5-3-76 FWD_JOG 109
5-3-77 GET 109
5-3-78 GOSUB RETURN 110
5-3-79 GOTO 110
5-3-80 HALT 111
5-3-81 I_GAIN 111
5-3-82 IF THEN ELSE ENDIF 111
5-3-83 IN 112
5-3-84 INITIALISE 113

5-3-85 INPUT 113
5-3-86 INT 113
5-3-87 JOGSPEED 114
5-3-88 KEY 114
5-3-89 LAST_AXIS 114
5-3-90 LINPUT 115
5-3-91 LIST 115
5-3-92 LN 116
5-3-93 LOCK 116
5-3-94 MARK 116
5-3-95 MERGE 116
5-3-96 MHELICAL 117
5-3-97 MOD 118
5-3-98 MOTION_ERROR 118
5-3-99 MOVE 119
5-3-100 MOVEABS 120
5-3-101 MOVECIRC 121
5-3-102 MOVELINK 122
5-3-103 MOVEMODIFY 125
5-3-104 MPOS 125
5-3-105 MSPEED 125
5-3-106 MTYPE 125
5-3-107 NEW 126
5-3-108 NIO 126
5-3-109 NOT 126
5-3-110 NTYPE 127
5-3-111 OFF 127
5-3-112 OFFPOS 127
5-3-113 ON 127
5-3-114 ON 127
5-3-115 OP 128
5-3-116 OPEN_WIN 129
5-3-117 OR 129
5-3-118 OUTLIMIT 130
5-3-119 OV_GAIN 130
5-3-120 P_GAIN 130
5-3-121 PI 130
5-3-122 PLC_READ 131
5-3-123 PLC_TYPE 132
5-3-124 PLC_WRITE 132
5-3-125 PMOVE 133
5-3-126 POWER_UP 134
5-3-127 PP_STEP 134
5-3-128 PRINT 134
5-3-129 PROC 136
5-3-130 PROCESS 136
5-3-131 PROCNUMBER 136
5-3-132 PSWITCH 136
5-3-133 RAPIDSTOP 137
5-3-134 READ_BIT 138
5-3-135 REG_POS 138
5-3-136 REGIST 138

5-3-137 REMAIN 139
5-3-138 RENAME 140
5-3-139 REP_DIST 140
5-3-140 REP_OPTION 140
5-3-141 REPEAT UNTIL 141
5-3-142 RESET 141
5-3-143 REV_IN 142
5-3-144 REV_JOG 142
5-3-145 REVERSE 142
5-3-146 RS_LIMIT 142
5-3-147 RUN 142
5-3-148 RUN_ERROR 143
5-3-149 RUNTYPE 143
5-3-150 SCOPE 144
5-3-151 SCOPE_POS 145
5-3-152 SELECT 145
5-3-153 SERVO 145
5-3-154 SET_BIT 146
5-3-155 SETCOM 146
5-3-156 SGN 146
5-3-157 SIN 147
5-3-158 SPEED 147
5-3-159 SQR 147
5-3-160 SRAMP 147
5-3-161 STEPLINE 148
5-3-162 STOP 148
5-3-163 TABLE 148
5-3-164 TAN 149
5-3-165 TICKS 150
5-3-166 TRIGGER 150
5-3-167 TROFF 150
5-3-168 TRON 150
5-3-169 TRUE 151
5-3-170 TSIZE 151
5-3-171 UNITS 151
5-3-172 VERSION 152
5-3-173 VFF_GAIN 152
5-3-174 VP_SPEED 152
5-3-175 VR 152
5-3-176 WA 153
5-3-177 WAIT IDLE 154
5-3-178 WAIT LOADED 154
5-3-179 WAIT UNTIL 154
5-3-180 WDOG 155
5-3-181 WHILE WEND 155
5-3-182 XOR 156

5-1 Notation Used in this Section

This section describes the notation used in describing commands, functions, and parameters. The section name gives the name of the command, function or parameter. The descriptions within the section are divided into the following parts. Individual parts are omitted when they are not applicable.

Type: The classification is given for command, function or parameter.

Syntax: Standard BASIC notation is used to show command or function syntax.

Text that must be typed exactly as given is in `typewriter font`.

The names of arguments are given in italic font with *underscores_for_spaces*. Replace these with the actual arguments.

Items in square brackets "[ ]" in the syntax notation are optional and can be omitted. Repetition is denoted by "{ }" brackets. Items enclosed in these brackets are repeated zero or more times.

Alternative: Any alternative form of a command, function or parameter is given.

Description: This field describes the purpose and application of the command, function or parameter.

Precautions: Specific precautions related to programming are provided.

Arguments: The name of each argument is given in bold italic font followed by a description of the argument.

See also: In this field the related commands, functions and parameters are given.

Example: One or more application examples is given for most commands, functions, and parameters.

5-2 Classifications and Outlines

The table below describes the groups into which commands, functions, and parameters are divided.

Groups
Motion Control Commands
I/O Commands and Functions
Loop and Conditional Structures
Program Commands and Functions
System Commands and Parameters
Mathematical Functions
Constants
Motion Perfect Commands and Parameters
Axis Parameters
Task Functions and Parameters
PC Data Exchange Commands

The rest of the tables in the following sections outline the commands, functions and parameters used for the MC Unit.

5-2-1 Motion Control Commands

The table below outlines the motion control commands. Refer to the specified pages for details.

Name	Syntax	Description	Page
ADDAX	ADDAX(*axis*)	ADDAX sets a link to a superimposed axis. All demand position movements for the superimposed axis will be added to any moves that are currently being executed.	88
BASE	BASE(*axis_1*[,*axis_2*[, ...]]) BA(*axis_1*[,*axis_2*[, ...]])	BASE is used to set the base axis to the axis specified with axis.	91
CAM	CAM(*start_point,end_point, table_multiplier,distance*)	CAM moves an axis according to values of a movement profile stored in the Table array.	93
CAMBOX	CAMBOX(*start_point,end_point, table_multiplier,link_distance, link_axis*[,*link_option* [,*link_position*]])	CAMBOX moves an axis according to values of a movement profile stored in the Table array. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox.	94
CANCEL	CANCEL[(1)] CA[(1)]	CANCEL cancels the move on an axis.	95
CONNECT	CONNECT(*ratio*,*driving_axis*) CO(*ratio*,*driving_axis*)	CONNECT connects the demand position of an axis to the measured movements of the axis specified for *driving_axis* to produce an electronic gearbox.	97
DATUM	DATUM(*sequence*)	DATUM performs one of 7 origin search sequences to position an axis to an absolute position or DATUM reset following errors.	100
DEFPOS	DEFPOS(*pos_1*[,*pos_2*[, ...]]) DP(*pos_1*[,*pos_2*[, ...]])	DEFPOS defines the current position as a new absolute position.	101
FORWARD	FORWARD FO	FORWARD moves an axis continuously forward at the speed set in the SPEED parameter.	108
MHELICAL	MHELICAL(*end_1*,*end_2*, *centre_1*,*centre_2*,*direction*, *distance_3*) MH(*end_1*,*end_2*,*centre_1*, *centre_2*,*direction*,*distance_3*)	MHELICAL performs an interpolated helical move by moving 2 orthogonal axes in a circular arc with a simultaneous linear move on a third axis.	117
MOVE	MOVE(*dist_1*[,*dist_2*[, ...]]) MO(*dist_1*[,*dist_2*[, ...]])	MOVE moves one or more axes at the demand speed, acceleration and deceleration to the position specified as increment from the current position.	119
MOVEABS	MOVEABS(*pos_1*[,*pos_2*[, ...]]) MA(*pos_1*[,*pos_2*[, ...]])	MOVEABS moves one or more axes at the demand speed, acceleration and deceleration to the position specified as absolute position, i.e., in reference to the origin.	120
MOVECIRC	MOVECIRC(*end_1*,*end_2*, *centre_1*,*centre_2*,*direction*) MC(*end_1*,*end_2*,*centre_1*, *centre_2*,*direction*)	MOVECIRC interpolates 2 orthogonal axes in a circular arc.	121
MOVELINK	MOVELINK(*distance*, *link_distance*,*link_acceleration*, *link_deceleration*,*link_axis* [,*link_option*[,*link_position*]]) ML(*distance*,*link_distance*, *link_acceration*,*link_deceleration*, *link_axis*[,*link_option* [,*link_position*]])	MOVELINK creates a linear move on the base axis linked via a software gearbox to the measured position of a link axis.	122
MOVEMODIFY	MOVEMODIFY(*position*) MM(*position*)	MOVEMODIFY changes the absolute end position of the current single-axis linear move (MOVE or MOVEABS).	125

Name	Syntax	Description	Page
RAPIDSTOP	RAPIDSTOP RS	RAPIDSTOP cancels the current move on all axes.	137
REVERSE	REVERSE	REVERSE moves an axis continuously in reverse at the speed set in the SPEED parameter.	142

5-2-2 I/O Commands and Functions

The table below outlines the I/O commands and functions. Refer to the specified pages for details.

Name	Syntax	Description	Page
GET	GET[*#n,*]*variable*	GET waits for the arrival of a single character and assigns the ASCII code of the character to *variable*.	109
IN	IN(*input_number* [,*final_input_number*])	IN returns the value of digital inputs.	112
INPUT	INPUT[*#n*,] variable{, ... }	INPUT waits for a string to be received and assigns the numerical value to *variable*.	113
KEY	KEY[*#n*]	KEY returns TRUE/FALSE depending on character received.	114
LINPUT	LINPUT[*#n*,] *vr_variable*	LINPUT waits for a string and puts it in VR variables.	115
OP	OP(*output_number*, *value*) OP(*binary_pattern*) OP	OP sets one or more outputs or returns the state of the first 24 outputs.	128
PRINT	PRINT[*#n*,] *expression*{, ... } ?[*#n*,] *expression*{, ... }	PRINT outputs a series of characters to a serial port.	134
PSWITCH	PSWITCH(*switch*,*enable*[,*axis*, *output_number*,*output_state*, *set_position*,*reset_position*])	PSWITCH turns ON an output when a predefined position is reached, and turns OFF the output when a second position is reached.	136
REGIST	REGIST(*mode*)	REGIST captures an axis position when a registration input or the Z mark on the encoder is detected.	138
SETCOM	SETCOM(*baud_rate*,*data_bits*, *stop_bits*,*parity*[,*port_number* [,*XON/XOFF_switch*]])	SETCOM sets the serial communications.	146

5-2-3 Loop and Conditional Structures

The table below outlines the loop and conditional structure commands. Refer to the specified pages for details.

Name	Syntax	Description	Page
FOR TO STEP NEXT	FOR *variable* = *start* TO *end* [STEP *increment*] *<commands>* NEXT *variable*	FOR ... NEXT loop allows a program segment to be repeated with increasing/decreasing *variable*.	107
GOSUB RETURN	GOSUB *label* ... RETURN	GOSUB jumps to a subroutine at the line just after *label*. The program execution returns to the next instruction after a RETURN is given.	110
GOTO	GOTO *label*	GOTO jumps to the line containing the *label*.	110

IF THEN ELSE ENDIF	IF *condition* THEN
	<commands>
	[ELSE
	<commands>]
	ENDIF
	IF *condition* THEN *<commands>*
		IF controls the flow of the program base on the results of the *condition*.	111

ON GOSUB or GOTO	ON *expression* GOSUB *label*{, *label*} ON *expression* GOTO *label*{, *label*}	ON GOSUB or ON GOTO enables a conditional jump to one of several labels.	127

Name	Syntax	Description	Page
REPEAT UNTIL	REPEAT <*commands*> UNTIL *condition*	The REPEAT ... UNTIL loop allows the program segment to be repeated until the *condition* becomes TRUE.	141
WHILE WEND	WHILE *condition* <*commands*> WEND	The WHILE ... WEND loop allows the program segment to be repeated until the *condition* becomes FALSE.	155

5-2-4 Program Commands and Functions

The table below outlines commands used for general programming purposes. Refer to the specified pages for details.

Name	Syntax	Description	Page
Statement separator	<*statement*>:<*statement*>	The statement separator enables more statements on one line.	87
Comment field	‘ [<*Comment field*>]	The single quote enables a line not to be executed.	87
AUTORUN	AUTORUN	AUTORUN starts all the programs that have been set to run at start-up.	90
COMPILE	COMPILE	COMPILE compiles the current program.	97
COPY	COPY *“program_name”* *“new_program_name”*	COPY copies an existing program in memory to a new program.	98
DEL	DEL [“*program_name”*] RM [“*program_name”*]	DEL deletes a program from memory.	102
DIR	DIR	DIR displays a list of the programs held in memory, their size and their RUNTYPE.	103
EDIT	EDIT[*line_number*] ED[*line_number*]	EDIT allows a program to be modified using a VT100 Terminal.	104
EPROM	EPROM	EPROM stores the BASIC programs in the MC Unit in the flash EPROM.	104
FREE	FREE	FREE returns the amount of available memory.	109
HALT	HALT	HALT stops execution of all programs currently running.	111
LIST	LIST	LIST prints the lines of a program.	115
NEW	NEW	NEW deletes all the program lines in MC Unit memory.	126
PROCESS	PROCESS	PROCESS returns the running status and task number for each current task.	136
RENAME	RENAME *“old_program_name“* *“new_program_name“*	RENAME changes the name of a program in the MC Unit directory.	140
RUN	RUN [*“program_name“* [,*task_number*]]	RUN executes a program.	142
RUNTYPE	RUNTYPE *“program_name“*, *auto_run*[,*step_number*]	RUNTYPE determines if a program is run at start-up, and which task it is to run on.	143
SELECT	SELECT *“program_name“*	SELECT specifies the current program.	145
STEPLINE	STEPLINE [“*program_name*” [, *task_number*]]	STEPLINE executes a single line in a program.	148
STOP	STOP [“*program_name*” [, *task_number*]	STOP halts program execution.	148
TROFF	TROFF [“*program_name*”]	TROFF suspends a trace at the current line and resumes normal program execution.	150
TRON	TRON	TRON creates a breakpoint in a program.	150

5-2-5 System Commands and Parameters

The table below outlines the system commands and parameters. Refer to the specified pages for details.

Name	Syntax	Description	Page
AXIS	AXIS(*axis_number*)	AXIS sets the axis for a command, axis parameter read, or assignment to a particular axis.	90
BASICERROR	BASICERROR	Used to run a routine when an error occurs in a a BASIC command.	92
BATTERY_LOW	BATTERY_LOW	BATTERY_LOW returns the status of the battery.	93
CHECKSUM	CHECKSUM	CHECKSUM contains the checksum for the RAM.	96
CLEAR	CLEAR	CLEAR clears all global variables and the local variables on the current task.	96
CLEAR_BIT	CLEAR_BIT(*bit_number*, *vr_number*)	CLEAR_BIT clears the specified bit of the specified VR variable.	96
COMMSERROR	COMMSERROR	COMMSERROR contains all the communications errors that have occurred since the last time that it was initialised.	97
CONTROL	CONTROL	CONTROL contains the type of MC Unit in the system.	98
DISPLAY	DISPLAY	DISPLAY contains a code indicating the application of the bank of 8 indicators on the front panel of the MC Unit.	103
ERROR_AXIS	ERROR_AXIS	ERROR_AXIS contains the number of the axis which caused the enable WDOG relay to open when a following error exceeded its limit.	104
INITIALISE	INITIALISE	INITIALISE set all axis parameters to their default values	113
LAST_AXIS	LAST_AXIS	LAST_AXIS contains the number of the last axis processed by the system.	114
LOCK	LOCK(*code*) UNLOCK(*code*)	LOCK prevents the programs from being viewed or modified.	116
MOTION_ERROR	MOTION_ERROR	MOTION_ERROR contains an error flag for axis motion errors.	118
NIO	NIO	NIO contains the number of inputs and outputs connected to the system.	126
PLC_TYPE	PLC_TYPE	PLC_TYPE contains the PC CPU Unit model that the MC402-E is connected to on the backplane.	132
POWER_UP	POWER_UP	POWER_UP contains the location of programs to be used at start-up.	134
READ_BIT	READ_BIT(*bit_number*, *vr_number*)	READ_BIT returns the value of the specified bit in the specified VR variable.	138
RESET	RESET	RESET resets all local variables on a task.	141
SET_BIT	SET_BIT(*bit_number*, *vr_number*)	SET_BIT command sets the specified bit in the specified VR variable to one.	146
TABLE	TABLE(*address, value*{, *value*}) TABLE(*address*)	TABLE loads and reads data to the Table array.	148
TSIZE	TSIZE	TSIZE returns the size of the currently defined table.	151
VERSION	VERSION	VERSION returns the version number of the BASIC language installed in the MC Unit.	152
VR	VR(*expression*)	VR calls the value of or assigns a value to a global numbered variable.	152
WA	WA(*time*)	WA holds program execution for the number of milliseconds specified.	153
WAIT IDLE	WAIT IDLE	WAIT IDLE suspends program execution until the base axis has finished executing its current move and any buffered move.	154
WAIT LOADED	WAIT LOADED	WAIT LOADED suspends program execution until the base axis has no moves buffered ahead other than the currently executing move.	154

WAIT UNTIL	`WAIT UNTIL` *condition*	WAIT UNTIL repeatedly evaluates the *condition* until TRUE.	154
WDOG	`WDOG`	WDOG contains a software switch used to control the enable relay contact used to enable all drivers.	155

5-2-6 Mathematical and Logical Functions

The table below outlines the mathematical and logical functions. Refer to the specified pages for details.

Name	Syntax	Description	Page
Multiply: *	*expression_1* * *expression_2*	* multiplies any two valid expressions.	84
Power: ^	expression_1 ^ expression_2	^ takes the power of any two valid expressions	84
Add: +	*expression_1* + *expression_2*	+ adds any two valid expressions.	84
Subtract: –	*expression_1* - *expression_2*	– subtracts any two valid expressions.	85
Divide: /	*expression_1* / *expression_2*	/ divides any two valid expressions.	85
Is Less Than: <	*expression_1* < *expression_2*	< returns TRUE if *expression_1* is less than *expression_2*, otherwise FALSE.	85
Is Less Than Or Equal To: <=	*expression_1* <= *expression_2*	<= returns TRUE if *expression_1* is less than or equal to *expression_2*, otherwise FALSE.	85
Is Not Equal To: <>	*expression_1* <> *expression_2*	<> returns TRUE if *expression_1* is not equal to *expression_2*, otherwise FALSE.	86
Is Equal To: =	*expression_1* = *expression_2*	= returns TRUE if *expression_1* is equal to *expression_2*, otherwise FALSE.	86
Is Greater Than: >	*expression_1* > *expression_2*	> returns TRUE if *expression_1* is greater than *expression_2*, otherwise FALSE.	86
Is Greater Than or Equal To: >=	*expression_1* >= *expression_2*	>= returns TRUE if *expression_1* is greater than or equal to *expression_2*, otherwise FALSE.	86
ABS	`ABS(`*expression*`)`	ABS returns the absolute value of *expression*.	87
ACOS	`ACOS(`*expression*`)`	ACOS returns the arc-cosine of *expression*.	87
AND	*expression_1* `AND` *expression_2*	AND performs an AND operation on corresponding bits of the integer parts of two valid BASIC expressions.	88
ASIN	`ASIN(`*expression*`)`	ASIN returns the arc-sine of *expression*.	89
ATAN	`ATAN(`*expression*`)`	ATAN returns the arc-tangent of *expression*.	89
ATAN2	`ATAN2(`*expression_1*`,` *expression_2*`)`	ATAN2 returns the arc-tangent of the ratio *expression_1* / *expression_2*.	89
COS	`COS(`*expression*`)`	COS returns the cosine of *expression*.	98
EXP	`EXP(`*expression*`)`	EXP returns the exponential value of *expression*.	105
FRAC	`FRAC(`*expression*`)`	FRAC returns the fractional part of *expression*.	108
INT	`INT(`*expression*`)`	INT returns the integer part of *expression*.	113
LN	`LN(`*expression*`)`	LN returns the natural logarithm of *expression*.	116
MOD	*expression_1* `MOD` *expression_2*	MOD returns the *expression_2* modulus of an *expression_1*.	118
NOT	`NOT(`*expression*`)`	NOT performs an NOT operation on corresponding bits of the integer part of the expression.	126
OR	*expression_1* `OR` *expression_2*	OR performs an OR operation between corresponding bits of the integer parts of two valid BASIC expressions.	129
SGN	`SGN(`*expression*`)`	SGN returns the sign of *expression*.	146
SIN	`SIN(`*expression*`)`	SIN returns the sine of *expression*.	147
SQR	`SQR(`*expression*`)`	SQR returns the square root of *expression*.	147
TAN	`TAN(`*expression*`)`	TAN returns the tangent of *expression*.	149
XOR	*expression_1* `XOR` *expression_2*	XOR performs an XOR function between corresponding bits of the integer parts of two valid BASIC expressions.	156

5-2-7 Constants

The table below outlines the constants. Refer to the specified pages for details.

Name	Description	Page
FALSE	FALSE returns the numerical value 0.	106
OFF	OFF returns the numerical value 0.	127
ON	ON returns the numerical value 1.	127
PI	PI returns the numerical value 3.1416.	130
TRUE	TRUE returns the numerical value -1.	151

5-2-8 Motion Perfect Commands, Functions and Parameters

The table below outlines the Motion Perfect commands, functions, and parameters. Refer to the specified pages for details.

Name	Syntax	Description	Page
SCOPE	SCOPE(*ON/OFF_control*, *period*, *table_start*, *table_stop*, *P0*[, *P1*[, *P2*[, *P3*]]])	SCOPE programs the system to automatically store up to 4 parameters every sample period.	144
SCOPE_POS	SCOPE_POS	SCOPE_POS contains the current Table position at which the SCOPE command is currently storing its first parameter.	145
TRIGGER	TRIGGER	TRIGGER starts a previously set SCOPE command.	150

5-2-9 Axis Parameters

The table below outlines the axis parameters. Refer to the specified pages for details.

Name	Description	Page
ACCEL	ACCEL contains the axis acceleration rate. The rate is in units/s^2.	87
ATYPE	ATYPE contains the axis type.	90
AXISSTATUS	AXISSTATUS contains the axis status.	91
CLOSE_WIN	CLOSE_WIN defines the end of the window in which a registration mark is expected.	96
CREEP	CREEP contains the creep speed on the current base axis.	99
D_GAIN	D_GAIN contains the derivative gain for the axis.	99
DAC	DAC contains a voltage applied directly to a servo axis.	99
DAC_OUT	DAC_OUT contains the voltage being applied to the servo.	100
DATUM_IN	DATUM_IN contains the input number to be used as the origin input. The number can be between 0 and 15.	101
DECEL	DECEL contains the axis deceleration rate in units/s^2.	101
DEMAND_EDGES	DEMAND_EDGES contains the current value of the DPOS axis parameter in edge units.	102
DPOS	DPOS contains the demand position, in user units, generated by the move commands.	103
ENCODER	ENCODER contains a raw copy of the encoder or resolver hardware register.	104
ENDMOVE	ENDMOVE holds the position of the end of the current move in user units.	104

Name	Description	Page
ERRORMASK	ERRORMASK contains a mask value that is ANDed bit by bit with the AXISSTATUS axis parameter on every servo cycle to determine if a runtime error should turn OFF the enable (WDOG) relay.	105
FAST_JOG	FAST_JOG contains the input number to be used as the fast jog input. The number can be between 0 and 15.	106
FE	FE is the position error in user units, and is equal to the demand position in the DPOS axis parameter minus the measured position in the MPOS axis parameter.	106
FE_LIMIT	FE_LIMIT contains the maximum allowable following error in user units.	106
FE_RANGE	FE_RANGE contains the following error report range.	106
FHOLD_IN	FHOLD_IN contains the input number to be used as the feedhold input. The number can be between 0 and 31.	106
FHSPEED	FHSPEED contains the feedhold speed.	107
FS_LIMIT	FS_LIMIT contains the absolute position of the forward software limit in user units.	109
FWD_IN	FWD_IN contains the input number to be used as a forward limit input. The number can be between 0 and 31.	109
FWD_JOG	FWD_JOG contains the input number to be used as a jog forward input. The number can be between 0 and 31.	109
I_GAIN	I_GAIN contains the integral gain.	111
JOGSPEED	JOGSPEED sets the slow jog speed in user units for an axis to run at when performing a slow jog.	114
MARK	MARK contains TRUE when a registration event has occurred to indicate that the value in the REG_POS axis parameter is valid.	116
MERGE	MERGE is a software switch that can be used to enable or disable the merging of consecutive moves.	116
MPOS	MPOS is the position of the axis in user units as measured by the encoder or resolver.	125
MSPEED	MSPEED represents the change in the measured position in user units/s in the last servo period.	125
MTYPE	MTYPE contains the type of move currently being executed.	125
NTYPE	NTYPE contains the type of the move in the Next Move buffer.	127
OFFPOS	OFFPOS contains an offset that will be applied to the demand position without affecting the move in any other way.	127
OPEN_WIN	OPEN_WIN defines the positions for the REGIST command.	129
OUTLIMIT	OUTLIMIT contains an output limit that restricts the voltage output from the MC Unit.	130
OV_GAIN	OV_GAIN contains the output velocity gain.	130
P_GAIN	P_GAIN contains the proportional gain.	130
PP_STEP	PP_STEP contains an integer value that scales the incoming raw encoder count.	134
REG_POS	REG_POS stores the position in user units at which a registration event occurred.	138

Name	Description	Page
REMAIN	REMAIN is the distance remaining to the end of the current move.	139
REP_DIST	REP_DIST contains the repeat distance, which is the allowable range of movement for an axis before the demand position and measured position are corrected.	140
REP_OPTION	REP_OPTION controls the application of the REP_DIST axis parameter.	140
REV_IN	REV_IN contains the input number to be used as a reverse limit input. The number can be between 0 and 31.	142
REV_JOG	REV_JOG contains the input number to be used as a jog reverse input. The input can be between 0 and 31.	142
RS_LIMIT	RS_LIMIT contains the absolute position of the reverse software limit in user units.	142
SERVO	SERVO determines whether the axis runs under servo control or open loop.	145
SPEED	SPEED contains the demand speed in units/s.	147
SRAMP	SRAMP contains the S-curve factor.	147
UNITS	UNITS contains the unit conversion factor.	151
VFF_GAIN	VFF_GAIN contains the speed feed forward gain.	152
VP_SPEED	VP_SPEED contains the speed profile speed in user units/s.	152

5-2-10 Task Commands and Parameters

The table below outlines the task commands and parameters. Refer to the specified pages for details.

Name	Description	Page
ERROR_LINE	ERROR_LINE contains the number of the line which caused the last BASIC program error.	105
PMOVE	PMOVE will contain 1 if the task buffers are occupied, and 0 if they are empty.	133
PROC	PROC allows a process parameter from a particular process to be read or set.	136
PROCNUMBER	PROCNUMBER contains the number of the task in which the currently selected program is running.	136
RUN_ERROR	RUN_ERROR contains the number of the last BASIC error that occurred on the specified task.	143
TICKS	TICKS contains the current count of the task clock pulses.	150

5-2-11 PC Data Exchange Commands

The table below outlines the PC Data Exchange Commands. Refer to the specified pages for details.

Name	Syntax	Description	Page
PLC_READ	PLC_READ(*PC_area*,*offset*,*length*,*vr_number*)	PLC_READ requests a data transfer from the CPU Unit of the PC to the MC Unit at the end of the next CPU Unit execution cycle.	131
PLC_WRITE	PLC_WRITE(*PC_Area*,*offset*,*length*,*vr_number*)	PLC_WRITE requests a data transfer from the MC Unit to the CPU Unit of the PC at the end of the next CPU Unit execution cycle.	132

5-3 Command, function and parameter description

This section describes the commands, functions and parameters which are used in the BASIC programming language.

⚠WARNING **It is the responsibility of the programmer to ensure that the motion functions are invoked correctly, with the correct number of parameters and values. Failure to do so may result in unexpected behavior, loss or damage to the machinery.**

5-3-1 Multiply: *

Type: Arithmetic Operation

Syntax: *expression_1 * expression_2*

Description: The multiply operator "*" multiplies any two valid expressions.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
factor = 10*(2.1+9)
```
The parentheses are evaluated first, and then the result, 11.1, is multiplied by 10. Therefore, *factor* would contain the value 111

5-3-2 Power: ^

Type: Arithmetic Operation

Syntax: *expression_1 ^ expression_2*

Description: The power operator "^" raises expression_1 to the power of expression_2.

⚠WARNING This operation uses floating point algorithms and may give small deviations for integer calculations.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
thirtytwo = 2^5
```
This sets the variable thirtytwo to 32.

5-3-3 Add: +

Type: Arithmetic Operation

Syntax: *expression_1 + expression_2*

Description: The add operator "+" adds any two valid expressions.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
result = 10+(2.1*9)
```
The parentheses are evaluated first, and the result, 18.9, is added to 10. Therefore, *result* would contain the value 28.9.

5-3-4 Subtract: –

Type: Arithmetic Operation

Syntax: *expression_1 - expression_2*

Description: The subtract operator "–" subtracts any two valid expressions.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
VR(0) = 10-(2.1*9)
```
The parentheses are evaluated first, and the result, 18.9, is subtracted from 10. Therefore, VR(0) would contain the value –8.9.

5-3-5 Divide: /

Type: Arithmetic Operation

Syntax: *expression_1 / expression_2*

Description: The divide operator "/" divides any two valid expressions.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
a = 10/(2.1+9)
```
The parentheses are evaluated first, and then 10 is divided by the result, 11.1. Therefore, *a* would contain the value 0.9009

5-3-6 Is Less Than: <

Type: Logical Operation

Syntax: *expression_1 < expression_2*

Description: The less than operator "<" returns TRUE if *expression_1* is less than *expression_2*, otherwise it returns FALSE.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
IF VR(1) < 10 THEN GOSUB rollup
```
If the value returned from VR(1) is less than 10, then subroutine "rollup" would be executed.

5-3-7 Is Less Than Or Equal To: <=

Type: Logical Operation

Syntax: *expression_1 <= expression_2*

Description: The less than or equal to operator "<=" returns TRUE if *expression_1* is less than or equal to *expression_2*, otherwise it returns FALSE.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
maybe = 1 <= 0
```

In the above line, 1 is not less than or equal to 0 and, therefore, variable *maybe* would contain the value 0 (FALSE).

5-3-8 Is Not Equal To: <>

Type: Logical Operation

Syntax: *expression_1* <> *expression_2*

Description: The not equal to operator "<>" returns TRUE if *expression_1* is not equal to *expression_2*, otherwise it returns FALSE.

Arguments:
expression_1
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
IF MTYPE <> 0 THEN GOTO 3000
```
If the base axis is not idle (MTYPE=0 indicates an axis idle), then a jump would be made to label 3000.

5-3-9 Is Equal To: =

Type: Logical Operation

Syntax: *expression_1* = *expression_2*

Description: The equal to operator "=" returns TRUE if *expression_1* is equal to *expression_2*, otherwise it returns FALSE.

Arguments:
expression_1
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
IF IN(7) = ON THEN GOTO label
```
If input 7 is ON, then program execution will continue at line starting "label:".

5-3-10 Is Greater Than: >

Type: Logical Operation

Syntax: *expression_1* > *expression_2*

Description: The greater than operator ">" returns TRUE if *expression_1* is greater than *expression_2*, otherwise it returns FALSE.

Arguments:
expression_1
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Examples:
Example 1
```
VR(0) = 1 > 0
```
In the above line, 1 is greater than 0 and, therefore, VR(0) would contain the value –1
Example 2
```
WAIT UNTIL MPOS > 200
```
Program execution will wait until the measured position is greater than 200.

5-3-11 Is Greater Than or Equal To: >=

Type: Logical Operation

Syntax: *expression_1* >= *expression_2*

Description: The greater than or equal to operator ">=" returns TRUE if *expression_1* is greater than or equal to *expression_2*, otherwise it returns FALSE.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
IF target >= 120 THEN MOVEABS(0)
```
If the variable *target* holds a value greater than or equal to 120, then the base axis will move to an absolute position of 0.

5-3-12 Statement separator: ":"

Type: Program command

Syntax: <*statement*>:<*statement*>

Description: The statement separator, represented by the colon ":", can be used to separate BASIC statements on a multi-statement line. This separator can be used both on the command line as in programs.

Example:
```
PRINT "THIS LINE": GET low : PRINT "DOES THREE THINGS"
```

5-3-13 Comment field: ‘

Type: Program command

Syntax: ‘ [<*Comment field*>]

Description: The single quote " ‘ " can be used in a program to mark a line as being comment which will not be executed. The single quote can be put at the beginning of a line or after any valid statement.

Example:
```
‘ This line will not be printed.
PRINT "Start"
```

5-3-14 ABS

Type: Mathematical Function

Syntax: ABS(*expression*)

Description: ABS converts a negative number into its positive equal. Positive numbers are unaltered.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
IF ABS(VR(0)) > 100 THEN PRINT "VR(0) Outside ±100"
```

5-3-15 ACCEL

Type: Axis parameter

Description: ACCEL contains the axis acceleration rate. The rate is in units/s^2. The parameter can have any positive value including zero.

See also: AXIS, DECEL, UNITS

Example:
```
BASE(0)
ACCEL = 100       ‘Set acceleration rate
PRINT "Acceleration rate: ";ACCEL;" mm/s/s"
ACCEL AXIS(2) = 100 ‘Sets acceleration rate for axis (2)
```

5-3-16 ACOS

Type: Mathematical Function

Syntax: ACOS(*expression*)

Description: ACOS returns the arc-cosine of the *expression*. The *expression* value must be between –1 and 1. The result in radians will be between 0 and PI. Input values outside the range will return zero.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT ACOS(-1)
3.1416
```

5-3-17 ADDAX

Type: Motion Control Command

Syntax: ADDAX(*axis*)

Description: The ADDAX command takes the demand position changes from the superimposed axis as specified by the *axis* argument and adds them to any movement running on the axis to which the command is issued. After the ADDAX command has been issued the link between the two axes remains until broken. Use ADDAX(-1) to cancel the axis link.

ADDAX works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

ADDAX allows an axis to perform the moves specified for 2 axes added together. Combinations of more than two axes can be made by applying ADDAX to the superimposed axis as well.

⚠ **WARNING** Beware that giving several ADDAX commands in a system can create a dangerous loop when for instance one axis is linked to another and vice versa. This may cause instability in the system.

Arguments: ***axis***
The axis to be set as a superimposed axis. Set the argument to –1 to cancel the link and return to normal operation.

See also: AXIS

Example: Pieces are placed onto a continuously moving belt and further along the line are picked up. A detection system gives an indication as to whether a piece is in front of or behind its nominal position, and how far.

In the example below, axis 0 is assumed to be the base axis and it executes a continuous forward movement and a superimposed move on axis 2 is used to apply offsets according to the offset calculated in a subroutine.

```
FORWARD                    'Set continuous move
ADDAX(4)                   'Add axis 4 for correction
REPEAT
   GOSUB getoffset         'Get offset to apply
   MOVE(offset) AXIS(2)
UNTIL IN(2) = ON           'Until correction is done
```

5-3-18 AND

Type: Logical Operator

Syntax: *expression_1* AND *expression_2*

Description: AND performs an AND operation on the corresponding bits of the integer parts of two valid BASIC expressions.

The AND operation between two bits is defined as follows:

Bit 1	Bit 2	Result
0	0	0
0	1	0
1	0	0
1	1	1

Arguments:

expression_1
Any valid BASIC expression.

expression_2
Any valid BASIC expression.

Examples:

Example 1

```
VR(0) = 10 AND (2.1*9)
```

The parentheses are evaluated first, but only the integer part of the result, 18, is used for the AND operation. Therefore, this expression is equivalent to the following:

```
VR(0) = 10 AND 18
```

The AND is a bit operator and so the binary action is as follows:

Therefore, VR(0) will contain the value 2.

```
          01010
AND       10010
-----------------
          00010
```

Example 2

```
IF MPOS AXIS(0) > 0 AND MPOS AXIS(1) > 0 THEN GOTO cycle1
```

5-3-19 ASIN

Type: Mathematical Function

Syntax: ASIN(*expression*)

Description: ASIN returns the arc-sine of the *expression*. The *expression* value must be between –1 and 1. The result in radians will be between –PI/2 and PI/2. Input values outside the range will return zero.

Arguments:

expression
Any valid BASIC expression.

Example:

```
>> PRINT ASIN(-1)
-1.5708
```

5-3-20 ATAN

Type: Mathematical Function

Syntax: ATAN(*expression*)

Description: ATAN returns the arc-tangent of the *expression*. ATAN can have any value and the result will be in radians.

Arguments:

expression
Any valid BASIC expression.

Example:

```
>> PRINT ATAN(1)
0.7854
```

5-3-21 ATAN2

Type: Mathematical Function

Syntax: ATAN2(*expression_1*,*expression_2*)

Description: ATAN2 returns the arc-tangent of the nonzero complex number (expression_2, expression_1), which is equivalent to the angle between a point with coordinate (expression_1, expression_2) and the x-axis. If expression_2 >= 0, the result is equal to the value of ATAN (*expression_1* / *expression_2*). The result in radians will be between –PI and PI.

Arguments: ***expression_1***
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
>> PRINT ATAN2(0,1)
0.0000
```

5-3-22 ATYPE

Type: Axis parameter

Description: ATYPE contains the axis type. The following values can be set:

0 Virtual axis
2 Servo axis
3 Encoder axis

The ATYPE parameters are set by the system at start-up to the default value of the axis. The user is able to change the type of the axis at any time. The default for axes 0 to 3 are 2 (servo axis) and for axes 4 to 7 are 0 (virtual axis).

Refer to *1-3 Motion Control Concepts* for more details on the different axis types.

Note Only the axes with the supported hardware, which are axis 0 to 3 can be operating as servo axis or a encoder axis.

See also: AXIS

Example:
```
>> PRINT ATYPE AXIS(2)
2.0000
```
The above command line and response show that axis 2 is operating as a servo axis.

5-3-23 AUTORUN

Type: Program Command

Syntax: `AUTORUN`

Description: AUTORUN starts all the programs that have been set to run at start-up.

See also: RUNTYPE

5-3-24 AXIS

Type: System Command

Syntax: `AXIS(`*axis_number*`)`

Description: The AXIS modifier sets the axis for a single motion command or a single axis parameter read/write to a particular axis. AXIS is effective only for the command line in which it is programmed. Use the BASE command to change the base axis for all following command lines.

Arguments: ***axis_number***
Any valid BASIC expression specifying the axis number.

Precautions: The AXIS command can be used to modify any axis parameter expression and the following axis dependent commands: ADDAX, CAM, CAMBOX, CANCEL, CONNECT, DATUM, DEFPOS, FORWARD, MOVEABS, MOVECIRC,

MHELICAL, MOVELINK, MOVE, MOVEMODIFY and REVERSE. Other commands for which AXIS is used are: REGIST, WAIT IDLE and WAIT LOADED.

See also: BASE

Examples:

Example 1

```
PRINT MPOS AXIS(3)
```

Example 2

```
MOVE(300) AXIS(2)
```

Example 3

```
REPDIST AXIS(3) = 100
```

5-3-25 AXISSTATUS

Type: Axis Parameter

Description: AXISSTATUS contains the axis status. The meaning of each bit is as follows:

Bit	Description	Value
0	Unused	1
1	Following Error Exceeds Warning Range	2
2	Unused	4
3	Unused	8
4	In Forward Limit	16
5	In Reverse Limit	32
6	Origin Search (DATUM) in progress	64
7	Feedhold	128
8	Following Error Exceeds Limit	256
9	In Forward Software Limit	512
10	In Reverse Software Limit	1024
11	Cancelling Move	2048

Note This parameter is read-only.

See also: AXIS, ERRORMASK

Example:

```
IF (AXISSTATUS AND 16)>0 THEN PRINT "In forward limit"
```

5-3-26 BASE

Type: Motion Control Command

Syntax: BASE(*axis_1*[,*axis_2*[, *axis_3*[, *axis_4*[, *axis_5*[, *axis_6*[, *axis_7*[, *axis_8*]]]]]]])
BASE

Alternative: BA(*axis_1*[, *axis_2*[, *axis_3*[, *axis_4*[, *axis_5*[, *axis_6*[, *axis_7*[, *axis_8*]]]]]]])
BA

Description: BASE is used to set the default base axis or to set a specified axis sequence group. All subsequent motion commands and axis parameters will apply to the base axis or the specified axis group unless the AXIS command is used to specify a temporary base axis. The base axis is effective until it is changed again with BASE.

Each BASIC process can have its own axis group and each program can set its own axis group independently. Use the PROC modifier to access the parameter for a certain task.

The BASE order grouping can be set by explicitly assigning the order of axes. This order is used for interpolation purposes in multi-axes linear, circular and helical moves. The default for the base axis group is (0,1,2,3,4,5,6,7) at start-up or when a program starts running on a task.

The BASE command without any arguments returns the current base order grouping.

Arguments: ***axis_i***

The number of the axis set as the base axis and any subsequent axes in the group order for multi-axis moves.

See also: AXIS

Examples: **Example 1**

It is possible to program each axis with its own speed, acceleration and other parameters.

```
BASE(1)
UNITS = 2000   'Set unit conversion factor for axis 1
SPEED = 100    'Set speed for axis 1
ACCEL = 5000   'Set acceleration rate for axis 1
BASE(2)
UNITS = 2000   'Set unit conversion factor for axis 2
SPEED = 125    'Set speed for axis 2
ACCEL = 10000  'Set acceleration rate for axis 2
```

Example 2

In the example below, axes 0, 1 and 2 will move to the specified positions at the speed and acceleration set for axis 0. BASE(0) sets the base axis to axis 0, which determines the three axes used by MOVE and the speed and acceleration rate.

```
BASE(0)
MOVE(100,-23.1,1250)
```

Example 3

Assume that we want to do a linearly interpolated move between axes 0,2 and 3. The following BASE command should be used to assign the grouping of axes. This will set the internal base group order to (0,2,3,4,5,6,7,1).

```
BASE(0,2,3)
MOVE(100,200,30)
```

Example 4

On the command line the base group order can be shown by typing BASE.

```
>> BASE(0,2,3)
>> BASE
(0,2,3,4,5,6,7,1)
```

Example 5

Use the PROC modifier to show the base group order of a certain task.

```
>> RUN "PROGRAM",5
>> BASE PROC(5)
(0,1,2,3,4,5,6,7)
```

5-3-27 BASICERROR

Type: System Command

Description: The BASICERROR command can be used to run a routine when a run-time error occurs in a program. BASICERROR can only be used as part of an ON ... GOSUB or ON ... GOTO command. This command is required to be executed once in the BASIC program. If several commands are used only the one executed last is effective.

See also: ERROR_LINE, ON, RUN_ERROR

Example: If an error occurs in a BASIC command in the following example, then the error routine will be executed.

```
ON BASICERROR GOTO error_routine
....
no_error = 1
```

```
   STOP
error_routine:
   IF no_error = 0 THEN
      PRINT "The error ";RUN_ERROR[ 0 ];
      PRINT " occurred in line ";ERROR_LINE[ 0 ]
   ENDIF
   STOP
```

The IF statement is present to prevent the program going into error routine when it is stopped normally.

5-3-28 BATTERY_LOW

Type: System Parameter

Description: This parameter monitors the status of the internal RAM backup battery. When the battery is too low the status of the parameter changes to TRUE, otherwise it will be FALSE.

Note This parameter is read-only.

5-3-29 CAM

Type: Motion Control Command

Syntax: CAM(*start_point*, *end_point*, *table_multiplier*, *distance*)

Description: The CAM command is used to generate movement of an axis following a position profile which is stored in the Table array. The table values are absolute positions relative to the starting point and are specified in encoder edges. The table of values is specified with the TABLE command. The movement can be defined with any number of points from 2 to 16,000. The MC Unit moves continuously between the values in the table to allow a number of points to define a smooth profile. Two or more CAM commands can be executed simultaneously using the same or overlapping values in the Table array. The Table profile is traversed once.

CAM requires that the start element in the Table array has value zero. The *distance* argument together with the SPEED and ACCEL parameters determine the speed moving through the Table array. Note that in order to follow the CAM profile exactly the ACCEL parameter of the axis must be at least 1000 times larger than the SPEED parameter.

CAM works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Arguments:

start_point

The address of the first element in the Table array to be used.

Being able to specify the start point allows the Table array to hold more than one profile and/or other information.

end_point

The address of the end element in the Table array.

table_multiplier

A value used to scale the values stored in the table.

The table values are specified in encoder edges. The table multiplier is set to 1 in this case but can be set to any non-zero value to alter the values set in the Table array.

distance

A factor given in user units that controls the speed of movement through the table. The time taken to execute CAM depends on the current axis speed and this distance. For example, assume the system is being programmed in mm and the speed is set to 10mm/s and the acceleration sufficiently high. If a distance of 100mm is specified, CAM will take 10 seconds to execute.

The SPEED parameter in the base axis allows modification of the speed of movement when using the CAM move.

See also: ACCEL, AXIS, CAMBOX, SPEED, TABLE

Example: Assume that a motion is required to follow the following position equation:

$$t(x) = x*25 + 10000(1–\cos(x))$$

Here, x is in degrees. This example is for a table that provides a simple oscillation superimposed with a constant speed. To load the table and cycle it continuously the following code would be used.

```
   GOSUB camtable
loop:
   CAM(1,19,1,200)
   GOTO loop
```

Note The subroutine *camtable* would load the data below into the Table array.

Table position	Degree	Value
1	0	0
2	20	1103
3	40	3340
4	60	6500
5	80	10263
6	100	14236
7	120	18000
8	140	21160
9	160	23396
10	180	24500
11	200	24396
12	220	23160
13	240	21000
14	260	18236
15	280	15263
16	300	12500
17	320	10340
18	340	9103
19	360	9000

5-3-30 CAMBOX

Type: Motion Control Command

Syntax: CAMBOX(*start_point*, *end_point*, *table_multiplier*, *link_distance*, *link_axis* [, *link_option*[, *link_position*]])

Description: The CAMBOX command is used to generate movement of an axis following a position profile in the Table array. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox. The table values are absolute position relative to the starting point and are specified in encoder edges.

The table of values is specified with the TABLE command. The movement can be defined with any number of points from 2 to 16,000. Being able to specify the start point allows the Table array to be used to hold more than one profile and/or other information. The MC Unit moves continuously between the values in the table to allow a number of points to define a smooth profile. Two or more CAMBOX commands can be executed simultaneously using the same or overlapping values in the Table array.

The CAMBOX command requires the start element of the table to have value zero. Note also that CAMBOX command allows traversing the table backwards as well as forwards depending on the master axis direction.

The link_option argument can be used to specify different options to start the command and to specify a continuous CAM. For example, if the link_option is set to 4 then the CAMBOX operates like a “physical” CAM.

CAMBOX works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Arguments:

start_point
The address of the first element in the Table array to be used.

end_point
The address of the end element in the Table array.

table_multiplier
A value used to scale the values stored in the table.

The table values are normally specified in encoder edges. The table multiplier is set to 1 in this case but can be set to another value to alter the values set in the Table array.

link_distance
The distance in user units the link axis must move to complete the specified output movement. The link distance must be specified as a positive distance.

link_axis
The axis to link to.

link_option

1	Link starts when registration event occurs on link axis.
2	Link starts at an absolute position on link axis (see *link_position*).
4	CAMBOX repeats automatically and bi-directionally. This option is canceled by setting bit 1 of REP_OPTION parameter (i.e. REP_OPTION = REP_OPTION OR 2).
5	Combination of options 1 and 4.
6	Combination of options 2 and 4.

link_position
The absolute position where CAMBOX will start when *link_option* is set to 2.

Precautions: While CAMBOX is being executed, the ENDMOVE parameter will be set to the end of the previous move. The REMAIN axis parameter will hold the remainder of the distance on the link axis.

See also: AXIS, CAM, REP_OPTION, TABLE

5-3-31 CANCEL

Type: Motion Control Command

Syntax: `CANCEL[(1)]`

Alternative: `CA[(1)]`

Description: CANCEL cancels the current move on an axis. Speed-profiled moves (FORWARD, REVERSE, MOVE, MOVEABS, MOVECIRC and MHELICAL) will be decelerated at the programmed deceleration rate and then stopped. Other moves will be immediately stopped.

CANCEL command cancels the contents of the current move buffer (MTYPE). The command CANCEL(1) command cancels the contents of the next move buffer (NTYPE) without affecting the current move in the MTYPE buffer.

CANCEL works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions:

- CANCEL cancels only the presently executing move. If further moves are buffered they will then be loaded.
- During the deceleration of the current move additional CANCELs will be ignored.
- CANCEL(1) cancels only the presently buffered move. Any moves stored in the task buffers indicated by the PMOVE variable can be loaded into the buffer as soon as the buffered move is cancelled.

See also: AXIS, MTYPE, NTYPE, PMOVE, RAPIDSTOP

Examples:

Example 1

```
FORWARD
WA(10000)
CANCEL
```

Example 2

```
MOVE(1000)
MOVEABS(3000)
CANCEL   'Cancel the move to 3000 and move to 4000 instead.
MOVEABS(4000)
```

Note that the command MOVEMODIFY is a better solution for modifying end-points of moves in this case.

5-3-32 CHECKSUM

Type: System Parameter

Description: CHECKSUM contains the checksum for the RAM. At start-up, the checksum is recalculated and compared with the previously held value.

Note This parameter is read-only.

5-3-33 CLEAR

Type: System Command

Syntax: `CLEAR`

Description: CLEAR resets all global VR variables to zero and when used in program will also reset the local variables on the current task to zero.

See also: RESET, VR

5-3-34 CLEAR_BIT

Type: System Command

Syntax: `CLEAR_BIT(`*bit_number*, *vr_number*`)`

Description: The CLEAR_BIT command resets the specified bit in the specified VR variable to zero. Other bits in the variable will keep their values.

Arguments:

bit_number

The number of the bit to be reset. Range: [0, 23].

vr_number

The number of the VR variable for which the bit will be reset. Range: [0, 250]

See also: READ_BIT, SET_BIT, VR

5-3-35 CLOSE_WIN

Type: Axis Parameter

Alternative: `CW`

Description: CLOSE_WIN defines the end of the window inside or outside which a registration mark is expected. The value is in user units.

See also: AXIS, OPEN_WIN, REGIST, UNITS

5-3-36 COMMSERROR

Type: System Parameter

Description: COMMSERROR contains the serial communication errors that have occurred since the last time that it was initialized. The bits in COMMSERROR are defined as follows:

Bit	Value
0	Receive buffer overrun on Network channel
1	Retransmit buffer overrun on Network channel
2	Receive structure error on Network channel
3	Transmit structure error on Network channel
4	Error RS-232C programming port A
5	Error RS-232C programming port A
6	Error RS-232C programming port A
7	Error RS-232C programming port A
8	NA
9	NA
10	NA
11	NA
12	Error RS-232C port B
13	Error RS-232C port B
14	Error RS-232C port B
15	Error RS-232C port B
16	NA
17	NA
18	NA
19	NA

Note This parameter is read-only.

5-3-37 COMPILE

Type: Program Command

Syntax: `COMPILE`

Description: COMPILE compiles the current program to intermediate code.

Program are compiled automatically by the system software prior to program execution or when another program is selected.

5-3-38 CONNECT

Type: Motion Control Command

Syntax: `CONNECT(`*ratio*,*driving_axis*`)`

Alternative: `CO(`*ratio*,*driving_axis*`)`

Description: CONNECT connects the demand position of an axis to the measured movements of the axis specified by *driving_axis* to produce an electronic gearbox.

The ratio can be changed at any time by executing another CONNECT command on the same axis. To change the driving axis the CONNECT command needs to be cancelled first. CONNECT with different driving axis will be ignored. The CONNECT command can be cancelled with a CANCEL or RAPIDSTOP command.

CONNECT works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Arguments:

ratio
Holds the number of edges the base axis is required to move per edge increment of the driving axis. The ratio value can be either positive or negative and has sixteen bit fractional resolution. Note that the ratio is specified as an encoder edge ratio.

driving_axis
Defines the axis that will drive the other axis.

See also: AXIS, CANCEL, RAPIDSTOP

Example: In a press feed, a roller is required to rotate at a speed one quarter of the measured rate from an encoder mounted on the incoming conveyor. The roller is wired to axis 0. An input channel monitors the encoder pulses from the conveyor and forms axis 1. The following code can be used.

```
BASE(1)
SERVO = OFF    'This axis is used to monitor the conveyor
BASE(0)
SERVO = ON
CONNECT(0.25,1)
```

5-3-39 CONTROL

Type: System Parameter

Description: CONTROL contains the type of MC Unit in the system. For the MC402-E the CONTROL parameter returns value 251.

Note This parameter is read-only.

5-3-40 COPY

Type: Program Command

Syntax: COPY *"program_name" "new_program_name"*

Description: COPY copies an existing program in memory to a new program with the specified name. The program names can also be specified without quotes.

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can select the command from the Program menu.

Arguments:

program_name
Name of the program to be copied.

new_program_name
Name to use for the new program.

See also: DEL, NEW, RENAME

Example:

```
>> COPY "prog" "newprog"
```

5-3-41 COS

Type: Mathematical Function

Syntax: COS(*expression*)

Description: COS returns the cosine of the *expression*. Input values are in radians and may have any value. The result value will be in the range from -1 to 1.

Arguments:

expression
Any valid BASIC expression.

Example:

```
>> PRINT COS(0)
1.0
```

5-3-42 CREEP

Type: Axis Parameter

Description: The CREEP parameter contains the creep speed on the axis. The creep speed is used for the slow part of an origin search sequence. CREEP is allowed to have any positive value (including zero).

The creep speed is entered in units/s using the unit conversion factor UNITS. For example, if the unit conversion factor is set to the number of encoder edges/inch, the speed is set in inches/s.

See also: AXIS, DATUM, UNITS

Example:

```
BASE(2)
CREEP = 10
SPEED = 500
DATUM(4)
CREEP AXIS(1) = 10
SPEED AXIS(1) = 500
DATUM(4) AXIS(1)
```

5-3-43 D_GAIN

Type: Axis Parameter

Description: D_GAIN contains the derivative gain for the axis. The derivative output contribution is calculated by multiplying the change in following error with D_GAIN. The default value is zero.

Adding derivative gain to a system is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used. High values may cause oscillation.

See section *1-4-2 Servo System Principles* for more details.

See also: AXIS, I_GAIN, OV_GAIN, P_GAIN, VFF_GAIN

Precautions: In order to avoid any instability the servo gains should be changed only when the SERVO is OFF.

5-3-44 DAC

Type: Axis Parameter

Description: DAC contains the voltage value which is applied directly to the Servo Driver when the base axis is in open loop (SERVO=OFF). The range of the DAC parameter depends on the OUTLIMIT parameter and is by default from -2048 to 2047. This corresponds to putting - 10 V to 10 V on the output.

The value currently being applied to the drive is read using the DAC_OUT axis parameter.

See also: AXIS, DAC_OUT, OUTLIMIT, SERVO

Example: The following lines can be used to force a square wave of amplitude ±5V and period of approximately 500ms on axis 0.

```
   WDOG = ON
   SERVO = OFF
square:
   DAC AXIS(0) = 1024
   WA(250)
   DAC AXIS(0) = -1024
   WA(250)
   GOTO square
```

5-3-45 DAC_OUT

Type: Axis Parameter

Description: DAC_OUT contains the voltage being applied to the servo.

The voltage being applied to the servo comes from one of two sources. If the SERVO axis parameter is OFF, then the value set in DAC axis parameter is written to the axis hardware. If the SERVO parameter is ON, then a value calculated using the servo algorithm is written to the axis hardware. Either way, the voltage can be read back using DAC_OUT.

The return value will be between –2048 and 2047 and will indicate voltages between 10V and -10V.

See also: AXIS, DAC, OUTLIMIT, SERVO

Example:

```
>> PRINT DAC_OUT AXIS(3)
288.0000
```

5-3-46 DATUM

Type: Motion Control Command

Syntax: `DATUM(`*sequence*`)`

Description: DATUM performs one of 6 origin search sequences to position an axis to an absolute position and also can be used to reset the following errors. DATUM uses both the creep speed and the demand speed for the origin searches. The creep speed in the sequences is set using the CREEP axis parameter and the demand speed is set using the SPEED axis parameter. The datum switch input number, which is used for sequences 3 to 7, is selected by the DATUM_IN parameter.

DATUM works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Arguments: ***sequence***

- 0 The currently measured position is set as the demand position for all axes. DATUM(0) will also reset a following error in the AXISSTATUS registers for the axes.
- 1 The axis moves at creep speed forward until the Z marker is encountered. The demand position is then reset to zero and the measured position is corrected to maintain the following error.
- 2 The axis moves at creep speed reverse until the Z marker is encountered. The demand position is then reset to zero and the measured position is corrected to maintain the following error.
- 3 The axis moves at the demand speed forward until the datum switch is reached. The axis then moves reverse at creep speed until the datum switch is reset. The demand position is then reset to zero and the measured position corrected so as to maintain the following error.
- 4 The axis moves at the demand speed reverse until the datum switch is reached. The axis then moves forward at creep speed until the datum switch is reset. The demand position is then reset to zero and the measured position corrected so as to maintain the following error.

5 The axis moves at demand speed forward until the datum switch is reached. The axis then reverses at creep speed until the Z marker is encountered. The demand position is then reset to zero and the measured position corrected so as to maintain the following error.

6 The axis moves at demand speed reverse until the datum switch is reached. The axis then moves forward at creep speed until the Z marker is encountered. The demand position is then reset to zero and the measured position corrected so as to maintain the following error.

Precautions: The origin input set with the DATUM_IN parameter is active low, i.e., the origin switch is set when the input is OFF. The feedhold, reverse jog, forward jog, forward and reverse limit inputs are also active low. Active low inputs are used to enable fail-safe wiring.

See also: ACCEL, AXIS, CREEP, DATUM_IN, DECEL, SPEED

5-3-47 DATUM_IN

Type: Axis Parameter

Alternative: DAT_IN

Description: The DATUM_IN parameter contains the input number to be used as the datum switch input for the DATUM command. The input number is valid from 0 to 15 and from 20 to 31. If DATUM_IN is set to –1, then no input is used as the datum switch input.

Precautions: The origin input is active low, i.e., the origin switch is set when the input is OFF. The feedhold, reverse jog, forward jog, forward and reverse limit inputs are also active low. Active low inputs are used to enable fail-safe wiring.

See also: AXIS, DATUM

Example:
```
DATUM_IN AXIS(0) = 12
```

5-3-48 DECEL

Type: Axis Parameter

Description: DECEL contains the axis deceleration rate. The rate is in units/s^2. The parameter can have any positive value including zero.

See also: ACCEL, AXIS, UNITS

Example:
```
DECEL = 100                          'Set deceleration rate
PRINT " Deceleration rate is ";DECEL;" mm/s/s"
```

5-3-49 DEFPOS

Type: Motion Control Command

Syntax: DEFPOS(*pos_1*[, *pos_2*[, *pos_3*[, *pos_4*[, *pos_5*[, *pos_6*[, *pos_7*[, *pos_8*]]]]]]])

Alternative: DP(*pos_1*[, *pos_2*[, *pos_3*,[*pos_4*[, *pos_5*[, *pos_6*[, *pos_7*[, *pos_8*]]]]]])

Description: DEFPOS defines the current demand position (DPOS) as a new absolute position. The measured position (MPOS) will be changed accordingly in order to keep the following error. DEFPOS is typically used after an origin search sequence (see DATUM command), as these set the current position to zero. DEFPOS can be used at any time. See also OFFPOS which can be used to perform a relative adjustment of the current position.

DEFPOS works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: Changes to the axis positions made via DEFPOS are made on the next servo update. This can potentially cause problems when a move is initiated in the same servo period as the DEFPOS. OFFPOS can be used to avoid this problem.

For example, the following sequence could easily fail to move to the correct position because DEFPOS will not have been completed when MOVEABS is loaded.

```
DEFPOS(100)
MOVEABS(0)
```

DEFPOS statements are internally converted into OFFPOS position offsets, which provides an easy way to avoid the problem by programming as follows:

```
DEFPOS(100)
WAIT UNTIL OFFPOS = 0
MOVEABS(0)
```

Arguments: ***pos_i***
The absolute position for (base+i) axis in user units. Refer to the BASE command for the grouping of the axes.

See also: AXIS, DATUM, DPOS, OFFPOS, MPOS, UNITS

Example: The last line defines the current position to (–1000,–3500) in user units. The current position would have been reset to (0,0) by the two DATUM commands.

```
BASE(2)
DATUM(5)
BASE(1)
DATUM(4)
WAIT IDLE
DEFPOS(-1000,-3500)
```

5-3-50 DEL

Type: Program Command

Syntax: `DEL` [“*program_name”*]

Alternative: `RM` [“*program_name”*]

Description: DEL deletes a program from memory. DEL without a program name can be used to delete the currently selected program (using SELECT). The program name can also be specified without quotes. DEL ALL will delete all programs.

DEL can also be used to delete the Table as follows:

```
DEL “TABLE”
```

The name “TABLE” must be in quotes.

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can select the command from the Program menu.

Arguments: ***program_name***
Name of the program to be deleted.

See also: COPY, NEW, RENAME, SELECT, TABLE

Example:

```
>> DEL oldprog
```

5-3-51 DEMAND_EDGES

Type: Axis Parameter

Description: DEMAND_EDGES contains the current value of the DPOS axis parameter in encoder edge units.

See also: AXIS, DPOS

Note This parameter is read-only.

5-3-52 DIR

Type: Program Command

Syntax: `DIR`

Alternative: `LS`

Description: DIR displays a list of the programs held in memory, their memory size and their RUNTYPE. Furthermore the controller's available memory size, power up mode and current selected program is displayed.

See also: FREE, POWER_UP, PROCESS, RUNTYPE, SELECT

5-3-53 DISPLAY

Type: System Parameter

Description: DISPLAY selects the bank of 8 in- or outputs which is displayed by the IO status LED indicators on the front panel of the MC Unit. The following values are valid:

0	Inputs 0 to 7	First bank of 8 physical inputs.
1	Inputs 8 to 15	Second bank of 8 physical inputs.
2	Inputs 16 to 23	Inputs16 to 19 are driver error inputs for axes 0 to 3. Inputs 20 to 23 are virtual inputs.
3	Inputs 24 to 31	Virtual inputs.
4	Outputs 0 to 7	Not existing on C200HW-MC402-E.
5	Outputs 8 to 15	Bank of 8 physical outputs.
6	Outputs 16 to 23	Output 16 is driver error reset for all drivers. Outputs 17 to 19 do not exist. Outputs 20 to 23 are virtual outputs.
7	Outputs 24 to 31	Virtual outputs.

See also: IN, OP

Example: The first line in the following example sets the indicators to show status of the physical outputs. The second line will cause the top left indicator to light.

```
DISPLAY = 5
OP(8,ON)
```

5-3-54 DPOS

Type: Axis Parameter

Description: DPOS contains the demand position in user units, which is generated by the move commands in servo control. When the controller is in open loop (SERVO=OFF), the measured position (MPOS) will be copied to the DPOS in order to maintain a zero following error.

The range of the demand position is controlled with the REP_DIST and REP_OPTION axis parameters. The value can be adjusted without doing a move by using the DEFPOS command or OFFPOS axis parameter. DPOS is reset to zero at start-up.

Note This parameter is read-only.

See also: AXIS, DEFPOS, DEMAND_EDGES, FE, MPOS, REP_DIST, REP_OPTION, OFFPOS, UNITS

Example:

```
>> PRINT DPOS AXIS(0)
34.0000
```

The above line will return the demand position in user units.

5-3-55 EDIT

Type: Program Command

Syntax: `EDIT` [*line_number*]

Alternative: `ED` [*line_number*]

Description: EDIT starts the built in screen editor allowing a program in the MC Unit memory to be modified using a VT100 Terminal. The currently selected program will be edited.

The editor commands are as follows:

Quit Editor	[CTRL] K and D
Delete Line	[CTRL] Y

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can select the command from the Program menu.

Arguments: ***line_number***
The number of the line at which to start editing.

See also: SELECT

5-3-56 ENCODER

Type: Axis Parameter

Description: ENCODER contains a raw copy of the encoder. For the servo axis a 12-bit (modulo 4095) number is used.

The MPOS axis parameter contains the measured position calculated from the ENCODER value automatically, allowing for overflows and offsets.

Note This parameter is read-only.

See also: AXIS, MPOS

5-3-57 ENDMOVE

Type: Axis Parameter

Description: ENDMOVE holds the position of the end of the current move in user units. If the SERVO axis parameter is ON, the ENDMOVE parameter can be written to produce a step change in the demand position (DPOS).

Note As the measured position is not changed initially, the following error limit (FE_LIMIT) should be considered. If the change of demanded position is too big, the following error will be exceeded.

See also: AXIS, DPOS, FE_LIMIT, UNITS

5-3-58 EPROM

Type: Program Command

Syntax: `EPROM`

Description: EPROM stores the BASIC programs in the MC Unit in the flash EPROM. This information is retrieved at start-up if the POWER_UP parameter has been set to 1.

Precautions: Motion Perfect offers this command as a button on the control panel.

See also: POWER_UP, RUNTYPE

5-3-59 ERROR_AXIS

Type: System Parameter

Description: ERROR_AXIS contains the number of the axis which has caused a motion error.

A motion error occurs when the AXISSTATUS state for one of the axes matches the ERRORMASK setting. In this case the enable relay (WDOG) will be turned OFF, the MOTION_ERROR parameter will have value 1 and the ERROR_AXIS parameter will contain the number of the first axis to have the error.

Note This parameter is read-only.

See also: AXISSTATUS, ERRORMASK, MOTION_ERROR, WDOG

5-3-60 ERROR_LINE

Type: Task Parameter

Description: The ERROR_LINE parameter contains the number of the line which caused the last BASIC run-time error in the program task. This value is only valid when the BASICERROR parameter is TRUE.

Each task has its own ERROR_LINE parameter. Use the PROC modifier to access the parameter for a certain task. Without PROC the current task will be assumed.

Note This parameter is read-only.

See also: BASICERROR, PROC, RUN_ERROR

Example:
```
>> PRINT ERROR_LINE PROC(4)
23
```

5-3-61 ERRORMASK

Type: Axis Parameter

Description: ERRORMASK contains a mask value that is ANDed bit by bit with the AXISSTATUS axis parameter on every servo cycle to determine if a motion error has occurred.

When a motion error occurs the enable relay (WDOG) will be turned OFF, the MOTION_ERROR parameter will have value 1 and the ERROR_AXIS parameter will contain the number of the first axis to have the error.

Check the AXISVALUES parameter for the status bit allocations. The default setting of ERRORMASK is 256. The enable relay will only be turned OFF if the following error exceeds its limits.

⚠ **Caution** It is up to the user to define in which cases the enable relay needs to be disabled. For safe operation it is strongly suggested to disable the enable relay when the following error has exceed its limit in all cases. This is done by setting bit 8 of ERRORMASK (ERRORMASK=ERRORMASK OR 256).

See also: AXIS, AXISSTATUS, ERROR_AXIS, MOTION_ERROR, WDOG

5-3-62 EXP

Type: Mathematical Function

Syntax: EXP(*expression*)

Description: EXP returns the exponential value of the *expression*.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> print exp(1.0)
2.7183
```

5-3-63 FALSE

Type: Constant

Description: FALSE returns the numerical value 0.

Note A constant is read-only.

Example:
```
test:
   res = IN(0) OR IN(2)
   IF res = FALSE THEN
      PRINT "Inputs are off"
   ENDIF
```

5-3-64 FAST_JOG

Type: Axis Parameter

Description: FAST_JOG contains the input number to be used as the fast jog input. The number can be from 0 to 15 and from 20 to 31. If FAST_JOG is set to –1, then no input is used for the fast jog.

The fast jog input controls the jog speed between two speeds. If the fast jog input is set, the speed as given by the SPEED axis parameter will be used for jogging. If the input is not set, the speed given by the JOGSPEED axis parameter will be used.

Note This input is active low.

See also: AXIS, FWD_JOG, JOGSPEED, REV_JOG, SPEED

5-3-65 FE

Type: Axis Parameter

Description: FE is the position error in user units, and is equal to the demand position in the DPOS axis parameter minus the measured position in the MPOS axis parameter. The value of the following error can be monitored by using the axis parameters FE_LIMIT and FE_RANGE.

Note This parameter is read-only.

See also: AXIS, DPOS, FE_LIMIT, FE_RANGE, MPOS, UNITS

5-3-66 FE_LIMIT

Type: Axis Parameter

Alternative: `FELIMIT`

Description: FE_LIMIT contains the maximum allowable following error in user units. When exceeded, bit 8 of the AXISSTATUS parameter of the axis will be set.

Depending on the value of ERRORMASK, the MC Unit will generate a motion error and reset the enable relay (WDOG).

This limit can be used to guard against fault conditions, such as mechanical lock-up, loss of encoder feedback, etc.

See also: AXIS, AXISSTATUS, ERRORMASK, FE, FE_RANGE, UNITS

5-3-67 FE_RANGE

Type: Axis Parameter

Alternative: `FERANGE`

Description: FE_RANGE contains the following error report range in user units. When the following error exceeds this value on a servo axis, bit 1 in the AXISSTATUS axis parameter will be turned ON.

See also: AXIS, AXISSTATUS, ERRORMASK, FE, FE_LIMIT, UNITS

5-3-68 FHOLD_IN

Type: Axis Parameter

Alternative: FH_IN

Description: FHOLD_IN contains the input number to be used as the feedhold input. The number can be from 0 to 15 and from 20 to 31. If FHOLD_IN is set to –1, then no input is used as a feedhold input.

If an input number is set and the feedhold input turns ON, the speed of the move on the axis is changed to the value set in the FH_SPEED axis parameter. The current move is NOT cancelled. Furthermore, bit 7 of the AXISSTATUS parameter is set. When the input turns OFF again, any move in progress when the input was set will return to the programmed speed.

Precautions: This feature only works on speed controlled moves. Moves which are not speed controlled (CAMBOX, CONNECT and MOVELINK) are not affected.

Note This input is active low.

See also: AXIS, AXISSTATUS, FHSPEED

5-3-69 FHSPEED

Type: Axis Parameter

Description: FHSPEED contains the feedhold speed. This parameter can be set to a value in user units/s at which speed the axis will move when the feed-hold input turns on. The current move is not cancelled. FHSPEED can have any positive value including zero. The default value is zero.

Precautions: This feature only works on speed controlled moves. Moves which are not speed controlled (CAMBOX, CONNECT and MOVELINK) are not affected.

See also: AXIS, FHOLD_IN, UNITS

5-3-70 FOR TO STEP NEXT

Type: Structural Command

Syntax: FOR *variable* = *start* TO *end* [STEP *increment*]
<commands>
NEXT *variable*

Description: The FOR ... NEXT loop allows the program segment between the FOR and the NEXT statement to be repeated a number of times.

On entering this loop, the *variable* is initialized to the value of *start* and the block of commands is then executed. Upon reaching the NEXT command, the *variable* is increased by the *increment* specified after STEP. The STEP value can be positive or negative, if omitted the value is assumed to be 1.

If *variable* is less than or equal to *end,* then the block of commands is repeatedly executed until *variable* is greater than *end,* at which time the command after NEXT is executed.

Precautions: FOR ... NEXT statements can be nested up to 8 levels deep in a BASIC program.

Arguments: ***variable***
Any valid BASIC expression.
start
Any valid BASIC expression.
end
Any valid BASIC expression.

increment
Any valid BASIC expression.

See also: REPEAT, WHILE

Examples: **Example 1**

The following loop turns ON outputs 0 to 10.

```
FOR opnum = 0 TO 10
   OP(opnum,ON)
NEXT opnum
```

Example 2

The STEP increment can be positive or negative.

```
loop:
   FOR dist = 5 TO -5 STEP -0.25
      MOVEABS(dist)
      GOSUB pick_up
   NEXT dist
```

Example 3

FOR...NEXT statements can be nested (up to 8 levels deep) provided the inner FOR and NEXT commands are both within the outer FOR...NEXT loop:

```
loop1:
   FOR l1 = 1 TO 8
loop2:
      FOR l2 = 1 TO 6
         MOVEABS(l1*100,l2*100)
         GOSUB 1000
      NEXT l2
   NEXT l1
```

5-3-71 FORWARD

Type: Motion Control Command

Syntax: `FORWARD`

Alternative: `FO`

Description: FORWARD moves an axis continuously forward at the speed set in the SPEED parameter.

FORWARD works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: The forward motion can be stopped by executing the CANCEL or RAPIDSTOP command, or by reaching the forward, inhibit, or origin return limit.

See also: AXIS, CANCEL, RAPIDSTOP, REVERSE, UNITS

Example:

```
start:
   FORWARD
   WAIT UNTIL IN(0) = ON   'Wait for stop signal
   CANCEL
```

5-3-72 FRAC

Type: Mathematical Function

Syntax: `FRAC(`*expression*`)`

Description: FRAC returns the fractional part of the *expression*.

Arguments: ***expression***
Any valid BASIC expression.

Example:

```
>> PRINT FRAC(1.234)
0.2340
```

5-3-73 FREE

Type: System Function

Syntax: FREE

Description: FREE returns the remaining amount of memory available for user programs and Table array elements.

Precautions: Each line takes a minimum of 4 characters (bytes) in memory. This is for the length of this line, the length of the previous line, number of spaces at the beginning of the line and a single command token. Additional commands need one byte per token; most other data is held as ASCII.

The MC Unit compiles programs before they are executed, this means that twice the memory is required to be able to execute a program.

Example:
```
>> PRINT FREE
47104.0000
```

5-3-74 FS_LIMIT

Type: Axis Parameter

Alternative: FSLIMIT

Description: FS_LIMIT contains the absolute position of the forward software limit in user units.

A software limit for forward travel can be set from the program to control the working envelope of the machine. When the limit is reached, the MC Unit will decelerate to zero, and then cancel the move. Bit 9 of the AXISSTATUS axis parameter will be turned ON when the axis position is greater than FS_LIMIT.

See also: AXIS, AXISSTATUS, RS_LIMIT, UNITS

5-3-75 FWD_IN

Type: Axis Parameter

Description: FWD_IN contains the input number to be used as a forward limit input. The number can be from 0 to 15 and from 20 to 31. If FWD_IN is set to –1, then no input is used as a forward limit.

If an input number is set and the limit is reached, any forward motion on that axis will be stopped. Bit 4 of the AXISSTATUS will also be set.

Note This input is active low.

See also: AXIS, AXISSTATUS, REV_IN

5-3-76 FWD_JOG

Type: Axis Parameter

Description: FWD_JOG contains the input number to be used as a jog forward input. The input can be from 0 to 15 and from 20 to 31. If FWD_JOG is set to –1 (default), then no input is used as a forward jog input.

Note This input is active low.

See also: AXIS, FAST_JOG, JOGSPEED, REV_JOG

5-3-77 GET

Type: I/O Function

Syntax: GET#*n, variable*

Description: GET waits for the arrival of a single character and assigns the ASCII code of the character to a variable. The BASIC program will wait until a character is

available. Channels 5 to 7 are logical channels that are superimposed on the RS-232C programming port when using Motion Perfect.

Arguments:

n

The specified input device.

0	RS-232C programming port A
1	RS-232C serial port B
5	Motion Perfect port A user channel 5
6	Motion Perfect port A user channel 6
7	Motion Perfect port A user channel 7

variable

The name of the variable to receive the ASCII code.

Precautions: Channel 0 is particularly used for the connection to Motion Perfect and/or the command line interface. Please be aware that this channel may give problems for this function.

See also: INPUT, KEY, LINPUT

Example: The following line can be used to store the ASCII character received on the Motion Perfect port channel 5 in k.

```
GET#5, k
```

5-3-78 GOSUB RETURN

Type: Structural Command

Syntax: `GOSUB` *label ...* `RETURN`

Description: GOSUB enables a subroutine jump. GOSUB stores the position of the line after the GOSUB command and then jumps to the specified *label*. Upon reaching the RETURN statement, program execution is returned to the stored position.

Precautions: Subroutines on each task can be nested up to 8 levels deep.

Arguments:

label

A valid label that occurs in the program. If the label does not exist, an error message will be displayed during structure checking at the beginning of execution and program execution will be halted.

See also: GOTO

Example:

```
main:
   GOSUB routine
   GOTO main
routine:
   PRINT "Measured position=";MPOS;CHR(13);
   RETURN
```

5-3-79 GOTO

Type: Structural Command

Syntax: `GOTO` *label*

Description: GOTO enables jump of program execution. GOTO jumps program execution to the line of the program containing the *label*.

Arguments:

label

A valid label that occurs in the program. If the label does not exist, an error message will be displayed during structure checking at the beginning of execution and program execution will be halted.

See also: GOSUB

Example:
```
loop:
   PRINT "Measured position = ";MPOS;CHR(13);
   GOTO loop
```

5-3-80 HALT

Type: System Command

Syntax: HALT

Description: HALT stops execution of all programs currently running. The command can be used both on command line as in programs. Refer to the STOP command to stop a single program.

See also: PROCESS, STOP

5-3-81 I_GAIN

Type: Axis Parameter

Description: I_GAIN contains the integral gain for the axis. The integral output contribution is calculated by multiplying the sums of the following errors with I_GAIN. The default value is zero.

Adding integral gain to a servo system reduces positioning error when at rest or moving steadily. It can produce or increase overshooting and oscillation and is therefore only suitable for systems working on constant speed and with slow accelerations.

See section *1-4-2 Servo System Principles* for more details.

Precautions: In order to avoid any instability the servo gains should be changed only when the SERVO is OFF.

See also: D_GAIN, OV_GAIN, P_GAIN, VFF_GAIN

5-3-82 IF THEN ELSE ENDIF

Type: Structural Command

Syntax:
IF *condition* THEN
    <*commands*>
[ELSE
    <*commands*>]
ENDIF
IF *condition* THEN <commands>

Description: IF controls the flow of the program based on the results of the *condition*. If the *condition* is TRUE the commands following THEN up to ELSE (or ENDIF if not included) will be executed. If the condition is FALSE the commands following ELSE will be executed or the program will resume at the line after ENDIF in case no ELSE is included. The ENDIF is used to mark the end of the conditional block.

IF evaluates the *condition.* If it is true, the commands following it will be executed. If it is not true, the commands are skipped. If *condition* is FALSE and an ELSE command sequence is specified, then the ELSE command sequence is executed.

Precautions: IF...THEN [ELSE] ENDIF sequences can be nested without limit. For a multi-line IF...THEN construction, there must not be any statement after THEN. A single-line construction must not use ENDIF.

Arguments:

condition
Any logical expression.

commands
Any valid BASIC commands.

Examples:

Example 1

```
IF MPOS > (0.22 * VR(0)) THEN GOTO exceeds_length
```

Example 2

```
IF IN(0) = ON THEN
   count = count + 1
   PRINT "COUNTS = ";VR(1)
   fail = 0
ELSE
   fail = fail + 1
ENDIF
```

5-3-83 IN

Type: I/O Function

Syntax:

`IN(`*input_number*[`,`*final_input_number*]`)`
`IN`

Description:

The IN function returns the value of digital inputs.

- IN(*input_number*, *final_input_number*) will return the binary sum of the group of inputs. The two arguments must be less than 24 apart.
- IN(*input_number*) with the value for input_number less than 32 will return the value of the particular channel.
- IN (without arguments) will return the binary sum of the first 24 inputs (as IN(0,23)).

Check *Accessing I/O* for a description of the various input types.

Arguments:

input_number.
The number of the input for which to return a value. Value: An integer between 0 and 31.

final_ input_number.
The number of the last input for which to return a value. Value: An integer between 0 and 31.

See also: DISPLAY, OP

Examples:

Example 1

The following lines can be used to move to the position set on a thumbwheel multiplied by a factor. The thumbwheel is connected to inputs 4, 5, 6 and 7, and gives output in BCD.

```
moveloop:
   MOVEABS(IN(4,7)*1.5467)
   WAIT IDLE
   GOTO moveloop
```

The MOVEABS command is constructed as follows:

Step 1: IN(4,7) will get a number between 0 and 15.

Step 2: The number is multiplied by 1.5467 to get required distance.

Step 3: An absolute move is made to this position.

Example 2

In this example a single input is tested:

```
test:
   WAIT UNTIL IN(4)=ON    'Conveyor is in position when ON
   GOSUB place
```

5-3-84 INITIALISE

Type: System Command

Syntax: `INITIALISE`

Description: INITIALISE sets all axis parameters to their default values for all axes. The parameters are also reset each time the MC Unit is started.

See also: EX

5-3-85 INPUT

Type: I/O Command

Syntax: `INPUT#`*n*, *variable*{, *variable*}

Description: The INPUT command will wait for a string to be received and will take the numerical value of this string which is terminated with a carriage return <CR>. The value of a valid string is assigned to the specified variable. If the string is invalid, the user will be prompted with an error message and the task will be repeated. Multiple inputs may be requested on one line, separated by commas, or on multiple lines, separated by carriage return. The maximum amount of inputs on one line has no limit other than the line length.

Channels 5 to 7 are logical channels that are superimposed on the RS-232C programming port when using Motion Perfect.

Arguments: ***n***

The specified input device.

0	RS-232C programming port A
1	RS-232C serial port B
5	Motion Perfect port A user channel 5
6	Motion Perfect port A user channel 6
7	Motion Perfect port A user channel 7

variable

The variable to write to.

Precautions: Channel 0 is particularly used for the connection to Motion Perfect and/or the command line interface. Please be aware that this channel may give problems for this function.

See also: GET, LINPUT

Example: Consider the following program to receive data from the terminal.

```
INPUT#5, num
PRINT#5, "BATCH COUNT=";num[0]
```

A possible response on the terminal could be:

```
123<CR>
BATCH COUNT=123
```

5-3-86 INT

Type: Mathematical Function

Syntax: `INT(`*expression*`)`

Description: INT returns the integer part of the *expression*.

Note To round a positive number to the nearest integer value take the INT function of the value added by 0.5. Similarly, to round for a negative value subtract 0.5 to the value before applying INT.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT INT(1.79)
1.0000
```

5-3-87 JOGSPEED

Type: Axis Parameter

Description: JOGSPEED sets the jog speed in user units for an axis. A jog will be performed when a jog input for an axis has been declared and that input is low. A forward jog input and a reverse jog input are available for each axis, respectively set by FWD_JOG and REV_JOG. The speed of the jog can be controlled with the FAST_JOG input.

See also: AXIS, FAST_JOG, FWD_JOG, REV_JOG, UNITS

5-3-88 KEY

Type: I/O Parameter

Syntax: KEY#*n*

Description: The KEY command returns TRUE or FALSE depending on if a character has been received on an input device or not. This command does not read the character but allows the program to test if any character has arrived. A TRUE result will reset when the character is read with GET.
Channels 5 to 7 are logical channels that are superimposed on the RS-232C programming port when using Motion Perfect.

Argument: ***n***
The specified input device.

0	RS-232C programming port A
1	RS-232C serial port B
5	Motion Perfect port A user channel 5
6	Motion Perfect port A user channel 6
7	Motion Perfect port A user channel 7

Precautions: Channel 0 is reserved for the connection to Motion Perfect and/or the command line interface. Please be aware that this channel may give problems for this function.

See also: GET

Example:
```
WAIT UNTIL KEY#1
GET#1, k
```
Beware that for using KEY#1 in an equation may require parentheses in the statement, in this case: `WAIT UNTIL (KEY#1)=TRUE`.

5-3-89 LAST_AXIS

Type: System Parameter

Description: LAST_AXIS contains the number of the last axis processed by the system.
Most systems do not use all the available axes. It would therefore be a waste of time to task the idle moves on all axes that are not in use. To avoid this to some extent, the MC Unit will task moves on the axes from 0 to LAST_AXIS, where LAST_AXIS is the number of the highest axis for which an AXIS or BASE command has been processed, whichever of the two is larger.

Note This parameter is read-only.

See also: AXIS, BASE

5-3-90 LINPUT

Type: I/O Command

Syntax: `LINPUT#`*n*, *vr_variable*

Description: The LINPUT command waits for an input string and stores the ASCII values of the string in an array of variables starting at the specified VR variable. The string must be terminated with a carriage return <CR>, which is also stored. The string is not echoed by the controller.

Channels 5 to 7 are logical channels that are superimposed on the RS-232C programming port when using Motion Perfect.

Arguments: ***n***

The specified input device.

0	RS-232C programming port A
1	RS-232C serial port B
5	Motion Perfect port A user channel 5
6	Motion Perfect port A user channel 6
7	Motion Perfect port A user channel 7

vr_variable

The first VR-variable to write to.

Precautions: Channel 0 is particularly used for the connection to Motion Perfect and/or the command line interface. Please be aware that this channel may give problems for this function.

See also: GET, INPUT,VR

Example: Consider the following line in a program.

```
LINPUT#5, VR(0)
```

Entering START<CR> will give

VR(0)=83	S
VR(1)=84	T
VR(2)=65	A
VR(3)=82	R
VR(4)=84	T
VR(5)=13	<CR>

5-3-91 LIST

Type: Program Command

Syntax: `LIST` [“*program_name*”]

Alternative: `TYPE` [“*program_name*”]

Description: The LIST command prints the current selected program or the program specified by *program_name*. The program name can also be specified without quotes. If the program name is omitted, the current selected program will be listed.

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can use the Program Editor.

Arguments: ***program_name***

The program to be printed.

See also: SELECT

5-3-92 LN

Type: Mathematical Function

Syntax: `LN(`*expression*`)`

Description: LN returns the natural logarithm of the *expression*. The input expression value must be greater than zero.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT LN(10)
2.3026
```

5-3-93 LOCK

Type: System Command

Syntax: `LOCK(`*code*`)`
`UNLOCK(`*code*`)`

Description: The LOCK command prevents the program from being viewed or modified by personnel unaware of the security code. The UNLOCK command allows the locked state to be unlocked. The code number can be any integer and is held in encoded form.

Precautions: LOCK is always an immediate command and can be issued only when the system is UNLOCKED. The security code must be remembered; it will be required to unlock the system.

Arguments: ***code***
Any valid integer with maximum 7 digits.

Example:
```
>> LOCK(561234)
```
The program cannot now be modified or seen.
```
>> UNLOCK(561234)
```
The system is now unlocked.

5-3-94 MARK

Type: Axis Parameter

Description: MARK contains TRUE when a registration event has occurred to indicate that the value in the REG_POS axis parameter is valid. MARK is set to FALSE when REGIST command has been executed and is set to TRUE when the register event occurs.

Note This parameter is read-only.

See also: AXIS, REG_POS, REGIST

5-3-95 MERGE

Type: Axis Parameter

Description: MERGE is a software switch that can be used to enable or disable the merging of consecutive moves. With MERGE is ON and the next move already in the next move buffer (NTYPE), the axis will not ramp down to zero speed but will load up the following move enabling a seamless merge. All non-zero values for MERGE are also considered ON.

⚠ **Caution** It is up to the programmer to ensure that merging is sensible. For example, merging a forward move with a reverse move will cause an attempted instantaneous change of direction.

MERGE will only function if the following are all true.

1. Only the speed profiled moves MOVE, MOVEABS, MOVECIRC, MHELICAL and MOVEMODIFY can be merged with each other.
2. There is a move in the next move buffer (NTYPE).
3. The axis group does not change for multi-axis moves.

When merging multi-axis moves, only the base axis MERGE axis parameter needs to be set.

Precautions: If the moves are short, a high deceleration rate must be set to avoid the MC Unit decelerating in anticipation of the end of the buffered move.

See also: AXIS

Example:
```
MERGE = OFF       'Decelerate at the end of each move
MERGE = ON        'Moves will be merged if possible
```

5-3-96 MHELICAL

Type: Motion Control Command

Syntax: `MHELICAL(`*end_1*, *end_2*, *centre_1*, *centre_2*, *direction*, *distance_3*`)`

Alternative: `MH(`*end_1*, *end_2*, *centre_1*, *centre_2*, *direction*, *distance_3*`)`

Description: MHELICAL performs an interpolated helical move by moving two orthogonal axes in a circular arc with a simultaneous linear move on a third axis. The path of the movement is determined by the 6 arguments, which are incremental from the starting position.

The first 5 arguments are the same as those of the MOVECIRC command. The arguments *end_1* and *centre_1* apply to the base axis and *end_2* and *centre_2* apply to the following axis. The linear mode is defined by *distance_3*. All arguments are given in user units of each axis. The speed of the movement along the circular arc is set by the SPEED, ACCEL and DECEL parameters of the base axis.

MHELICAL works on the default basis axis group (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: Both MHELICAL and MOVECIRC compute the nominal radius and the angle of movement from the centre and end point. If the endpoint does not lie on the calculated path, the move simply ends at the computed end and not the specified end point. It is the responsibility of the programmer to ensure that the two points correspond to correct points on a helix.

For MOVECIRC to be correctly executed, the two axes moving in the circular arc must have the same number of encoder pulses per linear axis distance. If they do not, it is possible to adjust the encoder scales in many cases by adjusting with PP_STEP axis parameters for the axis.

Arguments:

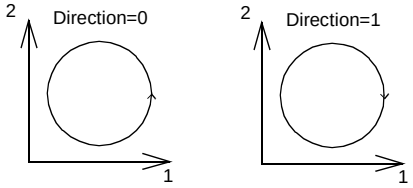
end_1
The end position for the base axis.

end_2
The end position for the next axis.

centre_1
The centre position around which the base axis is to move.

centre_2
The centre position around which the next axis is to move.

direction
A software switch that determines whether the arc is interpolated in a clockwise or counterclockwise direction. Value: 0 or 1.



If the two axes involved in the movement form a right-hand axis, set *direction* to 0 to produce positive motion about the third (possibly imaginary) orthogonal axis.

If the two axes involved in the movement form a left-hand axis, set *direction* to 0 to produce negative motion about the third (possibly imaginary) orthogonal axis.

Direction	Right-hand axis	Left-hand axis
0	Positive	Negative
1	Negative	Positive

distance_3
The distance to move on the third axis.

See also: AXIS, MOVECIRC, PP_STEP, UNITS

5-3-97 MOD

Type: Mathematical Function

Syntax: *expression_1* `MOD` *expression_2*

Description: MOD returns the *expression_2* modulus of *expression_1*. This function will take the integer part of any non-integer input.

Arguments:
expression_1
Any valid BASIC expression.
expression_2
Any valid BASIC expression.

Example:
```
>> PRINT 122 MOD 13
5.0000
```

5-3-98 MOTION_ERROR

Type: System Parameter

Description: MOTION_ERROR contains an error flag for axis motion errors. The parameter will have value 1 when a motion error has occurred.

A motion error occurs when the AXISSTATUS state for one of the axes matches the ERRORMASK setting. In this case the enable relay (WDOG) will be turned OFF, the MOTION_ERROR parameter will have value 1 and the ERROR_AXIS parameter will contain the number of the first axis to have the error.

A motion error can be cleared executing a DATUM(0) command.

Note This parameter is read-only.

See also: AXISSTATUS, DATUM, ERROR_AXIS, ERRORMASK, WDOG

5-3-99 MOVE

Type: Motion Control Command

Syntax: MOVE(*dist_1*[, *dist_2*[, *dist_3*[, *dist_4*[, *dist_5*[, *dist_6*[, *dist_7*, *dist_8*]]]]]])

Alternative: MO(*dist_1*[, *dist_2*[, *dist_3*[, *dist_4*[, *dist_5*[, *dist_6*[, *dist_7*, *dist_8*]]]]]])

Description: MOVE moves with one or more axes at the demand speed and acceleration and deceleration to a position specified as increment from the current position. In multi-axis moves the movement is interpolated and the speed, acceleration and deceleration are taken from the base axis.

The specified distances are scaled using the unit conversion factor in the UNITS axis parameter. If, for example, an axis has 4,000 encoder edges/mm, then the number of units for that axis would be set to 4000, and MOVE(12.5) would move 12.5 mm.

MOVE works on the default basis axis group (set with BASE) unless AXIS is used to specify a temporary base axis. Argument *dist_1* is applied to the base axis, *dist_2* is applied to the next axis, etc. By changing the axis between individual MOVE commands, uninterpolated, unsynchronised multi-axis motion can be achieved. Incremental moves can be merged for profiled continuous path movements by turning ON the MERGE axis parameter.

Considering a 4-axis movement, the individual speeds are calculated using the equations below. Given command MOVE(x_1, x_2, x_3, x_4) and the profiled speed v_p as calculated from the SPEED, ACCEL and DECEL parameters from the base axis and the total multi-axes distance L.

$$L = \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2}.$$

The individual speed v_i for axis i at any time of the movement is calculated as

$$v_i = \frac{x_i \cdot v_p}{L}.$$

Arguments: ***dist_i***

The distance to move for every axis *i* in user units starting with the base axis.

See also: AXIS, MOVEABS, UNITS

Examples: **Example 1**

A system is working with a unit conversion factor of 1 and has a 1000-line encoder. It is, therefore, necessary to use the following command to move 10 turns on the motor. (A 1000 line encoder gives 4000 edges/turn).

```
MOVE(40000)
```

Example 2

In this example, axes 1, 2 and 3 are moved independently (without interpolation). Each axis will move at its programmed speed and other axis parameters.

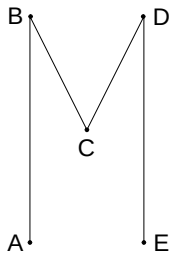
```
MOVE(10) AXIS(1)
MOVE(10) AXIS(2)
MOVE(10) AXIS(3)
```

Example 3

An X-Y plotter can write text at any position within its working envelope. Individual characters are defined as a sequence of moves relative to a start point so that the same commands can be used no matter what the plot position. The command subroutine for the letter “m” might be as follows:

```
m:
MOVE(0,12)      'move A -> B
MOVE(3,-6)      'move B -> C
```

```
MOVE(3,6)       'move C -> D
MOVE(0,-12)     'move D -> E
```



5-3-100 MOVEABS

Type: Motion Control Command

Syntax: MOVEABS(*pos_1*[,*pos_2*[,*pos_3*[,*pos_4[,pos_5*[,*pos_6*[,*pos_7*[,*pos_8*]]]]]]])

Alternative: MA(*pos_1*[,*pos_2*[,*pos_3*[,*pos_4[,pos_5*[,*pos_6*[,*pos_7*[,*pos_8*]]]]]]])

Description: MOVEABS moves one or more axes at the demand speed, acceleration and deceleration to a position specified as absolute position, i.e., in reference to the origin. In multi-axis moves the movement is interpolated and the speed, acceleration and deceleration are taken from the base axis.

The specified distances are scaled using the unit conversion factor in the UNITS axis parameter. If, for example, an axis has 4,000 encoder edges/mm, then the number of units for that axis would be set to 4000, and MOVEABS(12.5) would move to a position 12.5 mm from the origin.

MOVEABS works on the default basis axis group (set with BASE) unless AXIS is used to specify a temporary base axis. Argument *pos_1* is applied to the base axis, *pos_2* is applied to the next axis, etc. By changing the axis between individual MOVE commands, uninterpolated, unsynchronised multi-axis motion can be achieved. Absolute moves can be merged for profiled continuous path movements by turning ON the MERGE axis parameter.

Considering a 4-axis movement, the individual speeds are calculated using the equations below. Given command MOVE(ax_1, ax_2, ax_3, ax_4), the current position (ay_1, ay_2, ay_3, ay_4) and the profiled speed v_p as calculated from the SPEED, ACCEL and DECEL parameters from the base axis and the total multi-axes distance

$$L = \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2} \text{ where } x_i = ax_i - ay_i .$$

The individual speed v_i for axis i at any time of the movement is calculated as

$$v_i = \frac{x_i \cdot v_p}{L}.$$

Arguments: ***pos_i***
The position to move every axis *i* to in user units starting with the base axis.

See also: AXIS, MOVE, UNITS

Examples: **Example 1**
An X-Y plotter has a pen carousel whose position is fixed relative to the plotter origin. To change pen, an absolute move to the carousel position will find the target irrespective of the plot position when the command is executed.

```
MOVEABS(20,350)
```

Example 2
A pallet consists of a 6 by 8 grid in which gas canisters are inserted 85mm apart by a packaging machine. The canisters are picked up from a fixed point.

The first position in the pallet is defined as position (0,0) using the DEFPOS command. The part of the program to position the canisters in the pallet is as follows:

```
xloop:
FOR x = 0 TO 5
yloop:
   FOR y = 0 TO 7
      MOVEABS(-340,-516.5) 'Move to pick up point
      GOSUB pick           'Go to pick up subroutine
      PRINT "MOVE TO POSITION: ";x*6+y+1
      MOVEABS(x*85,y*85)
      GOSUB place          'Go to place down subroutine
   NEXT y
NEXT x
```

5-3-101 MOVECIRC

Type: Motion Control Command

Syntax: MOVECIRC(*end_1*,*end_2*,*centre_1*,*centre_2*,*direction*)

Alternative: MC(*end_1*,*end_2*,*centre_1*,*centre_2*,*direction*)

Description: MOVECIRC interpolates 2 orthogonal axes in a circular arc. The path of the movement is determined by the 5 arguments, which are incremental from the current position.

The arguments *end_1* and *centre_1* apply to the base axis and *end_2* and *centre_2* apply to the following axis. All arguments are given in user units of each axis. The speed of movement along the circular arc is set by the SPEED, ACCEL and DECEL parameters of the base axis.

MOVECIRC works on the default basis axis group (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: The MOVECIRC computes the radius and the total angle of rotation from the centre, and end-point. If the endpoint does not lie on the calculated path, the move simply ends at the computed end and not the specified end point. It is the responsibility of the programmer to ensure that the two points correspond to correct points on a circle.

For MOVECIRC to be correctly executed, the two axes moving in the circular arc must have the same number of encoder pulses per linear axis distance. If they do not, it is possible to adjust the encoder scales in many cases by adjusting with PP_STEP axis parameters for the axis.

Arguments:

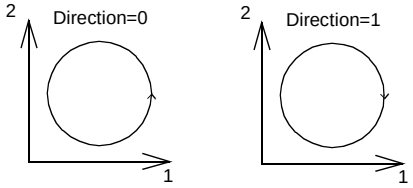
end_1
The end position for the base axis.

end_2
The end position for the next axis.

centre_1
The position around which the base axis is to move.

centre_2
The position around which the next axis is to move.

direction
A software switch that determines whether the arc is interpolated in a clockwise or counterclockwise direction. Value: 0 or 1



If the two axes involved in the movement form a right-hand axis, set *direction* to 0 to produce positive motion about the third (possibly imaginary) orthogonal axis.

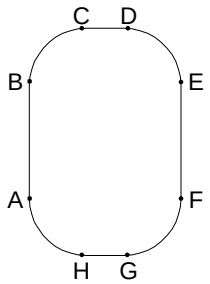
If the two axes involved in the movement form a left-hand axis. set *direction* to 0 to produce negative motion about the third (possibly imaginary) orthogonal axis.

Direction	Right-hand axis	Left-hand axis
1	Negative	Positive
0	Positive	Negative

See also: AXIS, MHELICAL, PP_STEP, UNITS

Example: The command sequence to plot the letter 0 might be as follows:

```
MOVE(0,6)                    'Move A -> B
MOVECIRC(3,3,3,0,1)          'Move B -> C
MOVE(2,0)                    'Move C -> D
MOVECIRC(3,-3,0,-3,1)        'Move D -> E
MOVE(O,-6)                   'Move E -> F
MOVECIRC(-3,-3,-3,0,1)       'Move F -> G
MOVE(-2,0)                   'Move G -> H
MOVECIRC(-3,3,0,3,1)         'Move H -> A
```



5-3-102 MOVELINK

Type: Motion Control Command

Syntax: `MOVELINK(`*distance*, *link_distance*, *link_acceleration*, *link_deceleration*, *link_axis*[, *link_option*[, *link_position*]]`)`

Alternative: `ML(`*distance*, *link_distance*, *link_acceleration*, *link_deceleration*, *link_axis* [, *link_option*[, *link_position*]]`)`

Description: MOVELINK creates a linear move on the base axis linked via a software gearbox to the measured position of a link axis. The link axis can move in either direction to drive the output motion.

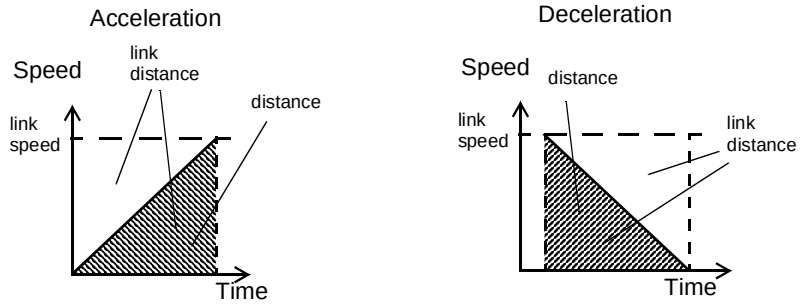
The parameters indicate what *distance* the base axis will move for a certain distance of the link axis (*link_distance*). The link axis distance is divided into three phases which apply to the movement of the base axis. These parts are the acceleration part, the constant speed part and the deceleration part. The link acceleration and deceleration distances are specified by the

link_acceleration and *link_deceleration* parameters. The constant speed link distance is derived from the total link distance and these two parameters.

The three phases can be divided into separate MOVELINK commands or can be added up together into one. Consider the following two rules when setting up the MOVELINK command.

Rule 1

In an acceleration and deceleration phase with matching speed, the *link_distance* must be twice the *distance*.



Rule 2

In a constant speed phase with matching speeds, the two axes travel the same distance so the *distance* to move must equal the *link_distance*.

MOVELINK works on the default basis axis group (set with BASE) unless AXIS is used to specify a temporary base axis. The axis set for *link_axis* drives the base axis.

Note If the sum of *link_acceleration* and *link_deceleration* is greater than *link_distance*, they are both reduced in proportion in order to equal the sum to *link_distance*.

Arguments:

distance

The incremental distance in user units to move the base axis, as a result of the measured *link_distance* movement on the link axis.

link_distance

The positive incremental distance in user units that is required to be measured on the link axis to result in the *distance* motion on the base axis.

link_acceleration

The positive incremental distance in user units on the link axis over which the base axis will accelerate.

link_deceleration

The positive incremental distance in user units on the link axis over which the base axis will decelerate.

link_axis

The axis to link to.

link_option

1 Link starts when registration event occurs on link axis.

2 Link starts at an absolute position on link axis (see *link_position*).

4 MOVELINK repeats automatically and bi-directionally. This option is canceled by setting bit 1 of REP_OPTION parameter (i.e. REP_OPTION = REP_OPTION OR 2).

5 Combination of options 1 and 4.

6 Combination of options 2 and 4.

link_position

The absolute position where MOVELINK will start when *link_option* is set to 2

See also: AXIS, REP_OPTION, UNITS

Example: A flying shear cuts a roll of paper every 160 m while moving at the speed of the paper. The shear is able to travel up to 1.2 m of which 1 m is used in this example. The paper distance is measured by an encoder, the unit conversion factor being set to give units of metres on both axes. Axis 3 is the link axis.

```
MOVELINK(0,150,0,0,3)      'wait distance
MOVELINK(0.4,0.8,0.8,0,3)  'accelerate
MOVELINK(0.6,1.0,0,0.8,3)  'match speed then decelerate
WAIT UNTIL NTYPE=0         'wait till last move started
OP(8,ON)                   'activate cutter
MOVELINK(-1,8.2,0.5,0.5,3) 'move back
```

In this program, the MC Unit waits for the roll to feed out 150m in the first line. After this distance, the shear accelerates to match the speed of the paper, coasts at the same speed, then decelerates to a stop within a 1m stroke. This movement is specified using two separate MOVELINK commands. The program then waits for the next move buffer to be clear NTYPE=0. This indicates that the acceleration phase is complete. The distances on the link axis (*link_distance*) in the MOVELINK commands are 150, 0.8, 1.0, and 8.2, which add up to 160m.

To ensure that the speeds and positions of the cutter and paper match during the cut task, the arguments of the MOVELINK command must be correct. Therefore it is easiest to first consider the acceleration, constant speed and deceleration phases separately. As mentioned before the acceleration and deceleration phases require the *link_distance* to be twice the *distance*. Both phases can therefore be specified as:

```
MOVELINK(0.4,0.8,0.8,0,1)      'This move is all accel
MOVELINK(0.4,0.8,0,0.8,1)      'This move is all decel
```

In a constant speed phase with matching speeds, the two axes travel the same distance so the distance to move must equal the link distance. The constant speed phase could, therefore, be specified as follows:

```
MOVELINK(0.2,0.2,0,0,1)        'This is all constant speed
```

The MOVELINK command allows the three sections to be added by summing the *distance, link_distance, link_acceleration* and *link_deceleration* for each phase, producing the following command.

```
MOVELINK(1,1.8,0.8,0.8,1)
```

In the program above, the acceleration phase is programmed separately. This is done to be able to perform some action at the end of the acceleration phase.

```
MOVELINK(0.4,0.8,0.8,0,1)
```

```
MOVELINK(0.6,1.0,0,0.8,1)
```

5-3-103 MOVEMODIFY

Type: Motion Control Command

Syntax: `MOVEMODIFY(`*position*`)`

Alternative: `MM(`*position*`)`

Description: MOVEMODIFY changes the absolute end position of the current single-axis linear move (MOVE or MOVEABS). If there is no current move or the current move is not a linear move, then MOVEMODIFY is treated as a MOVEABS command. The ENDMOVE parameter will contain the position of the end of the current move in user units.

MOVEMODIFY works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Arguments: ***position***
The absolute position to be set as the new end of move.

See also: AXIS, MOVE, MOVEABS, UNITS

5-3-104 MPOS

Type: Axis Parameter

Description: MPOS is the measured position of the axis in user units as derived from the encoder. This parameter can be set using the DEFPOS command. The OFFPOS axis parameter can also be used to shift the origin point. MPOS is reset to zero at start-up.

The range of the measured position is controlled with the REP_DIST and REP_OPTION axis parameters.

Note This parameter is read-only.

See also: AXIS, DEFPOS, DPOS, ENCODER, FE, OFFPOS, REP_DIST, REP_OPTION, UNITS

Example:
```
WAIT UNTIL MPOS >= 1250
SPEED = 2.5
```

5-3-105 MSPEED

Type: Axis Parameter

Description: MSPEED represents the change in the measured position in 10^{-3} user units/s in the last servo period. The servo period defaults to 1 ms. Therefore, the MSPEED parameter can be used to represent the speed measured in units/s.

MSPEED represents a snapshot of the speed and significant fluctuations, which can occur, particularly at low speeds. It can be worthwhile to average several readings if a stable value is required at low speeds.

See also: AXIS, VP_SPEED, UNITS

5-3-106 MTYPE

Type: Axis Parameter

Description: MTYPE contains the type of move currently being executed. The possible values are given below.

Move No.	Move Type
0	IDLE (no move)
1	MOVE
2	MOVEABS

3	MHELICAL
4	MOVECIRC
5	MOVEMODIFY
10	FORWARD
11	REVERSE
12	DATUM
13	CAM
14	JOG_FORWARD
15	JOG_REVERSE
20	CAMBOX
21	CONNECT
22	MOVELINK

MTYPE can be used to determine whether a move has finished or if a transition from one move type to another has taken place.

A non-idle move type does not necessarily mean that the axis is actually moving. It can be at zero speed part way along a move or interpolating with another axis without moving itself.

Note This parameter is read-only.

See also: AXIS, NTYPE

5-3-107 NEW

Type: Program Command

Syntax: `NEW` ["*program_name*"]

Description: NEW deletes all program lines of the program from memory. NEW without a program name can be used to delete the currently selected program (using SELECT). The program name can also be specified without quotes. NEW ALL will delete all programs.

NEW can also be used to delete the Table as follows:

NEW "TABLE"

The name "TABLE" must be in quotes.

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can select the command from the Program menu.

See also: COPY, DEL, RENAME, SELECT, TABLE

5-3-108 NIO

Type: System Parameter

Description: NIO contains the total number of inputs and outputs of the system.

5-3-109 NOT

Type: Logical Operator

Syntax: `NOT` *expression*

Description: NOT performs a NOT operation on all bits of the integer part of the expression.

The NOT operation is defined as follows:

Bit	Result
0	1
1	0

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT 7 AND NOT 1
6.0000
```

5-3-110 NTYPE

Type: Axis Parameter

Description: NTYPE contains the type of the move in the next move buffer. Once the current move has finished, the move present in the NTYPE buffer will be executed. The values are the same as those for the MTYPE axis parameter.
NTYPE is cleared by the CANCEL(1) command.

Note This parameter is read-only.

See also: AXIS, MTYPE

5-3-111 OFF

Type: Constant

Description: OFF returns the numerical value 0.

Note A constant is read-only.

Example:
```
OP (lever,OFF)
```
The above line sets the output named *lever* to OFF.

5-3-112 OFFPOS

Type: Axis Parameter

Description: OFFPOS contains an offset that will be applied to the demand position (DPOS) without affecting the move in any other way. The measured position will be changed accordingly in order to keep the following error. OFFPOS effectively adjusts the zero position of the axis. The value set in OFFPOS will be reset to zero by the system as the offset is loaded.

Precautions: The offset is applied on the next servo period. Other commands may be executed prior to the next servo period. Be sure that these commands do not assume the position shift has occurred. This can be done by using the WAIT UNTIL statement (see example).

See also: AXIS, DEFPOS, DPOS, MPOSUNITS

Example: The following lines define the current demand position as zero.
```
OFFPOS = -DPOS
WAIT UNTIL OFFPOS = 0      'Wait until applied
```
This example is equivalent to `DEFPOS(0)`.

5-3-113 ON

Type: Constant

Description: ON returns the numerical value 1.

Note A constant is read-only.

Example:
```
OP (lever,ON)
```
The above line sets the output named *lever* to ON.

5-3-114 ON

Type: Structural Command

Syntax: `ON` *expression* `GOSUB` *label*{, *label*}
`ON` *expression* `GOTO` *label*{, *label*}

Description: ON enables a conditional jump. The integer *expression* is used to select a *label* from the list. If the expression has value 1 the first *label* is used, for value 2 then the second *label* is used, and so on. Depending on the GOSUB or GOTO command the subroutine or normal jump is performed.

Precautions: If the expression is not valid, no jump is performed.

Arguments:

expression
Any valid BASIC expression.

label
Any valid label in the program.

See also: GOSUB, GOTO

Example:

```
REPEAT
   GET#5,char
UNTIL 1<=char and char<=3
ON char GOSUB mover, stopper, change
```

5-3-115 OP

Type: I/O Function/Command

Syntax:

OP(*output_number*, *value*)
OP(*binary_pattern*)
OP

Description: OP sets one or more outputs or returns the state of the first 24 outputs. OP has three different forms depending on the number of arguments.

- Command OP(*output_number*,*value*) sets a single output channel. The range of *output_number* is between 8 and 31 and *value* is the value to be output, either 0 or 1.
- Command OP(*binary_pattern*) sets the binary pattern to the 24 outputs according to the value set by *binary_pattern*.
- Function OP (without arguments) returns the status of the first 24 outputs. This allows multiple outputs to be set without corrupting others which are not to be changed.

Use the DISPLAY parameter to show the appropriate bank of 8 outputs or inputs on the uncommitted LEDs on the MC Unit. Refer to *Accessing I/O* for a description of the various types of output and inputs.

Precautions: The first 8 outputs (0 to 7) do not physically exist on the MC Unit. They can not be written to and will always return 0.

Arguments:

output_number
The number of the output to be set.

value
The value to be output, either OFF or ON. All non-zero values are considered as ON.

binary_pattern
The integer equivalent of the binary pattern is to be output.

See also: DISPLAY, IN

Examples:

Example 1

The following two lines are equivalent.

```
OP(12,1)
OP(12,ON)
```

Example 2

This following line sets the bit pattern 10010 on the first 5 physical outputs, outputs 13 to 31 would be cleared. The bit pattern is shifted 8 bits by multiplying by 256 to set the first available outputs as outputs 0 to 7 do not exist.

```
OP(18*256)
```

Example 3

This routine sets outputs 8 to 15 ON and all others OFF.

```
VR(0) = OP
VR(0) = VR(0) AND 65280
OP(VR(0))
```

The above programming can also be written as follows:

```
OP(OP AND 65280)
```

Example 4

This routine sets value *val* to outputs 8 to 11 without affecting the other outputs by using masking.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				val											

```
val = 8                          'The value to set
mask = OP AND NOT(15*256)        'Get current status and mask
OP(mask OR val*256)              'Set val to OP(8) to OP(11)
```

5-3-116 OPEN_WIN

Type: Axis Parameter

Alternative: OW

Description: OPEN_WIN defines the beginning of the window inside or outside which a registration event is expected. The value is in user units.

See also: CLOSE_WIN, REGIST, UNITS

5-3-117 OR

Type: Logical Operator

Syntax: *expression_1* OR *expression_2*

Description: OR performs an OR operation between corresponding bits of the integer parts of two valid BASIC expressions.

The OR operation between two bits is defined as follows:

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

Arguments: ***expression_1***

Any valid BASIC expression.

expression_2

Any valid BASIC expression.

Examples: **Example 1**

```
result = 10 OR (2.1*9)
```

The parentheses are evaluated first, but only the integer part of the result, 18, is used for the operation. Therefore, this expression is equivalent to the following:

```
result = 10 OR 18
```

The OR is a bit operator and so the binary action taking place is:

```
          01010
OR        10010
          -----
          11010
```

Therefore, *result* will contain the value 26.

Example 2

```
IF KEY OR VR(0) = 2 THEN GOTO label
```

5-3-118 OUTLIMIT

Type: Axis Parameter

Description: OUTLIMIT contains an output limit that restricts the voltage output from the MC Unit for both servo loop (SERVO=ON) and open loop (SERVO=OFF). The default is the maximum voltage output generated by a 12-bit DAC, which is value 2047. Therefore, the output values are normally limited to -2048 to 2047 (–10 V to 10 V). Although the OUTLIMIT can be given values higher than 2047, the output voltage can never be outside the -10 V to 10 V range.

See also: AXIS, SERVO, DAC_OUT

Example:

```
OUTLIMIT AXIS(0) = 1023
```

The above line will limit the voltage output to a 5 V limit output (-5 V to 5 V).

5-3-119 OV_GAIN

Type: Axis Parameter

Description: OV_GAIN contains the output velocity gain. The output velocity output contribution is calculated by multiplying the change in measured position with OV_GAIN. The default value is 0.

Adding output velocity gain to a system is mechanically equivalent to adding damping. It is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used, but at the expense of higher following errors. High values may cause oscillation and produce high following errors.

See section *1-4-2 Servo System Principles* for more details.

Precautions: In order to avoid any instability the servo gains should be changed only when the SERVO is OFF.

See also: AXIS, D_GAIN, I_GAIN, P_GAIN, VFF_GAIN

5-3-120 P_GAIN

Type: Axis Parameter

Description: P_GAIN contains the proportional gain. The proportional output contribution is calculated by multiplying the following error with P_GAIN. The default value is 1.0.

The proportional gain sets the 'stiffness' of the servo response. Values that are too high will cause oscillation. Values that are too low will cause large following errors.

See section *1-4-2 Servo System Principles* for more details.

Precautions: In order to avoid any instability the servo gains should be changed only when the SERVO is OFF.

See also: AXIS, D_GAIN, I_GAIN, OV_GAIN, VFF_GAIN

5-3-121 PI

Type: Constant

Description: PI returns the numerical value 3.1416.

Note A constant is read-only.

Example:

```
circum = 100
PRINT "Radius = ";circum/(2*PI)
```

5-3-122 PLC_READ

Type: Command

Syntax: PLC_READ(*PC_area*,*offset*,*length*,*vr_number*)

Description: The PLC_READ command is used to read data from the PC using one of the PC Data Exchange methods and write it to the VR variables. The PLC_READ enables the user to access PC data either by reading directly from the PC memory or by reading the two PC output words in IR/CIO area. Refer to *3-2 Overview of Data Exchanges* for more details on PC Data Exchange.

Direct data transfer

PLC_READ requests a data transfer from the CPU Unit to the MC Unit at the end of the next CPU Unit I/O refresh. The transfer of data takes place transparently to the PC program. The PC area to read from is specified by the *PC_area* argument. The input is given by PLC_xx, where xx is the specified memory area. The maximum amount of data to be transferred is 127 words.

Data words in IR/CIO area

The two word output data within the PC area is updated every I/O refresh to the allocated words in the MC Unit. These allocated words can be copied to VR variables by specifying the *PC_area* argument as PLC_REFRESH.

The other arguments specify the amount of data to be read and the source and destination addresses. The program execution will pause until the data is transferred.

Arguments:

PC_area

The memory area in the CPU Unit from which the data is to be transferred.

PC_area	Data area	Address range
PLC_DM (0)	DM area	0 to 1999 - *C200H* 0 to 6143 - *C200HS/HE/HG/HX, CS1*
PLC_IR (1)	IR(SR) area	0 to 255 - *C200H* 0 to 299 - *C200HS* 0 to 511 - *C200HE/HG/HX, CS1*
PLC_LR (2)	LR area	0 to 63 - *C200H/HS/HE/HG/HX, CS1*
PLC_HR (3)	HR area	0 to 99 - *C200H/HS/HE/HG/HX, CS1*
PLC_AR (4)	AR area	0 to 27 - *C200H/HS/HE/HG/HX, CS1*
PLC_TC (5)	TC area	0 to 511 - *C200H/HS/HE/HG/HX, CS1*
PLC_EM (6)	EM area	0 to 6143 - *C200HE/HG/HX, CS1*
PLC_REFRESH (7)	-	-

offset

The address offset into the specified memory area in the CPU Unit.

length

The number of words of data to be transferred starting from, and including, the specified address offset in the CPU Unit.

vr_number

The first VR variable into which word data is to be stored. Consecutive VR variables will be used for any subsequent words that are transferred.

Care must be taken when specifying *vr_number* to ensure that the specified VR variables are not outside the VR variable range. A parameter out-of-range error will occur if this restriction is not met.

Examples: **Example 1**

To read the contents of DM 500 to DM 599 from the CPU Unit into the VR(100) to VR(199) on the MC Unit, the following command must be executed.

```
PLC_READ(PLC_DM,500,100,100)
```

Example 2

The following command is used to read the word data from the CPU Unit's IR area from the allocated words in the MC Unit to VR(10) and VR(11).

```
PLC_READ(PLC_REFRESH,0,2,10)
```

5-3-123 PLC_TYPE

Type: System Parameter

Description: PLC_TYPE gives the PC CPU Unit model that the MC402 is connected to on the Backplane. The possible values are as follows:

0	Unknown PC
1	C200H
2	C200HS
3	C200HE (CPU11)
4	C200HE (not CPU11)
5	C200HG
6	C200HX (up to CPU85-Z)
7	C200HX (CPU85-Z and up)
	CS1

Note This parameter is read-only.

Example: Assuming the connected CPU Unit model is a C200HG, the following line will return give this result.

```
>> PRINT PLC_TYPE[0]
5
```

5-3-124 PLC_WRITE

Type: Command

Syntax: `PLC_WRITE(`*PC_area*`,`*offset*`,`*length*`,`*vr_number*`)`

Description: The PLC_WRITE command is used to and read data from the VR variables and write the data to the PC using one of the PC Data Exchange methods. The PLC_WRITE enables the user to access PC data either by writing directly to the PC memory or by writing to the two PC input words in IR/CIO area. Refer to *3-2 Overview of Data Exchanges* for more details on PC Data Exchange.

Direct data transfer

PLC_WRITE requests a data transfer from the MC Unit to the CPU Unit at the end of the next CPU Unit I/O refresh. The transfer of data takes place transparently to the PC program. The PC area to write to is specified by the *PC_area* argument. The input is given by PLC_xx, where xx is the specified memory area. The maximum amount of data to be transferred is 127 words.

Data words in IR/CIO area

The two word input data within allocated words in the MC Unit are updated every I/O refresh to the IR/CIO area of the CPU Unit. These allocated words can be written from the VR variables by specifying the *PC_area* argument as PLC_REFRESH.

The other arguments specify the amount of data to be written and the source and destination addresses. The program execution will pause until the data is transferred.

Arguments:

PC_Area

The memory area in the CPU Unit from which the data is to be transferred. The valid areas are as follows:

PC_area	Data area	Address range
PLC_DM (0)	DM area	0 to 1999 - *C200H* 0 to 6143 - *C200HS/HE/HG/HX, CS1*
PLC_IR (1)	IR(SR) area	0 to 255 - *C200H* 0 to 299 - *C200HS* 0 to 511 - *C200HE/HG/HX, CS1*
PLC_LR (2)	LR area	0 to 63 - *C200H/HS/HE/HG/HX, CS1*
PLC_HR (3)	HR area	0 to 99 - *C200H/HS/HE/HG/HX, CS1*
PLC_AR (4)	AR area	0 to 27 - *C200H/HS/HE/HG/HX, CS1*
PLC_TC (5)	TC area	0 to 511 - *C200H/HS/HE/HG/HX, CS1*
PLC_EM (6)	EM area	0 to 6143 - *C200HE/HG/HX, CS1*
PLC_REFRESH (7)	-	-

offset

The address offset into the specified memory area in the CPU Unit.

length

The number of words of data to be transferred starting from, and including, the specified address offset in the CPU Unit.

vr_number

The first VR variable from which word data is to be read. Consecutive VR variables will be used for any subsequent words that are transferred.

Care must be taken when specifying *vr_number* to ensure that the specified VR variables are not outside the VR variable range. A parameter out-of-range error will occur if this restriction is not met.

Example: To write the contents of variables VR(100) to VR(199) on the MC Unit to DM 500 to DM 599 in the CPU Unit, the following command must be executed.

```
PLC_WRITE(PLC_DM,500,100,100)
```

5-3-125 PMOVE

Type: Task Parameter

Description: PMOVE will contain TRUE if the task buffers are occupied, and FALSE if they are empty.

When the task executes a movement command, the task loads the movement information into the task move buffers. The buffers can hold one movement instruction for any group of axes. PMOVE will be set to TRUE when loading of the buffers has been completed. When the next servo interrupt occurs, the motion generator will load the movement into the next move (NTYPE) buffers of the required axes if they are available. When this second transfer has been completed, PMOVE will be cleared to zero until another move is executed in the task.

Each task has its own PMOVE parameter. Use the PROC modifier to access the parameter for a certain task. Without PROC the current task will be assumed.

Note This parameter is read-only.

See also: NTYPE, PROC

5-3-126 POWER_UP

Type: System Parameter

Description: POWER_UP contains the location of programs to be used at start-up, either in RAM or in flash EPROM. The POWER_UP parameter is stored in flash EPROM.

The following values are valid:

0 Use programs in RAM

1 Copy programs from flash EPROM into RAM before using program in RAM

Precautions:

- Programs are individually set to be run at start-up with the RUNTYPE command.
- POWER_UP cannot be included in BASIC programs.

See also: EPROM, RUNTYPE

5-3-127 PP_STEP

Type: Axis Parameter

Description: PP_STEP contains an integer value that scales the incoming raw encoder count. The incoming raw encoder count will be multiplied by PP_STEP before being applied. Scaling can be used to match encoders to high-resolution motors for position verification or for moving along circular arcs on machines where the number of encoder edges/distance is not the same on the axes.

The valid range is [-1023, -1] and [1, 1023]. Default is value 1.

See also: AXIS, MHELICAL, MOVECIRC, UNITS

Examples:

Example 1

A motor has 20,000 steps/rev. The MC Unit will thus internally process 40,000 counts/rev. A 2,500-pulse encoder is to be connected. This will generate 10,000 edge counts/rev. A multiplication factor of 4 is therefore required to convert the 10,000 counts/rev to match the 40,000 counts/rev of the motor. The following line would be used for axis 3.

```
PP_STEP AXIS(3) = 4
```

Example 2

An X-Y machine has encoders which input 50 edges/mm in the X axis (axis 0) and 75 edges/mm in the Y axis (axis 1). Circular arc interpolation is required between the axes. This requires that the interpolating axes have the same number of encoder counts/distance. It is not possible to multiply the X axis counts by 1.5, because PP_STEP must be an integer. Both X and Y axes must, therefore, be set to give 150 edges/mm. The settings would be as follows:

```
PP_STEP AXIS(0) = 3
PP_STEP AXIS(1) = 2
UNITS AXIS(0) = 150
UNITS AXIS(1) = 150
```

5-3-128 PRINT

Type: I/O Command

Syntax: `PRINT`[*#n*,] *expression*{, *expression*}

?[*#n*,] *expression*{, *expression*}

Description: PRINT outputs a series of characters to the serial ports. PRINT can output parameters, fixed ASCII strings, and single ASCII characters. By using PRINT#n, any port can be selected to output the information to. The default port is the RS-232C programming port A.

Multiple items to be printed can be put on the same line separated by a comma "," or a semi-colon ";". A comma separator in the print command places a tab between the printed items. The semi-colon separator prints the next item without any spaces between printed items.

The width of the field in which a number is printed can be set with the use of [*w,x*] after the number to be printed. The width of the column is given by *w* and the number of decimal places is given by *x*. Using only one parameter [*x*] takes the default width and specifies the number of decimal places to be printed. The numbers are right aligned in the field with any unused leading characters being filled with spaces. If the number is too long, then the field will be filled with asterisks to signify that there was not sufficient space to display the number. The maximum field width allowable is 127 characters.

CHR(x)

The CHR(x) command is used to send individual ASCII characters using their ASCII codes. The semi-colon on the end of the print line suppresses the carriage return normally sent at the end of a print line. ASCII(13) generates CR without a linefeed so the line above would be printed on top of itself if it were the only print statement in a program. PRINT CHR(x); is equivalent to PUT(x) in some forms of BASIC.

Arguments:

n

The specified output device. If omitted, the RS-232C programming port will be used.

0	RS-232C programming port A (default)
1	RS-232C serial port B
5	Motion Perfect port A user channel 5
6	Motion Perfect port A user channel 6
7	Motion Perfect port A user channel 7

expression

The expression to be printed.

Examples:

Example 1

```
PRINT "CAPITALS and lower case CAN BE PRINTED"
```

Example 2

Consider VR(1) = 6 and variab = 1.5, the print output will be as follows:

```
PRINT 123.45,VR(1)-variab
123.4500  4.5000
```

Example 3

In this example, the semi-colon separator is used. This does not tab into the next column, allowing the programmer more freedom in where the print items are placed.

```
length:
   PRINT "DISTANCE = ";mpos
   DISTANCE = 123.0000
```

Example 4

```
PRINT VR(1)[ 4,1 ];variab[ 6,2 ]
6.0 1.50
```

Example 5

```
params:
   PRINT "DISTANCE = ";mpos[ 0 ];" SPEED = ";v[ 2 ];
   DISTANCE = 123 SPEED = 12.34
```

Example 6

```
PRINT "ITEM ";total" OF ";limit;CHR(13);
```

5-3-129 PROC

Type: Task Command

Syntax: PROC(*task_number*)

Description: The PROC modifier allows a process parameter from a particular process to be read or written. If omitted, the current task will be assumed.

Argument: ***task_number***
The number of the task to access.

Example:
```
WAIT UNTIL PMOVE PROC(3)=0
```

5-3-130 PROCESS

Type: Program Function

Syntax: PROCESS

Description: PROCESS returns the running status and task number for each current task.

See also: HALT, RUN, STOP

5-3-131 PROCNUMBER

Type: Task Parameter

Description: The PROCNUMBER parameter contains the number of the task in which the currently selected program is running. PROCNUMBER is often required when multiple copies of a program are running on different tasks.

Note This parameter is read-only.

Example: MOVE(length) AXIS(PROCNUMBER)

5-3-132 PSWITCH

Type: I/O Command

Syntax: PSWITCH(*switch*, *enable*[, *axis*, *output_number*, *output_state*, *set_position*, *reset_position*])

Description: PSWITCH turns ON an output when a predefined position is reached, and turns OFF the output when a second position is reached. The positions are specified as the measured absolute positions.

There are 16 position switches each of which can be assigned to any axis. Each switch is assigned its own ON and OFF positions and output number.

The command can be used with 2 or all 7 arguments. With only 2 arguments a given switch can be disabled.

PSWITCHs are calculated on each servo cycle and the output result applied to the hardware. The response time is therefore 1 servo cycle (1 ms) approximately.

Precautions: An output may remain ON if it was ON when the PSWITCH was turned OFF. The OP command can be used to turn OFF an output as follows:

```
PSWITCH(2,OFF)
OP(14,OFF)     'Turn OFF pswitch controlling OP 14
```

Arguments: ***switch***
The switch number. Range: [0,15].

enable
The switch enable. Range: [ON, OFF].

axis
The number of the axis providing the position input.

output_number
The physical output to set. Range: [8,15] and [20,31].

output_state
The state to output. Range: [ON, OFF].

set_position
The absolute position in user units at which output is set.

reset_position
The absolute position in user units at which output is reset.

See also: OP, UNITS

Example: A rotating shaft has a cam operated switch which has to be changed for different size work pieces. There is also a proximity switch on the shaft to indicate the TDC of the machine. With a mechanical cam, the change from job to job is time consuming. This can be eased by using PSWITCH as a software cam switch. The proximity switch is wired to input 7 and the output is output 11. The shaft is controlled by axis 0 of a 3-axis system. The motor has a 900ppr encoder. The output must be on from 80 units.

PSWITCH uses the unit conversion factor to allow the positions to be set in convenient units. First the unit conversion factor must be calculated and set. Each pulse on an encoder gives four edges for the MC Unit to count. There are thus 3,600 edges/rev or 10 edges/degree. If we set the unit conversion factor to 10, we can work in degrees.

Next we have to determine a value for all the PSWITCH arguments.

sw	The switch number can be any switch that is not in use. In this example, we will use number 0.
en	The switch must be enabled to work; set the enable to 1.
axis	The shaft is controlled by axis 0.
opno	The output being controlled is output 11.
opst	The output must be on so set to 1.
setpos	The output is to produced at 80 units.
rspos	The output is to be on for a period of 120 units.

This can all be put together in the following lines of BASIC code:

```
switch:
   UNITS AXIS(0) = 10          'Set unit conversion factor
   REPDIST = 360
   REP_OPTION = ON
   PSWITCH(0,ON,0,11,ON,80,200)
```

This program uses the repeat distance set to 360 degrees and the repeat option ON so that the axis position will be maintained between 0 and 360 degrees.

5-3-133 RAPIDSTOP

Type: Motion Control Command

Syntax: RAPIDSTOP

Alternative: RS

Description: RAPIDSTOP cancels the current move on all axes from the current move buffer (MTYPE). Moves for speed profiled move commands (MOVE, MOVEABS, MOVEMODIFY, FORWARD, REVERSE, MOVECIRC, and MHELICAL) will decelerate to a stop. Moves for other commands will be immediately stopped.

Precautions:

- RAPIDSTOP cancels only the presently executing moves. If further moves are buffered in the next move buffers (NTYPE) or the task buffers they will then be loaded.

- During the deceleration of the current moves additional RAPIDSTOPs will be ignored.

See also: CANCEL, MTYPE, NTYPE

5-3-134 READ_BIT

Type: System Command

Syntax: READ_BIT(*bit_number*, *vr_number*)

Description: The READ_BIT command returns the value of the specified bit in the specified VR variable, either 0 or 1.

Arguments: ***bit_number***
The number of the bit to be read. Range: [0,23].
vr_number
The number of the VR variable for which the bit is read. Range: [0,250].

See also: CLEAR_BIT, SET_BIT, VR

5-3-135 REG_POS

Type: Axis Parameter

Alternative: RPOS

Description: REG_POS stores the position in user units at which a registration event occurred.

Note This parameter is read-only.

See also: AXIS, MARK, REGIST, UNITS

5-3-136 REGIST

Type: Axis Command

Syntax: REGIST(*mode*)

Description: REGIST captures an axis position when a registration input or the Z marker on the encoder is detected.

The capture is carried out by hardware, so software delays do not affect the accuracy of the position captured. If the registration input or Z marker is detected as specified within the specified window, the MARK axis parameter will be set to TRUE and the position will stored in the REG_POS axis parameter.

The inputs R0 to R3 correspond to registration inputs for servo axes 0 to 3. These registration inputs are fixed, but other functions like datum switch input and limit switch inputs can also be allocated to inputs R0 to R3. Refer to *1-5 Specifications* for the time delay on the inputs.

REGIST works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Inclusive windowing

Add 256 to the *mode* argument value to apply inclusive windowing. When inclusive windowing is applied, signals will be ignored if the axis measured position is not greater than the OPEN_WIN parameter and less than the CLOSE_WIN parameter.

Exclusive windowing

Add 768 to the *mode* argument value to apply exclusive windowing. When exclusive windowing is applied, signals will be ignored if the axis measured position is not less than the OPEN_WIN parameter or greater than the CLOSE_WIN parameter.

Precautions: REGIST must be executed once for each position capture.

Arguments: ***mode***

Specifies the type of capture to make.

1 Capture absolute position on rising edge of Z marker
2 Capture absolute position on falling edge of Z marker
3 Capture absolute position on rising edge of registration input
4 Capture absolute position on falling edge of registration input

See also: AXIS, CLOSE_WIN, MARK, OPEN_WIN, REG_POS

Examples: **Example 1**

```
catch:
   REGIST(3)
   WAIT UNTIL MARK
   PRINT "Registration input at:";REG_POS
```

Example 2

A paper cutting machine uses a CAM profile to quickly draw paper through servo-driven rollers and then stop it while it is cut. The paper is printed with a registration mark. This mark is detected and the length of the next sheet is adjusted by scaling the CAM profile with the third argument (*table_multiplier)* of the CAM command:

```
   ‘Set window open and close
   length = 200
   OPEN_WIN = 10
   CLOSE_WIN = length-10

   GOSUB Initial

loop:
   TICKS = 0              ’Set servo cycle counter to 0
   IF MARK THEN
      offset = REG_POS
      ’Next line makes offset -ve if at end of sheet
      IF ABS(offset-length) < offset THEN
         offset=offset - length
      ENDIF
      PRINT "Mark seen at:"offset[5.1]
   ELSE
      offset = 0
      PRINT "Mark not seen"
   ENDIF

   ‘Reset registration prior to each move
   DEFPOS(0)
   REGIST(3+768)

   ’Allow mark at first 10 mm or last 10 mm of sheet
   CAM(0,50,(length+offset*0.5)*cf,1000)
   WAIT UNTIL TICKS > 500
   GOTO loop
```

The variable *cf* is a constant which would be calculated depending on the machine draw length per encoder edge.

5-3-137 REMAIN

Type: Axis Parameter

Description: The REMAIN axis parameter is the distance remaining to the end of the current move. It can be tested to see how much of the move has been completed. REMAIN is defined in user units.

Note This parameter is read-only.

See also: AXIS, UNITS

Example: To change the speed to a slower value 5mm from the end of a move.

```
start:
   SPEED = 10
   MOVE(45)
   WAIT UNTIL REMAIN < 5
   SPEED = 1
   WAIT IDLE
```

5-3-138 RENAME

Type: Program Command

Syntax: RENAME "*old_program_name*" "*new_program_name*"

Description: RENAME changes the name of a program in the MC Unit directory. The program names can also be specified without quotes.

Precautions: This command is implemented for an offline (VT100) terminal. Within Motion Perfect users can select the command from the Program menu.

Arguments: ***old_program_name***
Current name of the program.
new_program_name
New name of the program.

See also: COPY, DEL, NEW

Example:

```
RENAME "car" "voiture"
```

5-3-139 REP_DIST

Type: Axis Parameter

Description: REP_DIST contains the repeat distance, which is the allowable range of movement for an axis before the demand position (DPOS) and measured position (MPOS) are corrected. REP_DIST is defined in user units. The exact range is controlled by REP_OPTION. The REP_DIST can have any non-zero positive value.

When the measured position has reached its limit, the MC unit will adjust the absolute positions without affecting the move in progress or the servo algorithm. Not that the demand position can be outside the range because the measured position is used to trigger the adjustment.

For every occurrence (DEFPOS, OFFPOS, MOVEABS, MOVEMODIFY) which defines a position outside the range, the end position will be redefined within the range.

See also: AXIS, DPOS, MPOS, REP_OPTION, UNITS

5-3-140 REP_OPTION

Type: Axis Parameter

Description: REP_OPTION controls the application of the REP_DIST axis parameter and the repeat option of the CAMBOX and MOVELINK motion control commands.

Bit	Description
0	The repeated distance range is controlled by bit 0 of the REP_OPTION parameter. • If REP_OPTION bit 0 is OFF, the range of the demanded and measured positions will be between -REP_DIST and REP_DIST. • If REP_OPTION bit 0 is ON, the range of the demanded and measured positions will be between 0 and REP_DIST.
1	The automatic repeat option of the CAMBOX and MOVELINK commands are controlled by bit 1 of the REP_OPTION parameter. The bit is set ON to request the system software to end the automatic repeat option. When the system software has set the option OFF it automatically clears bit 1 of REP_OPTION.

See also: AXIS, CAMBOX, MOVELINK, REP_DIST

5-3-141 REPEAT UNTIL

Type: Structural Command

Syntax:
```
REPEAT
   <commands>
UNTIL condition
```

Description: The REPEAT ... UNTIL loop allows the program segment between the REPEAT and the UNTIL statement to be repeated a number of times until the *condition* becomes TRUE.

Precautions: REPEAT ... UNTIL construct can be nested indefinitely.

Arguments:
commands
Any valid set of BASIC commands
condition
Any valid BASIC logical expression

See also: FOR, WHILE

Example: A conveyor is to index 100mm at a speed of 1000mm/s, wait for 0.5s and then repeat the cycle until an external counter signals to stop by turning ON input 4.

```
cycle:
   SPEED = 1000
   REPEAT
      MOVE(100)
      WAIT IDLE
      WA(500)
   UNTIL IN(4) = ON
```

5-3-142 RESET

Type: System Command

Syntax:
```
RESET
```

Description: The RESET command sets the value of all local variables of the current BASIC task to zero.

See also: CLEAR

5-3-143 REV_IN

Type: Axis Parameter

Description: REV_IN contains the input number to be used as a reverse limit input. The number can be from 0 to 15 and from 20 to 31. If REV_IN is set to –1, then no input is used as a reverse limit.

If an input number is set and the limit is reached, any reverse motion on that axis will be stopped. Bit 5 of the AXISSTATUS axis parameter will also be set.

Note This input is active low.

See also: AXIS, AXISSTATUS, FWD_IN

5-3-144 REV_JOG

Type: Axis Parameter

Description: REV_JOG contains the input number to be used as a jog reverse input. The input can be from 0 to 15 and from 20 to 31. If REV_JOG is set to –1 (default), then no input is used as a reverse jog input.

Note This input is active low.

See also: AXIS, FAST_JOG, FWD_JOG, JOGSPEED

5-3-145 REVERSE

Type: Motion Control Command

Syntax: REVERSE

Alternative: RE

Description: REVERSE moves an axis continuously in reverse at the speed set in the SPEED parameter.

REVERSE works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: The reverse motion can be stopped by executing the CANCEL or RAPIDSTOP command, or by reaching the reverse limit, inhibit, or origin return limit.

See also: AXIS, CANCEL, FORWARD, RAPIDSTOP

Example:

```
back:
   REVERSE
   WAIT UNTIL IN(0) = ON   'Wait for stop signal
   CANCEL
```

5-3-146 RS_LIMIT

Type: Axis Parameter

Alternative: RSLIMIT

Description: RS_LIMIT contains the absolute position of the reverse software limit in user units.

A software limit for reverse travel can be set from the program to control the working envelope of the machine. When the limit is reached, the MC Unit will decelerate to zero, and then cancel the move. Bit 10 of the AXISSTATUS axis parameter will be turned ON when the axis position is smaller than / below RS_LIMIT.

See also: AXIS, FS_LIMIT, UNITS

5-3-147 RUN

Type: Program Command

Syntax: RUN ["*program_name*"[, *task_number*]]

Description: RUN executes the program in the MC Unit as specified with *program_name*. RUN with the program name specification will run the current selected program. The program name can also be specified without quotes.

The task number specifies the task number on which the program will be run. If the task number is omitted, the program will run on the highest available task. RUN can be included in a program to run another program.

Precautions: Execution continues until one of the following occurs:

- There are no more lines to execute.
- HALT is typed at the command line to stop all programs.
- STOP is typed at the command line to stop a single program.
- A run-time error is encountered.

Arguments: ***program_name***
Any valid program name.
task_number
Any valid task number. Range: [1,5].

See also: HALT, STOP

Examples: **Example 1**
The following example executes the currently selected program.

```
>> SELECT "PROGRAM"
PROGRAM selected
>> RUN
```

Example 2
The following example executes the program named “sausage”.

```
RUN "sausage"
```

Example 3
The following example executes the program named “sausage” on task 3.

```
RUN "sausage",3
```

5-3-148 RUN_ERROR

Type: Task Parameter

Description: RUN_ERROR contains the number of the last BASIC run-time error that occurred on the specified task.

Each task has its own RUN_ERROR parameter. Use the PROC modifier to access the parameter for a certain task. Without PROC the current task will be assumed.

Note This parameter is read-only.

See also: BASICERROR, ERROR_LINE, PROC

Example:

```
>> PRINT RUN_ERROR PROC(5)
9.0000
```

5-3-149 RUNTYPE

Type: Program Command

Syntax: RUNTYPE “*program_name*”, *auto_run*[, *task_number*]

Description: RUNTYPE determines whether the program, specified by *program_name*, is run automatically at start-up or not and which task it is to run on. The task number is optional, if omitted the program will run at the highest available task.

The current RUNTYPE status of each programs is displayed when a DIR command is executed. If one program has compilation errors no programs will be started at power up.

Precautions: To set the RUNTYPE using Motion Perfect, select "Set Power-up mode" from the Program Menu. RUNTYPE information is stored into the flash EPROM only when the EPROM command is executed after.

Arguments:

program_name
The name of the program whose RUNTYPE is being set.

autorun

0 Running manually on command.

1 Automatically execute on power up. All non-zero values are considered as 1.

task_number
The number of the step on which to execute the program. Range: [1,5].

See also: AUTORUN, EPROM, EX

Example:

```
>> RUNTYPE progname,1,5
```

The above line sets the program "progname" to run automatically at start-up on task 5.

```
>> RUNTYPE progname,0
```

The above line sets the program "progname" to manual running.

5-3-150 SCOPE

Type: Motion Perfect Command

Syntax: SCOPE(*ON/OFF_control*, *period*, *table_start*, *table_stop*, *P0*[, *P1*[, *P2*[, *P3*]]])

Description: SCOPE programs the system to automatically store up to 4 parameters every sample period. The storing of data will start as soon as the TRIGGER command has been executed.

The sample period can be any multiple of the servo period. The parameters are stored in the Table array and can then be read back to a computer and displayed on the Motion Perfect Oscilloscope or written to a file for further analysis using the "Create Table file" option on the File Menu.

The current Table position for the first parameter which is written by SCOPE can be read from the SCOPE_POS parameter.

Note Motion Perfect uses SCOPE when running the Oscilloscope function.

Precautions: Applications like the CAM command, CAMBOX command and the SCOPE command all use the same Table as the data area. Do not use the same data area for different applications.

Arguments:

ON/OFF_control
Set ON or OFF to control SCOPE execution. If turned ON the SCOPE is ready to run as soon as the TRIGGER command is executed.

period
The number of servo periods between data samples.

table_start
The address of the first element in the Table array to start storing data.

table_stop
The address of the last element in the Table array to be used.

P0
First parameter to store.

P1
Optional second parameter to store.

P2
Optional third parameter to store.
P3
Optional fourth parameter to store.

See also: SCOPE_POS, TABLE, TRIGGER

Examples: **Example 1**

```
SCOPE(ON,10,0,1000,MPOS AXIS(4),DPOS AXIS(4))
```

This example programs the SCOPE function to store the MPOS parameter for axis 4 and the DPOS parameter for axis 4 every 10 ms. The MPOS parameter will be stored in table locations 0 to 499; the DPOS parameters, in table locations 500 to 999. The SCOPE function will wrap and start storing at the beginning again unless stopped. Sampling will not start until the TRIGGER command is executed.

Example 2

```
SCOPE(OFF)
```

This above line turns the scope function off.

5-3-151 SCOPE_POS

Type: Motion Perfect Parameter

Description: SCOPE_POS contains the current Table position at which the SCOPE command is currently storing its first parameter.

Note This parameter is read-only.

See also: SCOPE

5-3-152 SELECT

Type: Program Command

Syntax: SELECT *"program_name"*

Description: SELECT specifies the current program for editing, running, listing, etc., SELECT makes a new program if the name entered does not exist. The program name can also be specified without quotes.

When a program is selected, the commands COMPILE, DEL, EDIT, LIST, NEW, RUN, STEPLINE, STOP and TROFF will apply to the currently selected program unless a program is specified in the command line. When another program is selected, the previously selected program will be compiled. The selected program cannot be changed when a program is running.

Precautions: This command is implemented for an offline (VT100) terminal. Motion Perfect automatically selects programs when you click on their entry in the list in the control panel.

See also: COMPILE, DEL, EDIT, LIST, NEW, RUN, STEPLINE, STOP, TROFF

Example:

```
>> SELECT "PROGRAM"
PROGRAM selected
>> RUN
```

5-3-153 SERVO

Type: Axis Parameter

Description: SERVO determines whether the base axis runs under servo control or open loop. When SERVO is ON, the axis hardware will output a voltage depending on the gain settings and the following error.

When SERVO is OFF, the axis hardware will output a voltage dependent on the DAC axis parameter. All non-zero values for SERVO is also considered as ON.

See also: AXIS, DAC, FE_LIMIT, WDOG

Example:
```
SERVO AXIS(0) = ON        'Axis 0 is under servo control
SERVO AXIS(1) = OFF       'Axis 1 is run open loop
```

5-3-154 SET_BIT

Type: System Command

Syntax: SET_BIT(*bit_number*, *vr_number*)

Description: The SET_BIT command sets the specified bit in the specified VR variable to one. Other bits in the variable will keep their values.

Arguments:

bit_number

The number of the bit to be set. Range: [0,23].

vr_number

The number of the VR variable for which the bit is set. Range: [0,250].

See also: CLEAR_BIT, READ_BIT, VR

5-3-155 SETCOM

Type: I/O Command

Syntax: SETCOM(*baud_rate*, *data_bits*, *stop_bits*, *parity*[, *port_number* [, *XON/XOFF_switch*]])

Description: SETCOM sets the serial communications. By default, the RS-232C port settings are 9,600 baud, 7 data bits, 2 stop bits and even parity. These default settings are recovered at start-up.

Arguments:

baud_rate

1200, 2400,4800, 9600,19200, 38400

data_bits

7, 8

stop_bits

1, 2

parity

0	None
1	Odd
2	Even

port_number

0	RS-232C programming port A (default)
1	RS-232C serial port B

XON/ XOFF_switch

0	OFF
1	ON

This switch is available only on serial port B.

5-3-156 SGN

Type: Mathematical Function

Syntax: SGN(*expression*)

Description: SGN returns the sign of a number. It returns value 1 for positive values (including zero) and value -1 for negative values.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT SGN(-1.2)
-1.0000
```

5-3-157 SIN

Type: Mathematical Function

Syntax: `SIN(`*expression*`)`

Description: SIN returns the sine of the *expression*. Input values are in radians and may have any value. The result value will be in the range from -1 to 1.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT SIN(PI/2)
1.0000
```

5-3-158 SPEED

Type: Axis Parameter

Description: The SPEED parameter contains the demand speed in units/s. It can have any positive value (including zero). The demand speed is the maximum speed for the speed profiled motion commands.

See also: ACCEL, AXIS, DATUM, DECEL, FORWARD, MHELICAL, MOVE, MOVEABS, MOVECIRC, MOVEMODIFY, REVERSE, UNITS

Example:
```
SPEED = 1000
PRINT "Set speed = ";SPEED
```

5-3-159 SQR

Type: Mathematical Function

Syntax: `SQR(`*expression*`)`

Description: SQR returns the square root of the *expression*. The *expression* must have positive (including zero) value.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> PRINT SQR(4)
2.0000
```

5-3-160 SRAMP

Type: Axis Parameter

Description: SRAMP contains the S-curve factor. The S-curve factor controls the amount of rounding applied to the trapezoidal profiles. A value of 0 sets no rounding. A value of 10 sets maximum rounding.

Using S-curves increases the time required for the movement to complete.

SRAMP is applied to the FORWARD, MHELICAL, MOVE, MOVEABS, MOVECIRC and REVERSE commands.

Precautions: The S-curve factor must not be changed while a move is in progress.

See also: AXIS

5-3-161 STEPLINE

Type: Program Command

Syntax: STEPLINE ["*program_name*"[, *task_number*]]

Description: STEPLINE executes one line (i.e., "steps") in the program specified by *program_name*. The program name can also be specified without quotes. If STEPLINE is executed without program name on the command line the current selected program will be stepped. If STEPLINE is executed without program name in a program this program will be stepped.

If the program is specified then all occurrences of this program will be stepped. If there is no copy of the program running, then one will be started on the next available task. If the task is specified as well then only the copy of the program running on the specified task will be stepped. If there is no copy of the program running on the specified task then one will be started on it.

Arguments:

program_name
The name of the program to be stepped.

task_number
The number of the task with the program to be stepped. Range: [1,5].

See also: RUN, SELECT, STOP, TROFF, TRON

Examples:

Example 1

```
>> STEPLINE "conveyor"
```

Example 2

```
>> STEPLINE "maths",2
```

5-3-162 STOP

Type: Program Command

Syntax: STOP ["*program_name*"[, *task_number*]

Description: STOP halts execution of one program specified with *program_name*. The program name can also be specified without quotes. If the program name is omitted, then the currently selected program will be halted.

In case of multiple executions of a single program on different tasks the *task_number* can be used to specify the specific task to be stopped.

Arguments:

program_name
The name of the program to be stopped.

task_number
The number of the task with the program to be stepped. Range: [1,5].

See also: HALT, RUN, SELECT

Examples:

Example 1

```
>> STOP progname
```

Example 2

The lines from *label* on will not be executed in this example.

```
   STOP
label:
   PRINT var
   RETURN
```

5-3-163 TABLE

Type: System Command

Syntax: TABLE(*address, value*{, *value*})
TABLE(*address*)

Description: TABLE loads data to and reads data from the Table array. The Table has a maximum length of 16,000 elements. The table values are floating-point numbers with fractions. The table can also be used to hold information, as an alternative to variables. The TABLE command has two forms.

- TABLE(*address*, *value*{, *value*}) writes a sequence of values to the Table array. The location of the element is specified by *address*. The sequence can have a maximum length of 20 elements.
- TABLE(*address*) returns the table value at that entry.

A value in the table can be read only if a value of that number or higher has been previously written to the table. For example, printing TABLE(1001) will produce an error message if the highest table location previously written to the table is location 1000. The total Table size is indicated by the TSIZE parameter. Note that this value is one more than the highest defined element address.

The table entries are backed up by a battery. The table can be deleted with by using DEL “TABLE” or NEW “TABLE” on the command line.

Precautions: Applications like the CAM command, CAMBOX command and the SCOPE command in Motion Perfect all use the same Table as the data area. Do not use the same data area range for different purposes.

⚠ **Caution** If the voltage of the backup battery drops, Table and VR data will be lost. This can happen when the power to the MC Unit is turned OFF for a long period of time. You must rewrite table data, e.g., from a program, whenever the backup battery has been drained. The Low Battery Flag will turn ON when the voltage of the backup battery has dropped. Also the BATTERY_LOW system parameter will become TRUE. Refer to *3-1-2 Overview of IR/CIO Area Allocations* and
5-3-28 BATTERY_LOW for detailed information.

Arguments: ***address***
The first location in the Table to read or write. Range: [0,15999]
value
The value to write at the given location and at subsequent locations.

See also: BATTERY_LOW, CAM, CAMBOX, DEL, NEW, SCOPE, TSIZE, VR

Examples: **Example 1**

```
TABLE(100,0,120,250,370,470,530,550)
```

The above line loads the following internal table:

Table Entry	Value
100	0
101	120
102	250
103	370
104	470
105	530
106	550

Example 2
The following line will print the value at location 1000.

```
>> PRINT TABLE(1000)
```

5-3-164 TAN

Type: Mathematical Function

Syntax: TAN(*expression*)

Description: TAN returns the tangent of the *expression*. The *expression* is assumed to be in radians.

Arguments: ***expression***
Any valid BASIC expression.

Example:
```
>> print TAN(PI/4)
1.0000
```

5-3-165 TICKS

Type: Task Parameter

Description: TICKS contains the current count of the task clock pulses. TICKS is a 32-bit counter that is decremented on each servo cycle. TICKS can be written and read. It can be used to measure cycles times, add time delays, etc.

Each task has its own TICKS parameter. Use the PROC modifier to access the parameter for a certain task. Without PROC the current task will be assumed.

Example:
```
delay:
   TICKS = 3000
   OP(9,ON)
test:
   IF TICKS< = 0 THEN
      OP(9,OFF)
   ELSE
      GOTO test
   ENDIF
```

5-3-166 TRIGGER

Type: Motion Perfect Command

Syntax: `TRIGGER`

Description: TRIGGER starts a previously set up SCOPE command.

Note Motion Perfect uses TRIGGER automatically for its oscilloscope function.

See also: SCOPE

5-3-167 TROFF

Type: Program Command

Syntax: `TROFF` ["*program_name*"]

Description: TROFF suspends a trace at the current line and resumes normal program execution for the program specified with *program_name*. The program name can also be specified without quotes. If the program name is omitted, the selected program will be assumed.

Arguments: ***program_name***
The name of the program for which to suspend tracing.

See also: SELECT, TRON

Example:
```
>> TROFF "lines"
```

5-3-168 TRON

Type: Program Command

Syntax: `TRON`

Description: TRON creates a breakpoint in a program that will suspend program execution at the line following the TRON command. The program can then for example be executed one line at a time using the STEPLINE command.

- Program execution can be resumed without using the STEPLINE command by executing the TROFF command.
- The trace mode can be stopped by issuing a STOP or HALT command.
- Motion Perfect highlights lines containing TRON in the Edit and Debug Windows.

See also: TROFF

Example:
```
TRON
MOVE(0,10)
MOVE(10,0)
TRON
MOVE(0,-10)
MOVE(-10,0)
```

5-3-169 TRUE

Type: Constant

Description: TRUE returns the numerical value -1.

Note A constant is read-only.

Example:
```
test:
   t = IN(0) AND IN(2)
   IF t = TRUE THEN
      PRINT "Inputs are ON"
   ENDIF
```

5-3-170 TSIZE

Type: System Parameter

Description: TSIZE returns the size of the Table array, which is one more than the currently highest defined table element.

TSIZE is reset to zero when the Table array is deleted using DEL “TABLE” or NEW “TABLE” on the command line.

Note This parameter is read-only.

See also: DEL, NEW, TABLE

Example: The following example assumes that no location higher than 1000 has been written to the Table array.
```
>> TABLE(1000,3400)
>> PRINT TSIZE
1001.0000
```

5-3-171 UNITS

Type: Axis Parameter

Description: The UNITS axis parameter contains the unit conversion factor. The unit conversion factor enables the user to define a more convenient user unit like m, mm or motor revolutions by specifying the amount of encoder edges to include a user unit.

Axis parameters like speed, acceleration, deceleration and the motion control commands are specified in these user units.

Precautions: UNITS can be any non-zero value, but it is recommended to design systems with an integer number of encoder pulses per user unit. Changing UNITS will affect all axis parameters which are dependent on UNITS in order to keep the same dynamics for the system.

See also: AXIS, PP_STEP

Example: A leadscrew arrangement has a 5mm pitch and a 1,000-pulse/rev encoder. The units must be set to allow moves to be specified in mm.
The 1,000 pulses/rev will generate 1,000 x 4 = 4,000 edges/rev. One rev is equal to 5mm. Therefore, there are 4,000/5 = 800 edges/mm. UNITS is thus set as following.

```
>> UNITS = 1000*4/5
```

5-3-172 VERSION

Type: System Parameter

Description: VERSION returns the version number of the BASIC language installed in the MC Unit.

Note This parameter is read-only.

Example:

```
>> PRINT VERSION
1.5000
```

5-3-173 VFF_GAIN

Type: Axis Parameter

Description: VFF_GAIN contains the speed feed forward gain. The speed feed forward output contribution is calculated by multiplying the change in demand position with VFF_GAIN. The default value is zero.
Adding speed feed forward gain to a system decreases the following error during a move by increasing the output proportionally with the speed.
See section *1-4-2 Servo System Principles* for more details.

Precautions: In order to avoid any instability the servo gains should be changed only when the SERVO is OFF.

See also: AXIS, D_GAIN, I_GAIN, OV_GAIN, P_GAIN

5-3-174 VP_SPEED

Type: Axis Parameter

Description: VP_SPEED contains the speed profile speed in user units/s. The speed profile speed is an internal speed which is accelerated and decelerated as the movement is profiled.

Note This parameter is read-only.

See also: AXIS, MSPEED, UNITS

Example:

```
'Wait until at command speed
MOVE(100)
WAIT UNTIL SPEED = VP_SPEED
```

5-3-175 VR

Type: System Command

Syntax: VR(*expression*)

Description: The VR command calls the value of or assigns a value to a global variable. These VR variables hold real numbers and can be easily used as an element or as an array of elements. The MC Unit has in total 251 VR variables. The variables are accessed as variables 0 to 250.
The VR variables can be used for several purposes in BASIC programming.

- The VR variables are backed up by a battery and are not cleared between start-ups.

- The VR variables are globally shared between tasks and can be used for communications between tasks. The programs must be written so that only one program writes to the same global variable at the same time.

⚠Caution If the voltage of the backup battery drops, Table and VR data will be lost. This can happen when the power to the MC Unit is turned OFF for a long period of time. You must rewrite table data, e.g., from a program, whenever the backup battery has been drained. The Low Battery Flag will turn ON when the voltage of the backup battery has dropped. Also the BATTERY_LOW system parameter will become TRUE. Refer to *3-1-2 Overview of IR/CIO Area Allocations* and *5-3-28 BATTERY_LOW* for detailed information.

Arguments: ***expression***
Any valid BASIC expression. Range: [0,250].

See also: BATTERY_LOW, CLEAR_BIT, READ_BIT, SET_BIT, TABLE

Examples: **Example 1**

In the following example, the value 1.2555 is placed into VR variable 15. The local variable *val* is used to name the global variable locally:

```
val = 15
VR(val) = 1.2555
```

Example 2

A transfer gantry has 10 put down positions in a row. Each position may at any time be full or empty. VR(101) to VR(110) are used to hold an array of ten 1's and 0's to signal that the positions are full (1) or empty (0). The gantry puts the load down in the first free position. Part of the program to achieve this would be as follows:

```
movep:
   MOVEABS(115)   'Move to first put down position
   FOR VR(0) = 101 TO 110
      IF (VR(VR(0)) = 0) THEN GOSUB load
      MOVE(200)   '200 is spacing between positions
   NEXT VR(0)
   PRINT "All positions are full"
   WAIT UNTIL IN(3) = ON
   GOTO movep

load:             'Put load in position and mark array
   OP(15,OFF)
   VR(VR(0)) = 1
   RETURN
```

The variables are backed up by a battery so the program here could be designed to store the state of the machine when the power is OFF. It would of course be necessary to provide a means of resetting completely following manual intervention.

Example 3

```
loop:             'Assign VR(65) to VR(0) multiplied by
                  'axis 1 measured position
   VR(65) = VR(0)*MPOS AXIS(1)
   PRINT VR(65)
   GOTO loop
```

5-3-176 WA

Type: System Command

Syntax: WA(*time*)

Description: WA holds program execution for the number of milliseconds specified for time. The command can only be used in a program.

Arguments: ***time***
The number of milliseconds to hold program execution.

Example: The following lines would turn ON output 7 two seconds after turning OFF output 1.

```
OP(1,OFF)
WA(2000)
OP(7,ON)
```

5-3-177 WAIT IDLE

Type: System Command

Syntax: `WAIT IDLE`

Description: WAIT IDLE suspends program execution until the base axis has finished executing its current move and any buffered move. The command can only be used in a program.

WAIT IDLE works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

Precautions: The execution of WAIT IDLE does not necessarily mean that the axis will be stationary in a servo motor system.

See also: AXIS, WAIT LOADED

Example:

```
MOVE(100)
WAIT IDLE
PRINT "Move Done"
```

5-3-178 WAIT LOADED

Type: System Command

Syntax: `WAIT LOADED`

Description: WAIT LOADED suspends program execution until the base axis has no moves buffered ahead other than the currently executing move. The command can only be used in a program.

This is useful for activating events at the beginning of a move, or at the end when multiple moves are buffered together.

WAIT LOADED works on the default basis axis (set with BASE) unless AXIS is used to specify a temporary base axis.

See also: AXIS, WAIT IDLE

Example:

```
‘Switch output 8 ON at start of start of MOVE(500)
‘and OFF at end
MOVE(800)
MOVE(500)
WAIT LOADED
OP(8,ON)
MOVE(400)
WAIT LOADED
OP(8,OFF)
```

5-3-179 WAIT UNTIL

Type: System Command

Syntax: `WAIT UNTIL` *condition*

Description: WAIT UNTIL repeatedly evaluates the *condition* until it is TRUE. After this program execution will continue. The command can only be used in a program.

Arguments: ***condition***
Any valid BASIC logical expression.

Examples: **Example 1**
In this example, the program waits until the measured position on axis 0 exceeds 150, and then starts a movement on axis 7

```
WAIT UNTIL MPOS AXIS(0)>150
MOVE(100) AXIS(7)
```

Example 2
The expressions evaluated can be as complex as you like provided they follow BASIC syntax, for example:

```
WAIT UNTIL DPOS AXIS(2)< = 0 OR IN(1) = ON
```

The above line would wait until the demand position of axis 2 is less than or equal to 0 or input 1 is ON.

5-3-180 WDOG

Type: System Parameter

Description: The WDOG parameter contains the software switch used to control the enable relay contact, which is used to enable all drivers. This parameter should be turned ON before executing moves. WDOG can be turned ON and OFF under program control and on command line. All non-zero values are considered ON. The analogue outputs of the axis will always be zero when the WDOG is OFF.
The enable relay will automatically turn OFF when a MOTION_ERROR occurs. A motion error occurs when the AXISSTATUS state for one of the axes matches the ERRORMASK setting. In this case the enable relay (WDOG) will be turned OFF, the MOTION_ERROR parameter will have value 1 and the ERROR_AXIS parameter will contain the number of the first axis to have the error.

Precautions: The WDOG parameter can be executed automatically by Motion Perfect when the Drives Enable Button is clicked on the control panel.

See also: AXISSTATUS, ERROR_AXIS, ERRORMASK, MOTION_ERROR, SERVO

5-3-181 WHILE WEND

Type: Structural Command

Syntax:

```
WHILE condition
   <commands>
WEND
```

Description: The WHILE ... WEND loop allows the program segment between the WHILE and the WEND statement to be repeated a number of times until the *condition* becomes FALSE. In that case program execution will continue after WEND.

Precautions: WHILE ... WEND loops can be nested without limit.

Arguments: ***condition***
Any valid logical BASIC expression.

See also: FOR, REPEAT

Example:

```
WHILE IN(12) = OFF
   MOVE(200)
   WAIT IDLE
```

```
   OP(10,OFF)
   MOVE(-200)
   WAIT IDLE
   OP(10,ON)
WEND
```

5-3-182 XOR

Type: Logical Operator

Syntax: *expression_1* XOR *expression_2*

Description: XOR performs an XOR function between corresponding bits of the integer parts of two valid BASIC expressions.

The XOR function between two bits is defined as follows:

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Arguments: ***expression_1***

Any valid BASIC expression.

expression_2

Any valid BASIC expression.

Example: `a = 10 XOR (2.1*9)`

The parentheses are evaluated first, but only the integer part of the result, 18, is used for the operation. Therefore, this expression is equivalent to the following:

`VR(0)=10 XOR 18`

The XOR is a bit operator and so the binary action taking place is as follows:

```
            01010
XOR         10010
-----------------
            11000
```

The result is, therefore, 24.

SECTION 6
Programming Environment

The MC Unit is programmed using the Motion Perfect programming software. The Motion Perfect package is Microsoft Windows based and provides the user to program, monitor and debug motion based applications.

6-1 Motion Perfect Features 158
6-2 Motion Perfect Requirements 158
6-3 Going Online with the MC Unit 158
6-4 Motion Perfect Projects 159
6-4-1 Motion Perfect Project Manager 159
6-4-2 Using Motion Perfect on a MC Unit for the First Time 161
6-5 Motion Perfect Desktop 161
6-5-1 Control Panel 162
6-5-2 Editing and Running Simple Programs 163
6-6 Motion Perfect Tools 164
6-6-1 Terminal 164
6-6-2 Editor 165
6-6-3 Axis Parameters 168
6-6-4 Controller Configuration 169
6-6-5 VR and Table Editors 169
6-6-6 I/O Status Window 170
6-6-7 Full Controller Directory 171
6-6-8 Jog Screen 171
6-6-9 Oscilloscope 172
6-7 Suggestions and Precautions in Using Motion Perfect 177

6-1 Motion Perfect Features

Motion Perfect provides the following features.

- Using the Project Manager to maintain a consistent copy of application programs on the computer.
- Creating, copying, renaming, deleting, editing, running and debugging programs on the MC Unit.
- Using the Control Panel, Full Controller Directory, Axis Parameters Window, and I/O status to monitor the MC Unit and to control its status.
- Using the Program Debugger, Axis Parameters Window, Software Oscilloscope Window and Jog Axes Window to adjust the servo system.

It is possible to open several windows on the Motion Perfect desktop and run them simultaneously. A user could be stepping through a program displayed in an Editor Window, while checking the program's output and entering input characters via a separate terminal Window, and while also monitoring and updating the axis parameters and I/O status in their Windows.

Note Refer also to the Motion Perfect Help files for further details on this software package.

6-2 Motion Perfect Requirements

The C200HW-MC402E requires Motion Perfect version 2.0 or later. Please note that previous versions of the package will not work with this MC Unit.

The following are required to run Motion Perfect version 2.0.

1. IBM Personal Computer or 100% compatible.
2. Microsoft Windows 95, 98, 2000 or NT 4.0.
3. 66 MHz 486 based processor (133 MHz Pentium recommended).
4. 16 MB RAM (32 MB recommended).
5. 10 MB of hard disk space.
6. Enhanced serial communications port (UART 16550).
7. 800 x 600 pixel display or higher resolution with at least 256 colors.
8. Mouse or tracker ball.

6-3 Going Online with the MC Unit

Motion Perfect can be connected to the MC Unit once the MC Unit is powered-up and running in order to use all features. After installation Motion Perfect can be started by using the Start Button.

Note The computer must be connected to the MC Unit using a RS-232C Serial Cable (by OMRON) between a COM port on the computer and the MC Units RS-232C Programming Port. Refer to *2-3-3 Serial Port Connections* for details.

When started, Motion Perfect will display its introductory splash screen whilst looking for any controllers connected to the computer.

The status of the connection to the controller is displayed on the screen. The statements indicate at which COM port Motion Perfect is currently checking for a Motion Controller and which settings are used. The screen will confirm if Motion Perfect has found a controller or will indicate that no controller is found.

Motion Perfect will be disconnected when no suitable controllers have been found. The offline terminal will be shown. Refer to *SECTION 7 Troubleshooting*.

6-4 Motion Perfect Projects

Motion Perfect facilitates the works with MC Unit applications by using projects, which are a valuable aid in efficient application design and development. Projects are stored on the computer and each project contains the MC Unit programs, parameters and data required for one motion application. Managing each application as one project enables effective version control and provides a mechanism for verifying the application programs on the MC Unit.

6-4-1 Motion Perfect Project Manager

The project manager is a background process that automatically maintains consistency between the programs on the MC Unit and the project on the computer. When you edit a program in the Motion Perfect editor, it changes both copies of the program. This avoids the slow process of uploading and downloading programs and ensures that there is always a backup of changes you make. As programs are created, copied or erased on the MC Unit using the Motion Perfect tools, the project is updated so that the files and programs are always consistent.

Project Backups

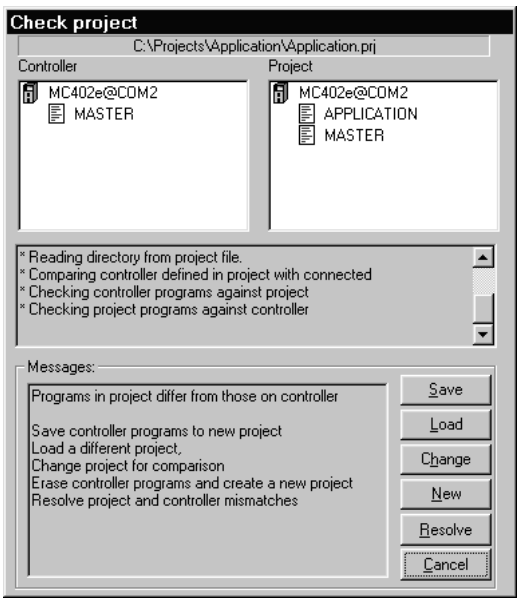
A backup copy of the project is stored on the computer after on-line operation has been successfully started. The backup copy can be loaded if the MC Unit version of the programs become corrupted for any reason by selecting ***Revert to backup*** from the File Menu.

The backup file will be overwritten each time a project is opened. If you open a new project during a development session, the new projects backup copy will overwrite the previous backup.

Consistency Check

When Motion Perfect starts, it always performs a consistency check between any programs on the MC Unit and the current project files on the computer. It will only enable its tool site when it has successfully verified that the programs in the controller matches the project on the computer. The CRC values of the programs are compared to perform the check. The Check Project Window shows the status of the check. When both projects are consistent, the statement “Project check OK” is shown in the message field.

If both projects differ, the window will display the options for the user to determine how to resolve this inconsistency. The Check Project Window is shown here:



The window enables the user to select the required option to resolve the discrepancy. The options available include:

Function	Purpose
Save	Save controller programs to new project. Create a new project on the computer and save the programs to this project.
Load	Load a different project. Select a new project on the computer and load this project into the controller.
Change	Change project for comparison. Select a different project on the computer with which to perform the programs consistency check.
New	Erase controller programs and create a new project. Create a new empty project on the computer and delete all programs on the controller.
Resolve	Resolve project and controller mismatches. Continue to check the project, enabling the user to resolve each individual program inconsistency by either saving the project version into the controller or loading the controller version into the project.
Cancel	Run Motion Perfect without connection to the controller.

You can force Motion Perfect to verify that the two copies are identical at any time by selecting ***Check project*** from the File Menu.

6-4-2 Using Motion Perfect on a MC Unit for the First Time

If this is the first time the MC Unit has been used with Motion Perfect and you do not have any programs on the MC Unit, click the New Button in the Check Project Options Window and then click the Yes Button when asked.

New Project Window

The New Project Window will open enabling you to enter the required name of the new project and specify the directory on your computer in which to store the project.

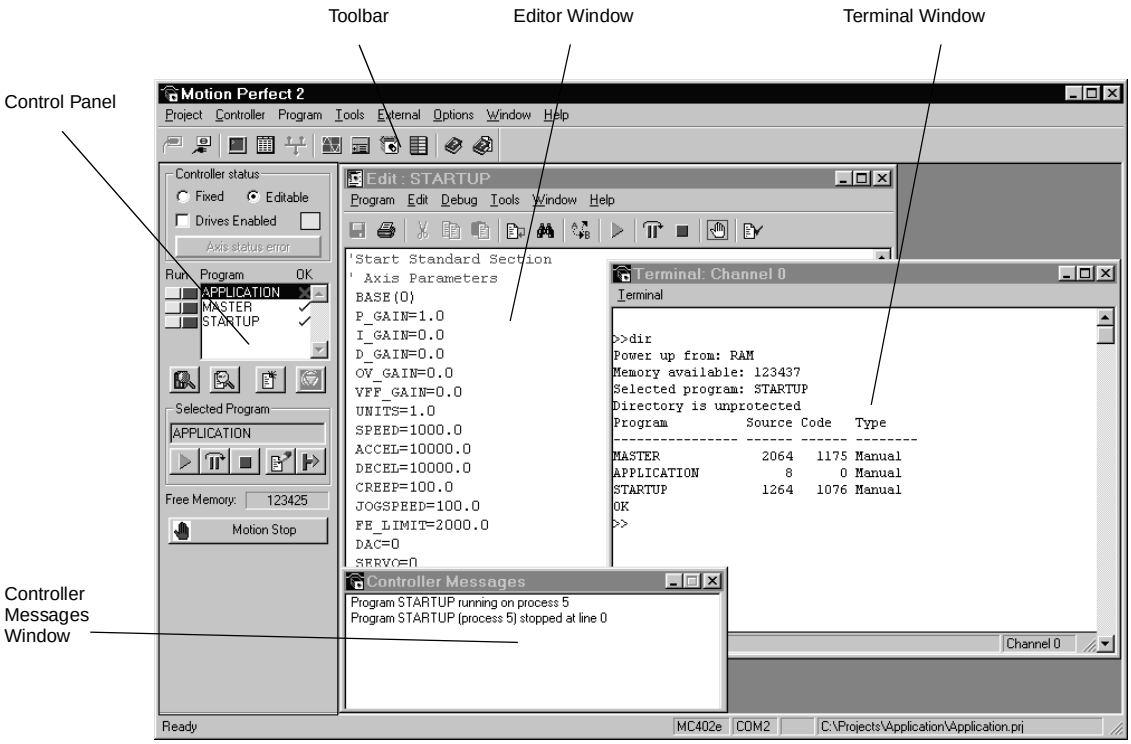
1,2,3...

1. Type a suitable project name in the Project Name text box, and then use the Disk Directory box to move to the directory in which to store the project.
2. If you wish to create a new directory on your computer, then move to the parent directory and click on the Create Directory button. Type the name of the new directory in the New Directory Window.
3. After selecting the required directory and entering the new project name, click the Create Button.
4. If the project path and name already exist, a window will appear asking whether to overwrite the existing project. If you confirm, the previous project will be overwritten and lost.
5. When a new project has been created, the Check Project Window will be displayed, with empty MC Unit Program and Project Program List Boxes and a 'Project check OK' message. Click the Ok Button to continue.

You have now created a new project on your computer, the Motion Perfect desktop will open, and all its facilities will be available.

6-5 Motion Perfect Desktop

Once Motion Perfect has verified that the contents of the controller and the project on the computer are consistent, the Motion Perfect Desktop will appear. The desktop work area of Motion Perfect is where you will open up the windows to use when editing programs and using the Motion Perfect tools. The general look of the desktop is displayed below.



6-5-1 Control Panel

Motion Perfect is equipped with a Control Panel that is used to control program execution and the MC Unit while editing and debugging the programs. The Control Panel will appear after the initial opening window on the left of the main Motion Perfect Window:

Controller Status

The ***Fixed/Editable buttons*** determine to fix the programs from RAM into flash EPROM. When the project is fixed, the programs cannot be modified with Motion Perfect. To continue editing the programs, click Editable.

The ***Drives enabled button*** toggles the state of the enable (watchdog) relay on the controller, which controls the drivers. See *5-3-180 WDOG* for details.

The ***Axis status error button*** monitors the Motion errors of the MC Unit. This button is normally greyed out, unless a motion error occurs on the controller. When an error does occur, you can use this button to clear the error condition. This is equivalent to using the DATUM(0) command. See *5-3-46 DATUM* for details.

Program List Box

The Program List Box in the middle of the Control Panel will show a list of the programs in the project. There are two buttons next to each program name to control the execution of this program.

The ***Run/Stop button (red)*** shows that the program is stopped and can be clicked to start program execution.

When a program is being executed, the program name in the Program List Box will appear in italics, the MC Unit task number on which the program is running will appear alongside the program name, and the color of the Run/Stop button will change into green.

To stop the program click the Run/Stop button again to stop the program. A program cannot be edited while it is being executed.

The ***Step button (yellow)*** can be used to step through the program. The Run/Stop Button will turn yellow and one line of the program will be executed each time the Run/Stop button is clicked. When the Step button is clicked again, the program will be run normally.

When a program is being stepped, a green bar will appear in the Editor Window, highlighting the line of the program about to be executed.

Programs are compiled before execution. If there are any compilation errors in the program, a window will appear briefly describing the error and giving the number of the line containing the error. You can correct the error and repeat the process.

Shortcut Buttons

Underneath the Program List Box you can find four shortcut buttons which are used for (from left to right):

- Controller configuration
- Full controller directory
- Create new program
- Halt all programs

Selected Program Box

The Select Program box displays the current selected program and the available buttons, which can be performed on the selected program. These five available functions are (from left to right):

- Run
- Step
- Stop
- Edit
- Set power up mode

Refer to *4-5-3 Program Execution* and *5-3-149 RUNTYPE* for details on setting the power up mode.

Free Memory

The Free Memory field indicates the remaining free memory available on the controller.

Motion Stop

The ***Motion Stop button*** stops all programs and cancels all moves in case of emergency.

6-5-2 Editing and Running Simple Programs

This section provides a couple of typical examples of a simple programming session using Motion Perfect. The following procedure assumes that the MC Unit is already on-line with Motion Perfect and a new project has been created.

Example 1

1,2,3...

1. Create a new program in the new project by selecting ***New*** from the Program Menu or by clicking on the Control Panel's shortcut button "Create New Program".
2. Name your program 'LED1' and click the ***OK Button***.
3. Using the editor, enter a simple program to flash the indicator connected to output 8. Type the program in lower case. When you press the Return Key, Motion Perfect will update the program on the computer and in the MC Unit, and will replace the BASIC keywords with their tokenised versions in upper case.

```
      DISPLAY = 5
loop:
   OP(8,ON)
   WA(1000)
   OP(8,OFF)
   WA(1000)
   GOTO loop
```

The program can now be run, stepped and stopped without closing the Editor Window. The Program Menu can be used, but it is easier to use the Control Panel as described in the previous section. The Editor Window has similar buttons itself to do the same.

When the program is executed, the LED indicator for output 8 will flash.

Note The command line will also remain available for immediate commands if the task 0 Terminal Window is open.

Compilation The system will compile and link the program before running it. If the compiler detects errors in the program, it will not run, but will print the line number at which the error occurred. The line can be located by looking at the current line number displayed in the bottom right of the Editor Window's status bar or by selecting ***Goto*** from the Edit Menu.

Example 2

A similar second program can be made based on the first program. This can be done quickly by copying the LED1 program and then editing it.

1,2,3...

1. Select ***Copy program*** from the Program Menu and copy the LED1 program, calling the new program 'LED2'.
2. Select the LED2 program and press the ***Edit Button*** to open it.
3. Change the LED2 program to control a different OP with a different period. Refer to for *SECTION 5 BASIC Motion Control Programming Language* details on programming.
4. Executed the programs together.

The Run and Step Buttons on the Control Panel will control only the currently selected program Use the Run/Stop and Step Button or the menus to control other programs at the same time.

6-6 Motion Perfect Tools

This section describes the main Motion Perfect tools.

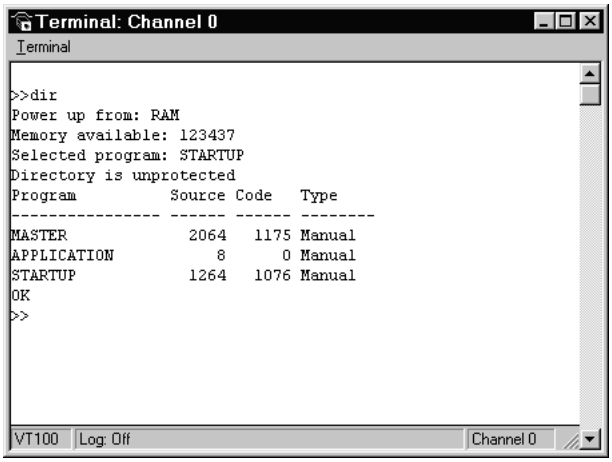
1. Terminal
2. Editor
3. Axis Parameters
4. Controller Configuration
5. VR and Table Editors
6. I/O Status
7. Full Controller Directory
8. Jog Axes
9. Oscilloscope

6-6-1 Terminal

The Terminal Window provides a direct connection to the MC Unit. Most commands, functions and parameter read/writes can be issued directly on the command line.

Most of the functions that must be performed during the installation, programming and final setup of a system with a MC Unit have been automated by the

options available in the Motion Perfect Menus. A Terminal Window is shown in the following display.



Up to four Terminal Windows can be opened simultaneously over the single serial port. Channel 0 must be used to issue commands. Channels 5, 6 and 7 can be used to provide I/O windows to programs running on the MC Unit.

Channel Number

The Channel Number Combo Box can be used to select one of the valid async communications channels.

VT100 Emulation

The MC Unit expects to talk to a terminal that accepts the DEC VT100 terminal protocol. This setting can be used for the Terminal Window to emulate a VT100 terminal.

ASCII Emulation

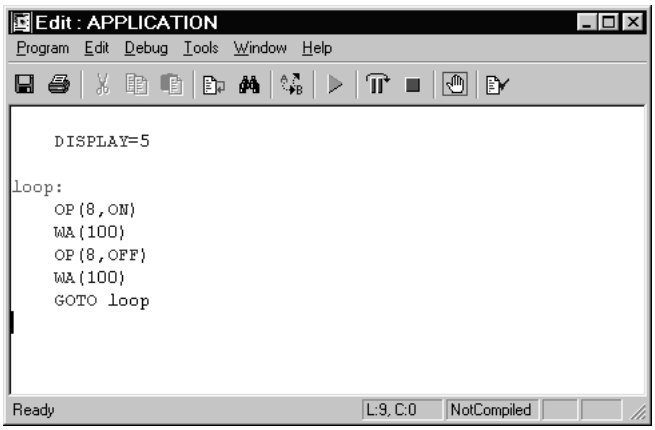
This mode will echo the ASCII description for the non-printing characters received. Also, CR and LF will cause the corresponding action.

6-6-2 Editor

This section describes the Editor used to edit BASIC programs for the MC Unit. The Editor is a fully featured Windows-based tool. An Editor Window will be opened when a new program is created or an existing program is selected for editing.

When the cursor is moved off the current line, any changes made to this line are sent to the MC Unit, which performs syntax checking, tokenises the line (all recognized BASIC keywords are converted to upper case), and returns the tokenised result to the window. When an Editor Window is closed, the project file is updated with the modified program.

Note It is not possible to open a new Editor Window while any program is running on the MC Unit.



Creating and Opening Programs

There are several ways that programs can be opened or created.

Opening Programs

Existing programs can be edited by opening an Editor Window using one of the following methods.

- Select ***Edit*** from the Program Menu and then selecting the required program.
- Click the Edit Button on the Control Panel to open an Editor Window for the selected program.
- Double-click a program in the Program List Box on the Control Panel.

Creating Programs

New programs can be created using one of the following methods.

- Select ***New*** from the Program Menu. The default name can be changed before opening the Editor Window. Click into the Program Name Text Box, enter the new name and then press the Edit Button.
- Click the Create New program button on the Control Panel.

When opening an Editor Window, Motion Perfect performs a CRC check between the program on the MC Unit and the program in the project. If the CRCs are different, the user will be advised to perform a project check to obtain further information on the differences.

Basic Editing Operations

The basic editing operations that can be used in an Editor Window are outlined below. The operation can be accessed by selecting the corresponding button on the top of the Editor Window or can be selected from one of the menu's of the window.The operations correspond to the buttons displayed in the picture below (from left to right).

Saving Program This enables the user to force the program to be saved on the computer harddisk. Motion Perfect saves the file automatically when the Editor Window is closed or the program is compiled.

Printing Program This will print the code of the program.

Cut, Copy and Paste Windows-style cut, copy paste operation can be performed using the mouse and/or the keyboard. Use the following procedure to cut or copy text.

Select the text, and cut or copy it to the clipboard.

Move the cursor to the insert point, and paste the text on the clipboard.

Listing and Jumping to Labels

A list of all labels in the program in the current Editor Window will be displayed. To jump to a specific label, click the desired label in the display to enter it in the text box at the bottom of the window and press the OK button. The cursor will move to the specified label in the program. Alternatively, a specific line number can be selected by entering the value of the line number text box at the bottom of the window, and the pressing the OK button.

Finding Text

The program in the current Editor Window can be searched for a specific text string. One can specify the search to be case sensitive and the search direction. The user can continue the program while the Find Window is displayed, by simply clicking back to the Editor Window. The Find Window will remain on the display until the Cancel Button is clicked.

Replacing Text

Text found in an Editor Window can be replaced with a specified text string. Enter both strings in the appropriate fields. The following buttons are available in the Find and Replace Window:

Button	Function
FindNext	A simple search will be made for the specified string.
Replace	A specified search string will be replaced with a replace string.
ReplaceAll	All occurrences of the search string will be replaced from the current cursor position to the beginning or end of the program, depending upon the search direction.

Run, Step and Stop

These operations are used to run the program, run a single line in the program and stop the program. These operations can also be found on the control panel (same buttons).

Add breakpoint

In the Editor Window breakpoints can be added to enable easy debugging. Debugging is explained in the Debugging part of this section below.

Compiling

This operation forces the program to be compiled.

Debugging

The Motion Perfect debugger allows you to run a program directly from the Editor Window in a special trace mode, executing one line at a time (known as stepping) whilst viewing the line in the window. It is also possible to set breakpoints in the program, and run it at normal speed until it reached the breakpoin.

Any open Editor Windows will automatically enter the 'Debug Mode - Read Only' when programs are running on the Motion Controller. Hence, breakpoints are set in the Editor Window, and the code viewed in the same window in debug mode when the program is running.

Stepping Through a Program

The next line in a program can be executed by doing one of the following:

- Use the Step button (yellow) alongside the required program name in the Program List box on the Control Panel.
- If the required program is currently selected, see Selected Program box of the Control Panel, then push the Step button of this box.
- Push the Step button of the Editor Window toolbar.
- Selecting Start Stepping... from the Program menu. If one program is executing on several tasks, then the task number can also be specified.

The next program line to be executed will be highlighted in the Editor Window with a green background. The operation can be repeated to step multiple lines.

Breakpoints

Breakpoints are special place markers in the code which allow us to identify a particular section (or sections) of the program when debugging the code. At

the point on which the breakpoint is inserted, the program will pause and return control to Motion Perfect. This is enabling to check the current state of the controller or single step through the code of the program. Breakpoints are indicated in the program using the TRON command.

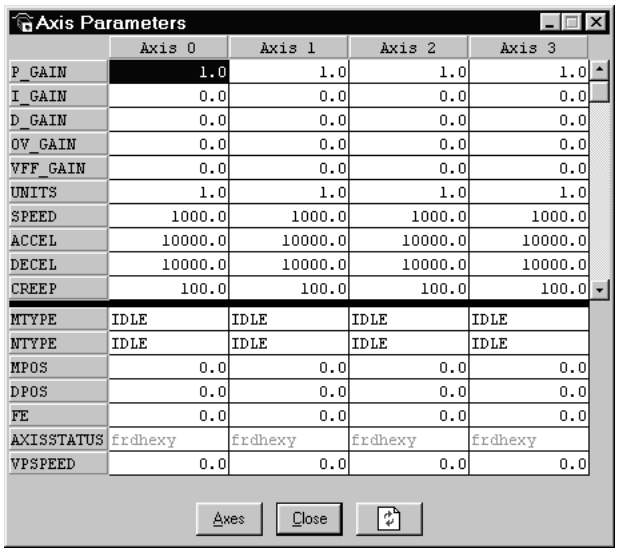
Breakpoints can be set by moving the cursor to the required line, and then either

- Typing command TRON on this line.
- Pushing the Add Breakpoint button on the Editor Window toolbar.
- Selecting ***Toggle Breakpoint*** from the Program Menu.
- Pressing Ctrl–B from the keyboard.

A TRON command will be inserted at the current line in the program, indicated by highlighting. The breakpoint can be removed to selecting the same operation again or to just by removing the line manually. All breakpoints can be removed from a program by selecting ***Clear All Breakpoints*** from the Debug Menu.

6-6-3 Axis Parameters

The Axis Parameters Window allows the user to set and read the axis parameter settings. This window works like a Windows-based spread sheet. The Axis Parameter Window is shown below.



The Axis Parameters Window is made up of a table of cells separated into two banks, bank 1 at the top and bank 2 at the bottom.

- Bank 1 contains the values of parameters that can be changed by the user. The values can be changed by clicking on it and entering the new value.
- Bank 2 is read-only and contains the values which are set by the system software of the MC Unit as it processes the BASIC commands and monitors the status. These values are updated continuously at a specified rate.

The following operations are possible on the Axis Parameters Window.

- The user is able to change the size of the window. The black dividing bar can be repositioned to change the space occupied by the two banks.
- When the user changes the UNITS parameter for an axis, all the parameters given in user units for that axis will be adjusted by the new factor. These new values will loaded automatically in the screen.

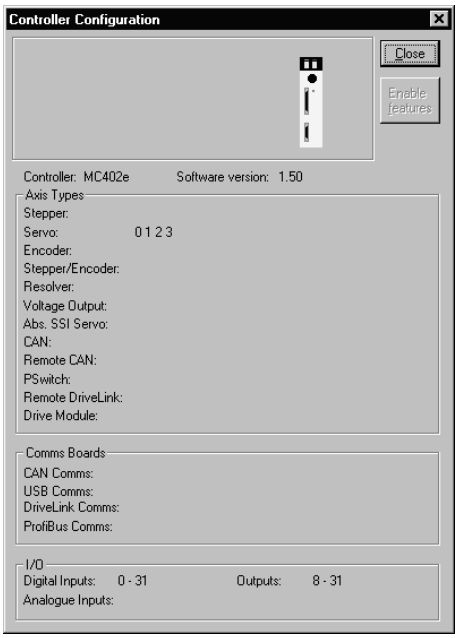
- The AXISSTATUS parameter displays status bits. The characters indicating each bit will turn red and capital if the bit is ON and green if the bit is OFF. The 'frdhexy' characters correspond to

f	Forward limit
r	Reverse limit
d	Datuming
h	Feed hold
e	Following error exceed limit
x	Forward software limit
y	Reverse software limit

- The Axes Button at the bottom of the window can be pressed to access a Window to select the axes that are displayed. By default, the axes set for the last modified start-up program from the File Menu, Jog Axes Window or Axes Parameters Window will be displayed.
- The parameters in the bank 1 section are only read when the screen is first displayed or the parameter is edited by the user. It is possible that if a parameter is changed in the controller then the value displayed may be incorrect. The refresh button will force Motion Perfect to read the whole selection again.

6-6-4 Controller Configuration

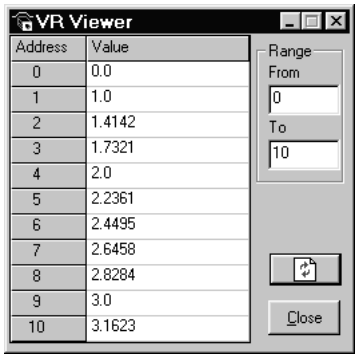
The Controller Configuration Window shows the hardware and software configuration of the MC Unit. The MC Unit configuration can be checked by selecting Controller Configuration from the Controller Menu or the appropriate button of the Control Panel.

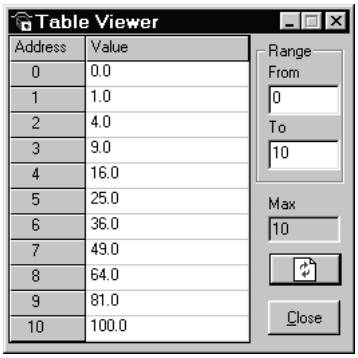


6-6-5 VR and Table Editors

The VR and Table Editor tools provide a spreadsheet style interface to view and modify a range of values in memory. To modify a value, click on the existing value with the mouse and type in the new value and press return. The

change will be immediate and can be made whilst programs are running. Push the refresh button to reload the values.





Range

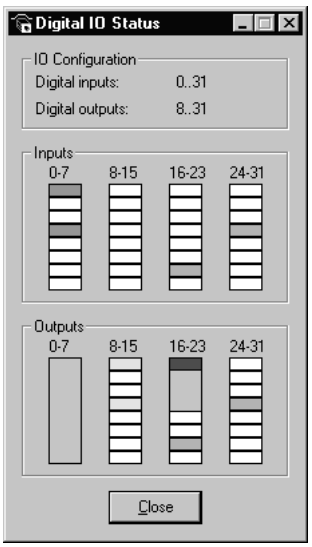
Both in the VR and Table Editor you can select the range of the view by giving the begin and end element. The range of the Table Editor is limited to the highest element, which is specified by the TSIZE system parameter. Both editor show up to a maximum of 100 elements. Use the scroll bar to scroll through the data.

Refresh Button

The editors do not update the shown values automatically. Push the Refresh Button to update the values of the elements or when you have changed the range of elements.

6-6-6 I/O Status Window

The I/O Status Window allows the user to view the status of all the I/O points and toggle the status of the output points. The I/O Status Window is shown in the centre of the screen below. Refer to *Accessing I/O* for a description of the different types of I/O.



Digital Inputs

This shows the total range of input channels on the current Motion Controller.

Digital Outputs

This shows the total range of output channels on the current Motion Controller.

Inputs

This field the status of the inputs of the Motion Controller. The banks contain 8 indicators which show the status of the inputs.

- Physical inputs (0-15): The indicator is green when the input is ON and white when the input is OFF.
- Driver alarms axes 0 to 3 (16-19): The indicator is red when the input is ON and white when the input is OFF.

- Virtual inputs (20-31): The indicator is turquoise when the input is ON and white when the input is OFF.

Outputs

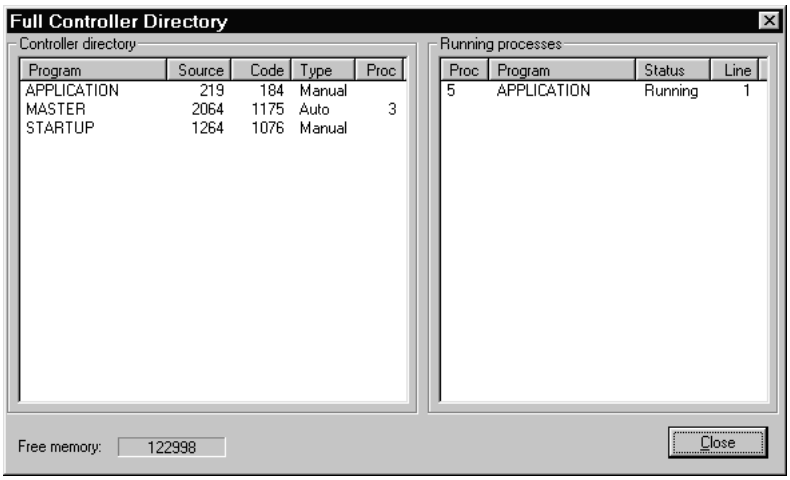
This field the status of the outputs of the Motion Controller. The banks contain 8 indicators which show the status of the outputs. These output points can be put ON or OFF by clicking on the indicators.

- Physical outputs (8-15): The indicator is yellow when the output is ON and white when the output is OFF.
- Driver alarm reset (16): The indicator is red when the output is ON and white when the output is OFF.
- Virtual outputs (20-31): The indicator is turquoise when the output is ON and white when the output is OFF.

Note that the virtual in- and outputs are bi-directional and are controlled by toggling the output indicators. The in- and outputs can be accessed by using controller commands IN and OP. Refer to *5-3-83 IN* and *5-3-115 OP*.

6-6-7 Full Controller Directory

The Full Controller Directory Window dynamically shows details of all programs on the MC Unit, and details of all running tasks or processes. The window can be opened by selecting ***Full Directory*** from the Program Menu or the appropriate button on the Control Panel.



6-6-8 Jog Screen

The Jog Screen can be used to set-up and operate the jogging operation of the motion controller with the bi-directional virtual I/O. The screen sets the axis parameters corresponding to jogging (FWD_JOG, REV_JOG and JOGSPEED) and controls the virtual inputs which are set to jogging. This tool will not use the Fast Jog feature of the controller and therefore the FAST_JOG parameter is assumed to be -1.

Note The jogging inputs which are connected are considered to be active low (normally closed). This implies that jogging is enabled when the input is low and is disabled when the input is high.

The Jog Screen is shown below.

Jog Inputs

The jog inputs must be between 20 and 31, i.e., the virtual bi-directional I/O channels. There are separate inputs for forward and reverse jogging of each axis. When a jog input is set to a valid input number, the corresponding output will be turned ON and then the corresponding FWD_JOG or REV_JOG axis parameter will be set.

Jog Speed Settings

This is the speed at which the jog will be performed, which is given by the JOGSPEED parameter. The value of the speed is limited to the range from 0 to the demand speed given by the SPEED parameter for this axis. This value can be changed by writing directly to this field or by using the jog speed control (up/down) buttons.

Jog Buttons

The screen provides Forward and Reverse Jog Buttons for each axis. When the button is pushed the jogging is activated and the corresponding virtual input will be OFF. Prior to the activation the value of the Jog Speed field will be written to the JOGSPEED parameter. When released this input is ON and the jogging will be stopped.

Warnings Area

The Warnings Area shows the status of the last jog request.

When a Jog Button is pressed, a warning will be given for any of the following:

- The axis is a SERVO axis and the servo is OFF
- The jog speed is 0.
- The acceleration or deceleration rate for this axis is 0
- The forward or reverse jog input is out of range
- There is already a move other than a jog being performed on this axis

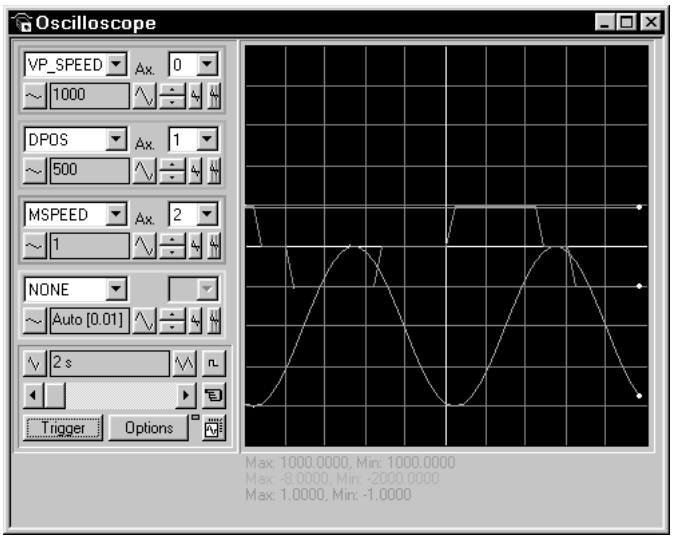
6-6-9 Oscilloscope

The software oscilloscope can be used to trace axis and motion parameters, which is a helpful tool for program development and system setup. The oscilloscope provides four channels, each capable of recording at up to 1,000 samples/s, with manual cycling or program-linked triggering.

The MC Unit records the data at the selected frequency, and then uploads the information to the scope to be displayed. If a larger time base value is used, the data is retrieved in sections, and the trace is seen to be plotted in sections across the display. The positions and other axis parameters are displayed in encoder edges.

Exactly when the MC Unit starts to record the required data depends upon whether it is in Manual or Program Trigger Mode.

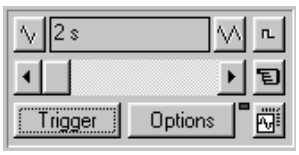
- In Program Trigger Mode, it starts recording data when it encounters a TRIGGER command in a program running on the MC Unit.
- In Manual Mode, it starts recording data immediately.



The Trigger Button can be used to start the scope as soon as the required settings have been made. The scope controls are divided into the two parts: the general controls and the channel specific controls.

General Controls

The oscilloscope general control appear at the bottom left of the oscilloscope window. From here you can control the time base, triggering modes, Table range used and others.



The general controls are explained here:

Time base	The required time base is selected using the up/down scale buttons either side of the current time base scale text box (left hand side button decreases the scale, and the right hand side button increases the scale value.) The value selected is the time per grid division on the display. If the time base is greater than a predefined value, then the data is retrieved from the controller in sections (as opposed to retrieving a compete trace of data at one time.) These sections of data are plotted on the display as they are received, and the last point plotted is seen as a white spot. After the scope has finished running and a trace has been displayed, the time base scale can be changed to view the trace with respect to different horizontal time scales. If the time base scale is reduced, a section of the trace can be viewed in greater detail, with access provided to the complete trace by moving the horizontal scroll bar.
Horizontal Scroll Bar	Once the scope has finished running and displayed the trace of the recorded data, only part of the trace will be displayed if the time base is changed to a faster value. The remainder can be viewed by moving the thumb box on the horizontal scroll bar. Additionally, If the scope is configured to record both motion parameters and plot table data, then the number of points plotted across the display can be determined by the motion parameter. If there are additional table points not visible, these can be brought into view by scrolling the table trace using the horizontal scroll bar. The motion parameter trace will not move.

One-shot/ Continuous Trigger Mode	The One-shot/Continuous Trigger Mode Button toggles between these two modes: **One Shot Trigger Mode (Button raised)** In One-shot Mode, the scope runs until it has been triggered and one set of data recorded by the MC Unit, retrieved and displayed. **Continuous Trigger Mode (Button pressed)** In Continuous Mode the scope continues running, retrieving data from the MC Unit each time it is re-triggered and new data is recorded. The scope continues to run until the Trigger Button is pressed for a second time to stop the scope.
Manual/Program Trigger Mode	The Manual/Program Trigger Mode Button toggles between these two modes. **Manual Mode (Button raised, pointing hand)** In Manual Mode, the MC Unit is triggered and starts to record data immediately after the Trigger Button is pressed. **Program Mode (Button pressed, program listing)** In Program Trigger Mode, the scope starts running when the Trigger Button is pressed. The MC Unit will start recording data when a TRIGGER command is executed in a program running on the MC Unit. After the TRIGGER command is executed by the program and the MC Unit has recorded the required data, the required data is retrieved by the scope and displayed. The scope stops running if in One-shot Trigger Mode, or it waits for the next trigger on the MC Unit if in Continuous Trigger Mode.
Trigger Button	When the Trigger Button is pressed, the scope will be started. If the scope is Manual Mode then the MC Unit immediately starts recording data. If it is in Program Trigger Mode then the MC Unit waits until it encounters a TRIGGER command in a running program. After the Trigger Button has been pressed, the text on the Button changes to 'Halt' while the scope is running. If the scope is in the One-shot Mode, then after the data has been recorded and plotted on the display, the Trigger Button text will return to 'Trigger', indicating that the operation has been completed. The scope can be halted at any time when it is running by pressing the trigger button (the 'Halt' text is displayed).
Reset Scope Configuration	The current scope configuration and all settings will be saved when the scope window is closed, and retrieved when the scope window is next opened. This removes the need to set each individual control again every time the scope window is opened. The Reset Scope Configuration Button can be pressed to reset the scope configuration, clearing all controls to their default values.
Status Indicator	The Status Indicator is located between the Options Button and the Reset Scope Configuration Button. This indicator changes color according to the current status of the scope as follows: Red Scope stopped. Black Waiting for MC Unit to complete recording data. Yellow Retrieving data from the MC Unit.

Channel-specific Controls

Each scope channel has the following channel-specific controls organized in each of four channel control blocks surrounded by a colored border. The color of the border is the same as the color for the channel trace on the display.

Each channel has the following:

Parameter Box	The parameters which the scope can record and display are selected using a pull-down list box in the upper left corner of each channel control block. Depending upon the parameter chosen, the next label will switch between axis or channel. It is also possible to plot the points held in the MC Unit Table array directly by selecting the Table parameter, followed by the number of a channel whose first/last points have been configured using the Advanced Options Window, which is described later in this section. If the scope channel is not required then 'NONE' should be selected in the parameter list box.
Axis/Channel List Box	The Axis/Channel List Box allows the user to select the required axis for a motion parameter, or channel for a digital input/output. The list box label will switch according to the setting in the Parameter List Box.
Vertical Scale	The scope vertical scale in units per grid division on the display can be set to either Automatic or Manual Mode. In Automatic Mode, the scope calculates the most appropriate scale when it has finished running and prior to displaying the trace. If the scope is running with continuous triggering, it will initially be unable to select a suitable vertical scale. When this happens, the scope must be halted and re-started, or used in the manual scaling mode. In Manual Mode, the user selects the scale per grid division. The vertical scale is changed by pressing the Up/Down Scale Buttons at the sides of the Current Scale Text Box. The button on the left decreases the scale value, and the button on the right increases the scale value. To return to the Automatic Mode, continue pressing the left button (decreasing the scale value) until the word 'AUTO' appears in the current scale text box.
Channel Trace Vertical Offset	The Vertical Offset Buttons are used to move a trace vertically on the display. This control is useful when two or more traces are identical, in which case they will overlay each other and only the uppermost trace will be seen on the display. The offset value will remain in effect for a channel until the Vertical Offset Reset Button is pressed or the scroll bar is used to return the trace to its original position.
Vertical Offset Reset	The vertical offset value applied using the vertical offset scroll bars can be cleared when the Vertical Offset Reset Button is pressed.
Cursor Button	After the scope has finished running and has displayed a trace, the cursor bars can be enabled. These are displayed as two vertical bars of the same color as the channel trace, and initially located at the maximum and minimum trace points. The values these represent are shown below the scope display, and the text is of the same color as the channel the values represent. The bars can be moved by positioning the mouse cursor over the required bar, holding down the left mouse button, and dragging the bar to the required position. The maximum or minimum value shown below the display is updated as the bar is dragged along with the value of the trace at the current bar position. The cursor bars are enabled/disabled by pressing the Cursor Button, which toggles alternately displaying and removing the cursor bars. When the cursor bars are disabled, the maximum and minimum points are indicated by a single white pixel on the trace.

Advanced Oscilloscope Configuration Options

When the Options Button of the General Options is pressed, the Advanced Oscilloscope Configuration Window will be displayed.

Oscilloscope: Samples per Division	The scope defaults to recording five points per horizontal time base grid division. This value can be adjusted using the adjacent scroll bar. To achieve the fastest scope sample rate it is necessary to reduce the number of samples per grid division to 1 and increase the time base scale to its fastest value of 1 ms per grid division. The trace might not be plotted completely to the right side of the display, depending upon the time base scale and number of samples per grid division.
Oscilloscope: Table Range	The MC Unit records the required parameter data values in the MC Unit's Table array prior to uploading these values to the scope. By default, the lowest scope Table value used is zero. If this conflicts with programs running on the MC Unit that require this section of the Table, then the lower scope table value can be set. The lower scope table value is adjusted by clicking into the text box and entering the new value. The upper scope table value will be automatically updated and cannot be changed by the user. The upper scope table value depends on the number of channels in use and the number of samples per grid division. If an attempt is made to enter a lower table value which causes the upper table value to exceed the maximum permitted value on the MC Unit, then the original value will be used by the scope.

It is possible to plot MC Unit Table ranges directly with the Oscilloscope. Select Table in the parameter box of the Axis Specific Controls to display the Table elements.

Table Graph: Points per division	The scope defaults to recording fifty points per horizontal time base grid division. This value can be adjusted using the adjacent scroll bar.
Table Graph: Table Range	The Table Limit Text Boxes are used to enter the Table ranges for the four possible channels of the Oscilloscope.

Parameter Checks

There is a maximum Table size on the MC Unit, and it is not possible to enter Table channel values beyond this value. It is also not possible to enter a lower scope table value or increase the samples per grid division to a value which causes the upper scope Table value to exceed the MC Unit maximum Table value.

If the number of samples per grid division is increased, and subsequently the time base scale is set to a faster value, causing an unobtainable resolution, the scope will automatically reset the number of samples per grid division.

Displaying MC Unit Table Points

If the scope is configured for both Table and motion parameters, then the number of points plotted across the display is determined by the time base and samples per division. If the number of points to be plotted for the table parameter is greater than the number of points for the motion parameter, the additional table points will not be displayed, but can be viewed by scrolling the table trace using the horizontal scroll bar. The motion parameter trace will not move.

Uploading Data from the MC Unit to the Scope

If the overall time base is greater than a predefined value, then the data is retrieved from the MC Unit in blocks, and the display can be seen to be updated in sections. The last point plotted in the current section will be displayed as a white spot.

If the scope is configured both to record motion parameters and to plot Table data, then the Table data is returned in one complete block, and the motion parameters are read either continuously or in block, depending upon the time base.

Even if the scope is in Continuous Trigger Mode, the Table data is not re-read; only the motion parameters are continuously read from the MC Unit.

Enabling/Disabling Scope Controls

While the scope is running, all the scope controls except the Trigger Button will be disabled. To change the time base or vertical scale, the scope must be stopped and restarted.

Display Accuracy

The MC Unit records the parameter values at the required sample rate in the Table, and then passes the information to the scope. The trace displayed is therefore accurate with respect to the selected time base. There is, however, a delay between when the data is recorded by the MC Unit and when it is displayed on the scope due to the time taken to upload the data via the serial link.

6-7 Suggestions and Precautions in Using Motion Perfect

Programming and Program Control

- Motion Perfect provides complete programming functions, such as edit, delete, rename, create, select and copy functions. When available, these should be used instead of the equivalent BASIC system commands in the Terminal Window. Motion Perfect cannot detect changes made by these BASIC system commands and a project check will be required to resolve inconsistencies.
- Use ***Reset the Controller*** on the MC Unit Menu to perform a software reset of the MC Unit.
- You do not need to close an Editor Window to run a program. It saves time not to. It is better to open an edit session for each program you want to see before running any programs. If there are programs already running, then it will not be possible to open an edit session.
- Do not turn the power ON and OFF or remove the serial connection when using Motion Perfect. If you do so, a communications error message will appear, and Motion Perfect will go off-line.
- You can force Motion Perfect to compare the computer project with the MC Unit programs at any time by selecting ***Check project*** from the File Menu.

Running Motion Perfect Off-line

Motion Perfect can be run in an off-line mode if it is unable to find a MC Unit and open a valid project. This may occur if it does not find any MC Units connected to the computer or if the project consistency check fails and the check is canceled.

In the offline mode, all project-related functions will be disabled. The user will only have access to

- Terminal Window (VT100 emulation).
- System software load.
- Communications setup.
- On-line help functions.

A Terminal Window can be opened and an attempt can be made to establish communications with the MC Unit. If the MC Units line mode >> prompt is returned when the Enter Key is pressed from the keyboard at the computer, then the MC Unit can be communicated with using the BASIC system commands (see the BASIC on-line help for further information). The commands given in *5-2-4 Program Commands and Functions* can be used to manipulate programs using a terminal.

Project Backups

If the MC Unit stops responding during a development session and the same project is reconnected, then it is likely the consistency check will be passed.

In the case that the programs are not consistent, the Check Project Options Window will be displayed. Please be aware that if the current project is re-

opened, the backup copy of the project will be overwritten. It is therefore necessary to determine which copy of the programs to use before re-connecting Motion Perfect.

To investigate the inconsistency further, a Terminal Window can be opened off-line and the programs on the MC Unit can be listed. Any computer-based editor or word processor can be used to examine the computer backup project file copy of the programs. In this way, the location of the uncorrupted or latest version of the programs can be identified. The correct program can be imported to the project by using the Load Program File option of the Project Menu.

The computer project copy of a program is updated during a Motion Perfect development session whenever an edit session for the program is closed.

Retrieving Backup

If you want to abandon changes made during a development session and reload the backup copy made at the start of the session, then select ***Revert to Backup*** from the Project Menu.

Downloading Firmware

The Motion Controller series feature a flash-EPROM for storage of both user programs and the system software. From Motion Perfect it is possible to upgrade the software to a newer version using a system file.

Select the 'Load System Software' option from the controller menu and a warning dialog will be presented to ensure the current project has been saved the user wishes to continue. Press OK and select the file which needs to be loaded.

⚠WARNING Each Motion Controller has its own system file, identified by the first letter of the file name. Please ensure to only load software designed for the specific controller.

Downloading will take several minutes, depending on the speed of the personal computer. When the download is complete, a checksum is performed to ensure that the download process was successful, and a confirmation screen will be presented to store the software into EPROM. The controller will take a few moments to store the software into the EPROM.

⚠WARNING During the process of storing the software into EPROM the power may NEVER be interrupted. If power is interrupted the MC Unit may disfunction and has to be returned to OMRON for re-initialisation.

If the storing has been completed the unit is back to normal operation. At this point you can check the controller configuration to confirm the new software version.

SECTION 7
Troubleshooting

This section provides procedures on troubleshooting problems that may arise with the MC Unit.

7-1 Problems and Countermeasures 180
7-2 Error Indicators 184
7-3 Error Handling 184
7-4 CPU Unit Error Flags and Control Bits 187

7-1 Problems and Countermeasures

The following table shows possible problems which may occur and the possible solution.

No.	Problem	Probable causes	Items to check	Remedy
1	None of the CPU Unit's indicators are lit when the power is turned ON.	Power supply lines are wired incorrectly.	Check the power supply wiring.	Correct the power supply wiring.
2		The power supply voltage is low.	Check the power supply voltage.	Check the power supply capacity and correct the power supply.
3		An internal fuse has blown.	Check the fuses.	Replace the fuse and determine what caused it to blow. (Refer to the troubleshooting section in the applicable CPU Unit operation manual).
4		The power supply is defective.	Check the power supply.	Replace the power supply.
5	None of the MC Unit's indicators are lit.	The power supply capacity is insufficient.	Add up the power supply capacity for all of the Units mounted to the same Backplane, including the CPU Unit, and compare that to the power supply capacity of the Power Supply Unit. If the combined capacity of the Units is greater than that of the Power Supply Unit, then they cannot be properly used.	Increase the power supply capacity. Change the configuration so that the power supply capacity at the Backplane is not exceeded.
6		The MC Unit is defective.	---	Replace the MC Unit.
7	Motion Perfect cannot connect to the MC Unit.	Serial communications cable is not connected properly.	Check the cable connection and wiring.	Correct the cable wiring. Replace cable if necessary.
8		Motion Perfect serial communications settings are different from the MC Unit settings.	Check Motion Perfect settings.	Correct the settings or set the settings to default and reset the MC Unit. Default settings of the Unit is 9600 baud, even parity and two stop bits.
9		A program is running which is printing to the port and interfering Motion Perfect's protocol.	Check the MC Unit by using a VT100 Terminal.	Stop the program (verify if it is safe to do so) by giving command HALT or STOP.
10		The MC Unit is defective	---	Replace the MC Unit.
11	Driver cannot be enabled.	The MC Unit is not operating.	Is the RUN indicator lit?	Check No. 5
12		The wiring is incorrect between the MC Unit and the Servo Driver.	Check the wiring with a tester. Change the connecting cables.	Correct the wiring.
13		A Servo Driver alarm has been generated.	Check the contents of the Servo Driver alarm.	If there is an alarm, follow the instructions.
14		The MC Unit is defective.	---	Replace the MC Unit.

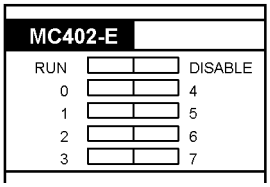
No.	Problem	Probable causes	Items to check	Remedy
15	Motor is not turning.	The Servo Driver is not enabled.	Check the MC Unit to see whether the Driver is enabled and the servo loop active (WDOG and SERVO parameters). Check whether the Servo Driver is operating.	Correct the MC Unit settings. Correct the Servo Driver operation.
16		The wiring is incorrect between the MC Unit and the Servo Driver.	Check the wiring with a tester. Change the connecting cables.	Correct the wiring.
17		A run prohibit input (limit switch), such is P-OT or N-OT, is ON for a U-series Servo Driver.	Check the run prohibit inputs.	Turn OFF the Servo Driver run prohibit input. Make the setting so that the Servo Driver run prohibit inputs will not be used.
18		The Servo Driver is not in the correct (speed control) mode (and is not receiving MC Unit speed reference).	Check the Servo Driver settings.	Correct the Servo Driver settings.
19		The mechanical axis is locked.	Check whether there is a mechanical limit or lock in effect.	Manually release the mechanical lock.
20		The MC Unit is defective.	---	Replace the MC Unit.
21	Rotation is reversed.	The Servo Driver is set for reverse rotation.	Check whether the Servo Driver is set for reverse rotation by jogging.	Correct the setting for the direction of Servo Driver rotation.
22		The feedback signal (phase A/B) is reversed.	Check whether the feedback signal (phase A/B) is reversed.	Correct the wiring.
23		The PP_STEP parameter setting is set for reverse rotation.	Check whether the parameter is set for reverse rotation (is negative).	Correct the parameter value.
24	There are unusual noises.	The machinery is vibrating.	Check for foreign objects in the machinery's moving parts, and inspect for damage, deformation, and looseness.	Make any necessary repairs.
25		The speed loop gain is insufficient. (The gain is too high.)	---	Perform autotuning. Manually adjust (decrease) the gain.
26		The wrong Servomotor is selected (so it cannot be adjusted).	Check the torque and inertia ratings and select another Servomotor.	Change to a suitable Servomotor.
27		There is eccentricity in the couplings connecting the Servomotor axis and the mechanical system.	---	Adjust the mounting of the Servomotor and machinery.

No.	Problem	Probable causes	Items to check	Remedy
28	Motor rotation is unstable.	The parameters are set incorrectly.	Check the MC Unit parameters with Motion Perfect.	Set the parameters correctly and modify the initialisation program accordingly.
29		The Servo Motor power lines and encoder lines are wired incorrectly.	Check the Servo Motor power lines and encoder lines.	Correct the wiring.
30		The speed reference (VREF_0, VREF_1, VREF_2, VREF_3) polarity is wrong.	Check the speed reference wiring.	Correct the wiring.
31		There is eccentricity in the couplings connecting the Servomotor axis and the mechanical system. There may be loose screws or load torque fluctuation due to the meshing of pulley gears.	Check the machinery. Try turning the motor with no load (i.e., with the machinery removed from the coupling).	Adjust the machinery.
32		The gain adjustment is insufficient.	---	Execute Servomotor autotuning. Manually adjust the Servomotor gain. Adjust the servo control parameters with Motion Perfect.
33		The wrong Servomotor is selected (so it cannot be adjusted).	Check the torque and inertia ratings and select another Servomotor.	Change to a suitable Servomotor.
34		The Servomotor bearings are damaged.	Turn OFF the Servo Driver power. If the Servomotor has a brake, turn ON the brake power supply and release the brake, and then manually turn the motor's output axis with the motor's power line disconnected (because the dynamic brake may be applied).	Replace the Servomotor.
35		The Servomotor windings are disconnected.	With a tester, check resistance between the Servomotor's U, V and W power lines. There should be a proper balance between the line resistances.	Replace the Servomotor.
36	Vibration is occurring at the same frequency as the application frequency.	Inductive noise is being generated.	Check whether the Servo Driver control signals are too long. Check whether the control signal lines and power lines are bundled together.	Shorten the control signals. Separate the control signal lines and the power lines. Use a low-impedance power supply for the control signal lines.
37		The control signals are not properly grounded.	Check whether the control signal shield is properly grounded at the Servo Driver. Check whether the control signal lines are in contact with ground.	Correct the wiring.
38		Twisted-pair or shielded cable is not being used between the MC Unit and the Servo Driver.	Check whether twisted-pair cables are used for the encoder signals and speed references, and whether the cables are shielded.	Use twisted-pair and shielded cable as in the wiring examples.

No.	Problem	Probable causes	Items to check	Remedy
39	The motor axis is vibrating unsteadily.	The gain adjustment is insufficient. (The gain is too low.)	---	Perform autotuning. Manually adjust (increase) the gain.
40		The gain cannot be adjusted because the mechanical rigidity is too weak.	This particularly tends to occur in systems with vertical axes, scalar robots, palletisers, and so on, which place a torsion load on the axes.	Increase the mechanical rgidity. Re-adjust the gain.
41		The mechanical structure is producing stick slip (high viscosity statical friction).	---	Perform autotuning. Manually adjust the gain.
42		The wrong Servomotor is selected (so it cannot be adjusted).	Check the torque and inertia ratings and select another Servomotor.	Change to a suitable Servomotor.
43		The Servomotor or Servo Driver is defective.	---	Replace the Servomotor or the Servo Driver.
44	There is slippage in positioning.	The slippage is not constant. Malfunction due to noise.	Is shielded cable being used?	Use shielded cable.
45		The shield is not properly grounded at the Servo Driver.	Check the ground wiring.	Correct the wiring.
46		The MC Unit's output power supply is not separated from other power supplies.	Check whether the MC Unit's output power supply is separated from other power supplies.	Separate the MC output supply from other power supplies.
47				Install a noise filter at the primary side of the MC Unit's output power supply.
48				Ground the MC Unit's output power supply
49		The cable between the MC Unit and the Servomotor is too long.	Check whether the cable is two meters or less.	The maximum cable length is two meters.
50		Twisted-pair cable is not being used for the pulse outputs.	Check whether twisted-pair cable is being used for the pulse outputs. (The connected voltage is 0 V or 5/24 VDC.)	Use twisted-pair cable for pulse outputs.
51		The cable between the MC Unit and the Servo Driver is not separated from other power lines.	Check whether the cable is separated from other power lines.	Separate the cable from other power lines.
52		The cable between the MC Unit and Servo Driver is too long.	Check whether the cable is two meters or less.	The maximum cable length is two meters.
53		There is malfunctioning due to noise from a welding machine, inverter, etc.	Check whether there is a device such as a welding machine or inverter nearby.	Separate the Unit from the noise source.
54		There is slippage in the mechanical system.	Check for slippage by marking the mechanical connections.	Tighten the connections.

7-2 Error Indicators

The following errors are displayed at the LED indicators at the top of the MC Unit's front panel.



RUN	DISABLE	Error	Remedy
ON	---	(Normal)	---
OFF	OFF	The MC Unit is defective.	Replace the MC Unit.
Flashing	---	The battery voltage is low.	Replace the battery.
---	Flashing	The following error has exceeded the limit. The Servo Drives have been disabled.	Check what caused the error, correct the problem and restart application.
Flashing	Flashing	One of the following errors occurred in the configuration of MC Unit and CPU Unit: Illegal unit number Unit number duplication Cyclic initial error Unit number setting error	Set the correct unit number and turn ON the power again.

7-3 Error Handling

Motion error

The motion coordinator is designed to trap error conditions in hardware, and if required to automatically disable the drive and outputs to the drive. As this happens automatically, it may not be immediately apparent that an error has occurred and therefore there are some software flags which identify the situation.

A motion error occurs when the AXISSTATUS state for one of the axes matches the ERRORMASK parameter setting. In this case the enable relay (WDOG) will be turned OFF, the MOTION_ERROR parameter will have value 1 and the ERROR_AXIS parameter will contain the number of the first axis to have the error. As a result of the WDOG turning OFF, the speed reference signals of the axes will be set to zero. The relevant parameters are again given here.

Parameter	Description
WDOG	WDOG controls the enable relay which enables the drivers. In case of a motion error this enable relay will be disabled.
AXISSTATUS	AXISSTATUS contains the status bits of an axis.
ERRORMASK	ERRORMASK is bit wise ANDed with the AXISSTATUS to determine whether a motion error has occurred and the driver enable should be set OFF. The default value of ERRORMASK is 256 (i.e. bit 8 set) which implies a motion error occurs if following error exceeds limits.
MOTION_ERROR	The MOTION_ERROR parameter will be ON when a motion error has occurred.
ERROR_AXIS	The ERROR_AXIS parameter contains the number of the axis for which a motion error has occurred.

Run-time BASIC Errors

Run-time BASIC errors will stop the program or will go into the error routine as defined by BASICERROR. The following parameters are relevant when checking a run-time error.

Parameter	Description
BASICERROR	The BASICERROR command traps the error and allows the control of the program to go to an error handling routine
ERROR_LINE	The ERROR_LINE parameter which shows which line in the program has encountered the error.
RUN_ERROR	The RUN_ERROR shows the identity number of the actual error.

The table below shows a list of the different types of BASIC run-time errors which are detected.

Error No.	Message Displayed
1	Command not recognized
2	Invalid transfer type
3	Error programming Flash
4	Operand expected
5	Assignment expected
6	Quotes expected
7	Stack overflow
8	Too many named variables
9	Divide by zero
10	Extra characters at end of line
11	] expected in PRINT
12	Cannot modify a special program
13	THEN expected in IF
14	Error erasing Flash
15	Start of expression expected
16	) expected
17	, expected
18	Command line broken by ESC
19	Parameter out of range
20	No process available
21	Parameter is read only
22	Modifier not allowed
23	DriveLink axis is in use
24	Command is command line only
25	Command runtime only
26	LABEL expected
27	Program not found
28	Duplicate label
29	Program is locked
30	Program(s) running
31	Program stopped
32	Cannot select program
33	No program selected
34	No more programs available
35	Out of memory
36	No code available to run
37	Command out of context

Error No.	Message Displayed
38	Too many nested structures
39	Structure nesting error
40	ELSE/ENDIF without previous IF
41	WEND without previous WHILE
42	UNTIL without previous REPEAT
43	Variable expected
44	TO expected if FOR
45	Too many nested FOR/NEXT
46	NEXT without FOR
47	UNTIL/IDLE expected after WAIT
48	GOTO/GOSUB expected
49	Too many nested GOSUB
50	RETURN without GOSUB
51	LABEL must be at start of line
52	Cannot nest one line IF commands
53	LABEL not found
54	Line number cannot have decimal point
55	Cannot have multiple instances of REMOTE
56	Invalid use of $
57	VR(x) expected
58	Program already exists
59	Process already selected
60	Duplicate axes not permitted
61	PLC type is invalid
62	Evaluation error
63	Reserved keyword not available on this controller
64	Label not found
65	Table index range error
66	Table is full
67	Invalid line number
68	String exceeds permitted length
69	Scope period should exceed number of Ain params
70	Value is incorrect
71	Invalid I/O channel
72	Value cannot be set.
73	Directory not locked
74	Directory already locked
75	Program not running on this process
76	Program not running
77	Program not paused on this process
78	Program not paused
79	Command not allowed when running Motion Perfect
80	Directory structure invalid
81	Directory is locked
82	Cannot edit program

PC Transfer Error Flag

The PC Transfer Error Flag in the PC's IR/CIO area will be set for a IORD/IOWR instruction in the following cases:

- The control code of the IORD/IOWR instruction is not valid for the MC Unit.
- The amount of words transferred is not a multiple of three for three-word format transfer.
- The MC Unit's Table or VR address in combination with the amount of data is invalid.
- There is an overflow due to several IORD/IOWRs instructions and BASIC commands PLC_WRITE/PLC_READ.

7-4 CPU Unit Error Flags and Control Bits

The PC error flags in the following tables will indicate the following errors.

- Duplicate unit numbers on Special I/O Units
- Refreshing between the CPU Unit and Special I/O Unit did not proceed normally.

To restart a Special I/O Unit, toggle the corresponding Restart Bit shown in the tables. These bits can be used to restart the Unit without turning off the power supply.

C200HX/HG/HE PCs

Address	Function
SR 25415	ON when an error occurs in a Special I/O Unit.
SR 282 bit i	ON when an error occurs in Unit i
SR 281 bit i	Restarts Unit i

C200HS PCs

Address	Function
SR 25415	ON when an error occurs in a Special I/O Unit.
AR 00 bit i	ON when an error occurs in Unit i
AR 01 bit i	Restarts Unit i

CS1 PCs

Address	Function
A40202	ON when an installed Unit does not match the Unit registered in the I/O table.
A42800 to A43315	Specifies the corresponding Unit number 0 to 95 when the Unit does not match the Unit registered in the I/O table.
A40206	ON when an error occurs in the data exchange with a Unit.
A41800 to A42315	Specifies the corresponding Unit number 0 to 95 when an error occurs in the data exchange with the Unit.
A50200 to A50715	Restarts Units 0 to 95.

SECTION 8
Maintenance and Inspection

This section explains the maintenance and inspection procedures that must be followed to keep the MC Unit operating in optimum condition. It also includes proper procedures when replacing an MC Unit or battery.

8-1 Routine Inspections 190
8-2 Handling Precautions 191
8-2-1 Procedure for Replacing an MC Unit 191
8-2-2 Procedure for Replacing a Battery 192

8-1 Routine Inspections

In order for your MC Unit to continue operating at optimum condition, periodic inspections are necessary. The main components of the Unit are semiconductors and have a long service life, but depending on the operating environment, there may be more or less deterioration of these and other parts. A standard inspection schedule is once every six months to one year. More frequent inspections may be advisable depending on the operating environment. Maintain the inspection schedule once it has been set.

Inspection Points

Check to be sure that the power supply, ambient temperature, humidity, and other specifications are within the specifications. Be sure that there are no loose screws and that all battery and cable connections are secure. Clean any dust or dirt that has accumulated.

Item	Inspection points	Criteria	Remarks
I/O Power Supply	Measure the voltage variations at the I/O power supply terminal block. Do they meet the standards?	24 VDC: 20.4 to 26.4 VDC	With a voltage tester, check between the terminals and make sure that the power supply falls within the acceptable range.
Installation and wiring	Is the MC Unit securely mounted?	There must be looseness.	With a Phillips screwdriver, tighten all mounting screws.
	Are the cable connectors properly inserted and locked?		Carefully insert and lock all cable connectors.
	Are there any loose screws in the external wiring?		With a Phillips screwdriver, tighten all screws in the external wiring.
	Are any crimp terminals for external wiring too close together?	There must be sufficient distance between them.	Do a visual check and separate the terminals as required.
	Are any external cables disconnected?	There must be no external abnormalities.	Do a visual check and connect or replace cables as required.
Environment conditions	Is the ambient temperature within the acceptable range? (When used in a panel, the ambient temperature inside the panel must be checked.)	0 to 55°C	With a thermometer, check the ambient temperature inside the panel and make sure that it falls within the acceptable range.
	Is the ambient humidity within the acceptable range? (When used in a panel, the ambient temperature inside the panel must be checked.)	10% to 90% RH (with no condensation)	With a hydroscope, check the ambient humidity inside the panel and make sure that it falls within the acceptable range.
	Is the Unit exposed to direct sunlight?	It must not be exposed to direct sunlight.	Shield the Unit from direct sunlight.
	Is there any accumulation of dust (especially iron dust) or salts?	There must be none of these present.	Remove any accumulation of dust or salts and protect against them.
	Is the Unit exposed to any spray of water, oil, or chemicals?	It must not be exposed to any of these.	Protect the Unit from water, oil, and chemicals.
	Is the location subject to corrosive or flammable gases?	The Unit must not be exposed to these.	Check for smells or use a gas sensor.
	Is the location subject to shock or vibration?	The amount of shock or vibration must be within the acceptable ranges given in the specifications.	Install a cushion or other device to reduce shock and vibration.
	Is the location near any source of noise?	There must be no noise.	Remove the Unit from the noise source or apply countermeasures.

Item	Inspection points	Criteria	Remarks
Installation and wiring	Is the MC Unit securely mounted?	There must be looseness.	With a Phillips screwdriver, tighten all mounting screws.
	Are the cable connectors properly inserted and locked?		Carefully insert and lock all cable connectors.
	Are there any loose screws in the external wiring?		With a Phillips screwdriver, tighten all screws in the external wiring.
	Are any crimp terminals for external wiring too close together?	There must be sufficient distance between them.	Do a visual check and separate the terminals as required.
	Are any external cables disconnected?	There must be no external abnormalities.	Do a visual check and connect or replace cables as required.

Required Tools

The following tools, materials, and equipment are required when performing an inspection.

- Phillips screwdriver
- Voltage tester or digital voltage meter
- Industrial alcohol and a clean cotton cloth
- Synchroscope
- Oscilloscope
- Thermometer
- Hydrometer

8-2 Handling Precautions

- Turn OFF the power before replacing the Unit.
- If a Unit is found to be faulty and is replaced, check the new Unit again to ensure there are no errors.
- When returning a faulty Unit for repair, make a detailed record of the Unit's malfunction and take it together with the Unit to your nearest OMRON office or sales representative.
- If a contact is not good, put some industrial alcohol on a clean cotton cloth and wipe the surface. After doing this, install the Unit.
- Before restarting operation, transfer the required programs and (position) data to the MC Unit that was changed, and save them to the flash memory.

8-2-1 Procedure for Replacing an MC Unit

Use the following procedure when it is necessary to replace an MC Unit.

1,2,3...

1. Make a note of the unit number of the MC Unit to be replaced.
2. To retain the status of the MC Unit that is to be replaced, use Motion Perfect to check the project of the Unit and to have a local copy saved on the personal computer.
3. Turn OFF the power supply.
4. Replace the MC Unit, and reconnect the wiring as before.
5. Set the unit number for the MC Unit.
6. Turn ON the power supply to the PC.
7. Clear all the programs in the MC Unit.
8. Download all of the programs and data to the MC Unit, and save the programs in flash memory.

8-2-2 Procedure for Replacing a Battery

Battery Type

Type: Sonnenschein Lithium 1/2 AA

Model: SL-350/S

The battery in the MC Unit has a life expectancy of 5 years at an ambient temperature of 25°C, whether or not the MC Unit is connected to electricity. The life time will be shorter at higher ambient temperatures.

Replacing Battery

When the battery voltage begins to fail, the RUN indicator will start flashing, the BATTERY_LOW parameter will be set to TRUE and the Low Battery Flag bit the IR/CIO area will be turned ON. Replace the battery as outlined below within 1 week.

1,2,3...

1. Be sure that the programs have been saved either in flash memory or on disk.
2. Turn OFF the power.
3. Remove the two retaining screws at the back of the MC Unit.
4. Remove the back cover of the MC Unit.
5. Slide the controller boards out of its case.
6. Undo the screws at the pillars holding the two boards together.
7. Gently separate the two boards from each other. Do not damage the interboard connector.
8. Remove the old battery and replace it with the new one. This operation must be completed within 5 minutes.
9. Put the two boards back together again. Again, do not damage the interboard connector.
10. Tighten the screws at the retaining pillars.
11. Gently slide the controller boards back into the sleeve until they are properly home.
12. Replace the back cover of the MC Unit.
13. Replace and tighten the retaining screws.

Appendix A
PC Interface Area Lists

PC Interface Area Outputs (CPU Unit to MC Unit)

Output word	Bit	Name	Function
n	00 to 15	Output Word 1	First transfer output word.
n + 1	00 to 15	Output Word 2	Second transfer output word.

PC Interface Area Inputs (MC Unit to CPU Unit)

Input word	Bit	Name	Function
n + 2	00	Unit Operating Flag	OFF: MC Unit is not operating. ON: MC Unit is operating.
	01	Servo Error Flag	OFF: No error. ON: A motion error has occurred.
	02	Task 1 Flag	OFF: Task1 is inactive. ON: Task 1 is active.
	03	Task 2 Flag	OFF: Task 2 is inactive. ON: Task 2 is active.
	04	Task 3 Flag	OFF: Task 3 is inactive. ON: Task 3 is active.
	05	Task 4 Flag	OFF: Task 4 is inactive. ON: Task 4 is active.
	06	Task 5 Flag	OFF: Task 5 is inactive. ON: Task 5 is active.
	07	PC Transfer Busy Flag	ON: The MC Unit is exchanging data with the PC Unit.
	08 to 15	Digital Input Status Flags	Indicate the status of digital inputs 0 to 7.
n + 3	00 to 15	Digital Input Status Flags	Indicate the status of the digital inputs 8 to 23.
n + 4	00 to 15	Digital Output Status Flags	Indicate the status of digital outputs 8 to 23.
n + 5	00	Axis 0 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 0.
	01	Axis 0 Forward Limit Flag	ON: A forward limit is set for axis 0.
	02	Axis 0 Reverse Limit Flag	ON: A reverse limit is set for axis 0.
	03	Axis 0 Origin Search Flag	ON: An origin search is in progress for axis 0.
	04	Axis 0 Feedhold Flag	ON: A feedhold is set for axis 0.
	05	Axis 0 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 0.
	06	Axis 0 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 0.
	07	Axis 0 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 0.
	08	Axis 1 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 1.
	09	Axis 1 Forward Limit Flag	ON: A forward limit is set for axis 1.
	10	Axis 1 Reverse Limit Flag	ON: A reverse limit is set for axis 1.
	11	Axis 1 Origin Search Flag	ON: An origin search is in progress for axis 1.
	12	Axis 1 Feedhold Flag	ON: A feedhold is set for axis 1.
	13	Axis 1 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 1.
	14	Axis 1 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 1.
	15	Axis 1 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 1.

Input word	Bit	Name	Function
n + 6	00	Axis 2 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 2.
	01	Axis 2 Forward Limit Flag	ON: A forward limit is set for axis 2.
	02	Axis 2 Reverse Limit Flag	ON: A reverse limit is set for axis 2.
	03	Axis 2 Origin Search Flag	ON: An origin search is in progress for axis 2.
	04	Axis 2 Feedhold Flag	ON: A feedhold is set for axis 2.
	05	Axis 2 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 2.
	06	Axis 2 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 2.
	07	Axis 2 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 2.
	08	Axis 3 Following Error Warning Limit Flag	ON: The warning limit was exceeded for the following error for axis 3.
	09	Axis 3 Forward Limit Flag	ON: A forward limit is set for axis 3.
	10	Axis 3 Reverse Limit Flag	ON: A reverse limit is set for axis 3.
	11	Axis 3 Origin Search Flag	ON: An origin search is in progress for axis 3.
	12	Axis 3 Feedhold Flag	ON: A feedhold is set for axis 3.
	13	Axis 3 Following Error Limit Flag	ON: The limit was exceeded for the following error for axis 3.
	14	Axis 3 Software Forward Limit Flag	ON: The software forward limit was exceeded for axis 3.
	15	Axis 3 Software Reverse Limit Flag	ON: The software reverse limit was exceeded for axis 3.
n + 7	00	Task 1 BASIC Error Flag	ON: An error occurred in the BASIC program in task 1.
	01	Task 2 BASIC Error Flag	ON: An error occurred in the BASIC program in task 2.
	02	Task 3 BASIC Error Flag	ON: An error occurred in the BASIC program in task 3.
	03	Task 4 BASIC Error Flag	ON: An error occurred in the BASIC program in task 4.
	04	Task 5 BASIC Error Flag	ON: An error occurred in the BASIC program in task 5.
	05	Low Battery Flag	ON: The voltage of the backup battery is low.
	06	Not used	---
	07	PC Transfer Error Flag	ON: An error has occurred during data transfer between MC Unit and PC Unit.
	08 to 15	Indicator Mode	Contains the value of the DISPLAY system parameter, which determines the display mode of the bank of the 8 LED indicators on the front panel. Refer to *5-3-53 DISPLAY* for details.
n + 8	00 to 15	Input Word 1	First transfer input word.
n + 9	00 to 15	Input Word 2	Second transfer input word.

Appendix B
Programming Examples

Example 1: Turning an Output ON and OFF Every 100ms

The following code controls output 10 to go on and off every 100 ms. The DISPLAY parameter is included to show the blinking on the MC Unit's LED indicators.

```
   DISPLAY = 5
loop:
   OP(10, ON)
   WA(100)
   OP(10, OFF)
   WA(100)
   GOTO loop
```

Example 2: Flashing MC Unit Outputs

The following code will sequentially step through all available outputs on the MC Unit and flash them for 0.5 seconds each

```
start:
   FOR a = 8 TO (NIO-1)
      OP(a, ON)
      WA(500)
      OP(a, OFF)
   NEXT a
   GOTO start
```

Example 3: Turning ON Outputs

The following code will sequentially step through all available outputs on the MC Unit and turn them ON only when input 0 is ON.

```
start:
   FOR a = 8 TO 15
      WAIT UNTIL IN(0) = ON
      OP(a, ON)
      WAIT UNTIL IN(0) = OFF
      OP(a, OFF)
   NEXT a
   GOTO start
```

Example 4: Positioning a Rotary Table

A rotary table must stop at one of 8 equally spaced positions according to the value of a thumbwheel input (inputs 4 to 7). The table will not move until a start button is pressed (input 15).

```
start:
   WAIT UNTIL IN(15) = ON
   WAIT UNTIL IN(15) = OFF
   GOSUB get_tws

   MOVEABS(45 * tw_value)
   WAIT IDLE
   GOTO start

get_tws:
   tw_value = IN(4,7)
   RETURN
```

Example 5: Positioning with Product Detection

A ballscrew is required to move forward at a creep speed until it reaches a product, at which point a microswitch (IN(2)) is turned ON. The ballscrew is stopped immediately, the position at which the product was sensed is indicated and the ballscrew is returned at a rapid speed back to the start position.

```
start:
   IF ( IN(1) = ON ) THEN WAIT UNTIL IN(8) = OFF
   WAIT UNTIL IN(1) = ON
   SPEED = 10

   FORWARD
   WAIT UNTIL IN(2) = ON

   prod_pos = MPOS
   CANCEL
   WAIT IDLE

   PRINT "Product Position : "; prod_pos
   SPEED = 100
   MOVEABS(0)
   WAIT IDLE
   GOTO start
```

Example 6: Positioning on a Grid

A square palette has sides 1m long. It is divided into a 5 x 5 grid, and each of the positions on the grid contains a box which must be filled using the same square pattern of 100mm by 100mm. A dispensing nozzle controlled by digital output 8 must be turned ON when filling the box and OFF at all other times.

```
   nozzle = 8

start:
   FOR x = 0 TO 4
      FOR y = 0 TO 4
         MOVEABS(x*200, y*200)
         WAIT IDLE
         OP(nozzle, ON)
         GOSUB square_rel
         OP(nozzle, OFF)
      NEXT y
   NEXT x
   GOTO start

square_rel:
   MOVE(0, 100)
   MOVE(100, 0)
   MOVE(0, -100)
   MOVE(-100,0)
   WAIT IDLE
   WA(1000)
   RETURN
```

Example 7: Synchronising Cutter Movement

A flying shear cutter is required to synchronise with a continuously moving web and to cut a roll of paper every 5m:

- The cutter (axis 1) can move a total of 600mm. We use a maximum 500mm of this travel.
- The blade is operated by a solenoid which is switched by digital output 8.
- The blade must be operated mid-way through the cutter motion.
- The cutter must synchronise to cut, and return to its start position all within not more than 80% of the repeat cycle.

To ensure that speeds and positions of the cutter and paper match during the cut process, the arguments of the MOVELINK command must be correct. It is normally easiest to consider the acceleration, constant speed and deceleration phases separately and then combine them as required.

```
start:
   UNITS AXIS(0) = 5000 'Meters
   UNITS AXIS(1) = 5000
   BASE(0)
   FORWARD

loop:
   BASE(1)
   MOVELINK(0, 4, 0, 0, 0)           'Wait distance

   MOVELINK(0.1, 0.2, 0.2, 0, 0)     'Accelerate
   MOVELINK(0.3, 0.3, 0, 0, 0)       'Match speed
   MOVELINK(0.1, 0.2, 0, 0.2, 0)     'Decelerate

   MOVELINK(-0.5, 5, 3, 3, 0)        'Move back
   GOTO loop
```

The middle MOVELINK commands can be done in one move using the following line.

```
   MOVELINK(0.5, 0.7, 0.2, 0.2, 0)
```

Example 8: Generating Smooth High-speed Profiles

It is often desirable to generate a smooth profiled move for the maximum operational speed in high-speed machines. An optimal profile for this is a sine squared:

$$y = mx - n(\sin(x))$$

In this example we work in radians.

```
start:
   GOSUB filltable
   BASE(0)
   SPEED = 200
   FORWARD
   BASE(1)

loop:
   CAMBOX(0,36,1,100,0)
   WAIT IDLE
   WA(1000)
   GOTO loop

filltable:
   num_p = 37
   scale = 2000
   FOR p=0 TO num_p
   TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
   NEXT p
   RETURN
```

Example 9: Coordinating Two Moving Objects

Two conveyors run in parallel, conveyor A (axis 1) carries a product that must be transferred into boxes evenly spaced on conveyor B (axis 0). The transfer operation requires the products to be aligned at the end of the conveyor.

A registration process checks the position of the product on the conveyor and calculates the amount that conveyor A must be advanced or retarded in order to align with conveyor B. Input 1 indicates that the registration process has been completed and the correction amount loaded serially into VR(1).

```
setup:
   BASE(1)
```

```
   CONNECT(1,0)
   ADDAX(2)
   BASE(2)

loop:
   IF IN(1) = ON THEN WAIT UNTIL IN(1) = OFF
   WAIT UNTIL IN(1) = ON
   correction = VR(1)
   MOVE(correction)
   WAIT IDLE
   GOSUB do_transfer
   GOTO loop

do_transfer:
   OP(15,ON)
   WA(500)
   OP(15,OFF)
   RETURN
```

Example 10: Coordinating Motion with Mark Detection

A cyclic cut-to-length operation requires a rolled product to be cut in relation to a printed mark.

The product is nominally 150mm long and the printed registration mark appears 30mm from the end of the product. The product must be stationary when cut, but the draw motion should be one continuous move.

A high-speed optical sensor is connected to the registration input of the feed axis.

```
loop:
   REGIST(3)
   DEFPOS(0)
   MOVE(150)
   WAIT UNTIL MARK
   MOVEMODIFY(REG_POS+30)
   WAIT IDLE
   GOSUB cut_operation
   GOTO loop

cut_operation:
   'Omitted from this example.
   RETURN
```

Example 11: Pick and Place Machine

A pick-and-place machine is filling boxes on a palette. When the palette is full, a transfer carousel will replace the palette with a fresh one and a secondary operation will spiral-wrap the whole palette with plastic film.

We can assume the palletising operation is similar to example 6.

The wrapping operation takes approximately half the time taken to fill the palette and involves the following operations:

- Wait for "finished" signal from pick-and-place unit
- Rotate the carousel 180°
- Signal pick-and-place unit "OK to continue"
- Engage the wrapper drive (digital output)
- Rotate wrapper 4 times
- Disengage the wrapper drive
- Signal operator to remove palette
- Wait until operator presses the continue button

```
   GOSUB constants

start:
   FOR x = 0 TO 4
      FOR y = 0 TO 4
         MOVEABS(x*200, y*200)
         WAIT IDLE
         OP(nozzle, ON)
         GOSUB square_rel
         OP(nozzle, OFF)
      NEXT y
   NEXT x

   VR(fill_complete) = 1
   WAIT UNTIL VR(carousel_ok) = 1
   VR(fill_complete) = 0
   GOTO start

constants:
   ax_X = 0
   ax_Y = 1
   ax_carousel = 2
   ax_wrapdrive = 3

   but_con = 1
   engage_wrap = 8
   lamp = 9
   nozzle = 10
   fill_complete = 10
   carousel_ok = 11

   RETURN
```

```
   GOSUB constants

loop:
   WAIT UNTIL VR(fill_complete) <> 0
   VR(carousel_ok) = 0
   MOVE(180) AXIS(ax_carousel)
   WAIT IDLE
   VR(carousel_ok) = 1
   OP(engage_wrap, ON)
   WA(1000)
   MOVE(4) AXIS(ax_wrapdrive)
   WAIT IDLE AXIS(ax_wrapdrive)
   OP(engage_wrap, OFF)
   OP(lamp_OK)
   WAIT UNTIL IN(but_con) = ON
   OP(lamp, OFF)
   GOTO loop

constants:
   ax_X = 0
   ax_Y = 1
   ax_carousel = 2
   ax_wrapdrive = 3

   but_con = 1
   engage_wrap = 8
   lamp = 9
   nozzle = 10
   fill_complete = 10
   carousel_ok = 11

   RETURN
```

Example 12: Master Program

This master shell program is designed to master most applications. The program controls the executions of the application programs and continuously monitors the status of the system. Please be careful to use it in the application and check that it is functioning correctly in all cases before relying on its safety operation. This program should be set to run at power up at low priority (tasks 1, 2 or 3).

Note This master shell program does not include any hand-shaking for communications with the PC Unit. If the application is using dynamic data from the PC memory, be sure to transfer the data to the MC Unit during the initialization phase of the application.

```
   GOSUB vars

   'Define the highest real axis
   top = 3

   GOSUB initial

   'Set error check with ERRORMASK in conjunction with
   'MOTION_ERROR. BEWARE not to use this for MC402-UK.

   ERRORMASK=32+64+256+512+1024

loop:
   'Continuous check for errors for all axes
   FOR i = 0 TO top
      BASE(i)
```

```
      IF MOTION_ERROR THEN GOSUB crash
   NEXT i

   ‘Continuous check for emergency stop
   IF IN(7)=0 THEN GOSUB e_stop
   GOTO loop

vars:
   ‘Initiate the application variables
   RETURN

initial:
   ‘Initiate the application

   ‘Stop your application tasks
   STOP “applic”
   WA(100)

   ‘Cancel all moves
   RAPIDSTOP
   FOR a=0 TO top
      BASE(a)
      CANCEL
      CANCEL
      CANCEL
      WAIT IDLE
      SERVO = OFF
   NEXT a
   WDOG = OFF

   ‘Initialise the axis parameters
   WA(1000)
   RUN “startup”
   WA(100)
   BASE(top)
   WAIT UNTIL SERVO=1
   WA(100)

   ‘Reset any following errors
   FOR b = 0 TO top
      BASE(b)
      DATUM(0)
   NEXT b

   ‘Enable axes
   WDOG=1

   ‘Run the application individually on a specified task
   RUN “applic”,2
   RETURN

crash:
   ‘This routine will cancel all moves in case of emergency
   ‘and will store the errors.
   VR(27 + ERROR_AXIS)=0
   BASE(ERROR_AXIS)

   ‘IN(16) TO IN(19) refer to the driver alarm inputs
   IF IN(16+ERROR_AXIS) THEN   VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+1
   IF AXISSTATUS AND 16 THEN   VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+16
   IF AXISSTATUS AND 32 THEN   VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+32
   IF AXISSTATUS AND 256 THEN  VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+256
```

```
    IF AXISSTATUS AND 512 THEN  VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+512
    IF AXISSTATUS AND 1024 THEN VR(27+ERROR_AXIS)=VR(27+ERROR_AXIS)+1024

    'Cancel all moves
    RAPIDSTOP
    FOR a=0 TO top
       BASE(a)
       CANCEL
       CANCEL
       CANCEL
       WAIT IDLE
       SERVO = OFF
    NEXT a
    WDOG = OFF

    'Stop all application programs individually. Another
    'option is to halt all programs with HALT.
    STOP "applic"

    'A bit may be set to restart the programs (if safe)
    'WAIT UNTIL IN(24)
    'GOSUB initial
    RETURN

e_stop:
    'Emergency Stop
    RAPIDSTOP
    FOR a=0 TO top
       BASE(a)
       CANCEL
       CANCEL
       CANCEL
       WAIT IDLE
       SERVO = OFF
    NEXT a
    WDOG = OFF

    'Stop all application programs individually.
    STOP "applic"

    WAIT UNTIL IN(7)
    RETURN
```

Index

A

absolute moves, 7
acceleration rate, 7
adding axes, 13
applicable PCs, 20
application, precautions, xiii
ASCII emulation, 165
axis
  adding, 13
  demand position, 17
  encoder, 6
  following error, 17
  measured position, 17
  repeat distance, 140
  servo, 6
  status, 91
  types, 6
  virtual, 6
axis connector, 27
axis types, 6

B

BASIC
  commands, 58
  data structures, 59
  driver I/O, 64
  functions, 58
  I/O access, 63
  labels, 60
  parameters, 58
  physical I/O, 64
  statements, 58
  variables, 59
  virtual I/O, 64
BASIC commands, functions and parameters
  listed alphabetically, 84
  ABS, 87
  ACCEL, 7, 87
  ACOS, 88
  add operator, 84
  ADDAX, 13, 88
  AND, 88
  ASIN, 89
  ATAN, 89
  ATAN2, 90
  ATYPE, 6, 90
  AUTORUN, 90
  AXIS, 59, 90
  AXISSTATUS, 91, 169, 184
  BASE, 58, 91
  BASICERROR, 92, 185
BATTERY_LOW, 93
CAM, 11, 93
CAMBOX, 12, 94
CANCEL, 13, 95
CHECKSUM, 96
CLEAR, 60, 96
CLEAR_BIT, 96
CLOSE_WIN, 97
comment field, 87
COMMSERROR, 97
COMPILE, 97
CONNECT, 12, 97
CONTROL, 98
COPY, 98
COS, 98
CREEP, 99
D_GAIN, 17, 61, 99
DAC, 61, 99
DAC_OUT, 100
DATUM, 13, 100
DATUM_IN, 64, 101
DECEL, 7, 101
DEFPOS, 5, 101
DEL, 102
DEMAND_EDGES, 102
DIR, 103
DISPLAY, 45, 103, 194
divide operator, 85
DPOS, 17, 103
EDIT, 104
ENCODER, 104
ENDMOVE, 104
EPROM, 65, 104
equal operator, 86
ERROR_AXIS, 105, 184
ERROR_LINE, 105, 185
ERRORMASK, 105, 184
EXP, 105
FALSE, 106
FAST_JOG, 14, 64, 106
FE, 17, 106
FE_LIMIT, 37, 106
FE_RANGE, 106
FHOLD_IN, 64, 107
FHSPEED, 107
FOR, 107
FORWARD, 9, 108
FRAC, 108
FREE, 109
FS_LIMIT, 109
FWD_IN, 38, 64, 109
FWD_JOG, 14, 64, 109
GET, 109
GOSUB, 60, 110
GOTO, 60, 110
greater than operator, 86
greater than or equal operator, 87

HALT, 111
I_GAIN, 17, 61, 111
IF, 111
IN, 112
INITIALISE, 58, 113
INPUT, 113
INT, 113
JOGSPEED, 114
KEY, 114
LAST_AXIS, 114
less than operator, 85
less than or equal operator, 85
LINPUT, 115
LIST, 115
LN, 116
LOCK, 116
MARK, 116
MERGE, 14, 116
MHELICAL, 10, 117
MOD, 118
MOTION_ERROR, 118, 184
MOVE, 7, 9, 119
MOVEABS, 7, 9, 120
MOVECIRC, 10, 121
MOVELINK, 13, 122
MOVEMODIFY, 125
MPOS, 17, 125
MSPEED, 125
MTYPE, 62, 125
multiply operator, 84
NEW, 126
NIO, 126
NOT, 126
not equal operator, 86
NTYPE, 62, 127
OFF, 127
OFFPOS, 127
ON, 127–128
OP, 128
OPEN_WIN, 129
OR, 129
OUTLIMIT, 130
OV_GAIN, 17, 61, 130
P_GAIN, 17, 61, 130
PI, 130
PLC_READ, 48, 131
PLC_TYPE, 132
PLC_WRITE, 48, 132
PMOVE, 63, 133
power operator, 84
POWER_UP, 65, 134
PP_STEP, 134
PRINT, 134
PROC, 59
PROCESS, 136
PROCNUMBER, 136
PSWITCH, 64, 136
RAPIDSTOP, 13, 137
READ_BIT, 138
REG_POS, 138
REGIST, 14, 64, 138
REMAIN, 140
RENAME, 140
REP_DIST, 140
REP_OPTION, 141
REPEAT, 141
RESET, 60, 141
RETURN, 110
REV_IN, 38, 64, 142
REV_JOG, 14, 64, 142
REVERSE, 9, 142
RS_LIMIT, 142
RUN, 143
RUN_ERROR, 143, 185
RUNTYPE, 67, 143
SCOPE, 144
SCOPE_POS, 145
SELECT, 145
SERVO, 61, 145
SET_BIT, 146
SETCOM, 146
SGN, 146
SIN, 147
SPEED, 7, 147
SQR, 147
SRAMP, 147
statement separator, 87
STEPLINE, 148
STOP, 148
subtract operator, 85
TABLE, 59, 149
TAN, 150
TICKS, 150
TRIGGER, 150
TROFF, 150
TRON, 150
TRUE, 151
TSIZE, 151
UNITS, 5, 7, 151
UNLOCK, 116
VERSION, 152
VFF_GAIN, 18, 61, 152
VP_SPEED, 152
VR, 59, 152
WA, 154
WAIT IDLE, 154
WAIT LOADED, 154
WAIT UNTIL, 155
WDOG, 61, 155, 184
WHILE, 155
XOR, 156

BASIC programs
compile description, 66
debugging, 150, 167

editing, 165
error processing, 68
managing, 65
multitasking, 58
priority, 66
run at start-up, 67
stepping, 148
storing, 65
tasks, 66
trace function (TRON/TROFF), 150

BASIC statement groups
axis parameters, 81
constants, 81
I/O commands and functions, 77
loop and conditional structure commands, 77
mathematical and logical functions, 80
motion control commands, 76
Motion Perfect statements, 81
PC data exchange commands, 83
program commands and functions, 78
system commands and parameters, 78
task commands and parameters, 83

buffer
actual move, 62
next move, 62
task, 62

C

C200HW-MC402-UK, comparison with, 20

cables
computer to MC Unit, 32
MC Unit to terminal block, 33
part numbers, 5
terminal block to Servo Drivers, 33

CAM control, 11

cancelling moves, 13

circular interpolation, 10

clearing following error, 100

command line interface, 65, 164

communication errors, 97

comparison with C200HW-MC402-UK, 20

components, 24

connections
axis, 27
I/O, 26
serial ports, 31
Servo Driver, 33
terminal block, 33

continuous moves, 9

continuous path control, 9

control system, 14

coordinate system
description, 5
scaling, 134

CP control. See continuous path control

CPU Unit, data transfers, 46

D

data format, 47

datuming. See origin search

debugging. See BASIC programs, debugging

deceleration rate, 7

decoder, 15

demand position, 17

demand speed, 7

derivative gain, 17

dimensions
MC Unit, 26
terminal block, 33

driver I/O, in BASIC, 64

E

EG control. See electronic gearing

electronic gearbox, 12

electronic gearing, 11

enable relay. See Servo Driver, enable relay

encoder, 14

encoder axis, 6

error processing, 68

errors
BASIC error code list, 185
BASIC run-time errors, 185
motion errors, 184
PC error flags, 187
PC transfer error flag, 51, 55
serial communication, 97

examples
controlling I/O, 195
coordinating two moving objects, 197
coordinating with mark detection, 198
grid positioning, 196
high-speed profiles, 197
master shell program, 199
pick and place, 198
product detection, 196
programming, 195
rotary table, 195
synchronising cutter, 196

F

features, 2
feedback pulses, 14
feedhold
  input, 107
  speed, 107
flags
  BASIC error, 45, 194
  indicator mode, 45, 194
  low battery, 45, 194
  PC control, 187
  PC error, 187
  PC transfer error, 45, 194
  transfer input, 45, 194
  transfer output, 43
floating point
  comparison, 61
  definition, 60
flying shear, 196
following error
  description, 17
  limit, 106
  limit setting, 37
  range, 106
functional specifications, 18

G-I

gain
  derivative, 17
  integral, 17
  output speed, 17
  proportional, 17
  speed feedforward, 18
general specifications, 18
global variables, 59
helical interpolation, 10
I/O access in BASIC, 63
I/O connector, 26
I/O specifications, 29
indicators, 24
initialisation, application, 61
installation, 23
integer, definition, 60
integral gain, 17
interpolation
  circular, 10
  helical, 10
  linear, 9
IORD instruction, 49
IOWR instruction, 52
IR/CIO Area Allocation, 42

J-L

jogging, 14, 171
  forward input, 109
  reverse input, 142
  speed, 114
labels, definition, 60
LED indicators, 24
limit switches
  description, 38
  forward input, 109
  reverse input, 142
linear interpolation, 9
linked CAM control, 12
linked move, 13
local variables, 60

M

master program, 69, 199
mathematical specifications, 60
measured position, 17
motion control
  algorithm, 16
  concepts, 5–14
  initialisation, 61
  types, 2
motion errors, 184
motion generator, 62
Motion Perfect
  axis parameters window, 168
  connecting to MC Unit, 158
  control panel, 162
  controller configuration window, 169
  debugging, 167
  desktop, 161
  features, 158
  firmware download, 178
  full controller directory window, 171
  I/O status window, 170
  jog screen, 171
  oscilloscope, 172
  program editor, 165
  project backup, 177
  requirements, 158
  retrieving backup, 178
  serial connection, 31
  simple examples, 163

Table editor, 169
terminal window, 164
tools, 164
VR editor, 169
Motion Perfect projects
backup, 159
consistency check, 160
description, 159
manager, 159
motor runaway, 36
mounting Units, 25
move loading, 63
moves
absolute, 7
calculations, 9
cancelling, 13
continuous, 9
defining, 7
execution, 62
jogging, 14, 171
merging, 14
relative, 7
multitasking, 58

N-O

noise, precautions against, 38
number format, 60
one-word format, 47
operating environment, precautions, xiii
origin search, 13
output speed gain, 17

P

PC data exchange
data in IR/CIO area, 48
error flag, 51, 55
methods, 46
one-word format, 47
three-word format, 47
transfer by CPU Unit, 49
transfer by MC Unit, 55
PC, applicable models, 20
physical I/O, in BASIC, 64
point-to-point control, 7
positioning
continuous path, 9
electronic gearing, 11
point-to-point, 7
precautions
application, xiii
Motion Perfect, 177
operating environment, xiii
safety, xii
servo system, 36
servomotor, 36
wiring, 38
precedence, 61
print registration, 14
delay time, 19
programming examples, 195
proportional gain, 17
PTP control. See point-to-point control

Q-R

quadrature, 14
registration. See print registration
relative moves, 7
restrictions on CS1 data exchange, 56
RS-232C communication errors, 97
RS-232C connections, 31
runaway
due to disconnected wiring, 37
due to faulty wiring, 36

S

safety precautions, xii
S-curve factor, 147
semi-closed loop system, 16
sequencing, 63
servo axis, 6
Servo Driver
alarm inputs, 64
connection to terminal block, 33
enable relay, 155
resetting alarm, 64
servo period, 66
servo system, 16
precautions, 36
software limit
forward, 109
reverse, 142
software reset, 177
Special I/O Unit Area, word allocation, 42–43
specifications
functional, 18
general, 18

I/O, 29
mathematical, 60
speed feedforward gain, 18
statements
axis, 58
description, 58
system, 59
task, 59
system configuration, 4

T-U

table variables, 59
task
buffer, 62
clock pulses, 150
priority, 66
terminal window, 65, 164
terminating resistors, 34
three-word format, 47
troubleshooting BASIC programs, 167
unit conversion factor, 5, 7

V-Z

variables
global, 59, 152
local, 60
table, 59, 149
virtual axis, 6
virtual I/O, in BASIC, 64
VR variables. See variables, global
VT100 Emulation, 165
wiring
axis connector, 27
I/O connector, 26
precautions, 38
Z-marker, 15

Revision History

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

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

Revision code	Date	Revised content
1	January 2000	Original production
2	May 2001	Modifications made for MC402-E version and for support of CS1 series. This manual has been extensively revised throughout all sections and appendices.