TS3000 series Robot Controller

TS3000 SCARA / LINEAR system
TS3100 SCARA / LINEAR system

INSTRUCTION MANUAL

ROBOT LANGUAGE MANUAL

Notice

- Make sure that this instruction manual is delivered to the final user of Toshiba Machine's industrial robot.
- Before operating the industrial robot, read through and completely understand this manual.
- After reading through this manual, keep it nearby for future reference.

TOSHIBA MACHINE CO., LTD.

The information contained in this manual is subject to change without prior notice to effect improvements.

Preface

This manual explains the SCOL robot language, commands and programming procedures as they apply to Toshiba Machine's TS series system robot controller. SCOL stands for "Symbolic Code Language for Robots" and is a robot language made up of various commands used to control the robot. By using these commands, it is possible to create programs to make the robot do what you want.
This manual is directed at those who have never written a robot program, and at those who have much programming experience. However, this manual only covers SCOL robot language. For the outline and operating method of the TS series system robot controllers, please refer to the following manuals:

Startup Manual
Operator's Manual

This manual is organized as follows:

[1] An Outline of Robot Language
This section explains the connection between robot language and robot movement, and presents a rough outline of commands used in robot language. Be sure to read this section in order to get a grasp of the fundamentals of robot language.

[2] Writing Programs in Robot Language

This section describes various rules for writing a program with robot language. Be sure to read this section before starting to write your own programs.

[3] Explanation of Robot Commands
This section details what each command means and does. The commands are listed in alphabetical order for your convenience. This section will come in useful when you write programs on your own.

[4] Program Examples
This section contains various examples of robot language programs. Be sure to use this section for reference when writing your own programs.

[5] Programming Hints and Warnings
This section explains timing considerations, things not to do, and things to watch out for when writing a program. Be sure to read it before beginning work on your own program. Also, be sure to look this section over should your program not be working the way you intended.

Table of Contents

Page

Section 1

An Outline of Robot Language

This section describes the connection between robot language and robot movement, and presents a rough outline of commands used in robot language.

1.1 Robot Movement

Robots do work in place of people. For example, let's say that somebody has to attach a part to a workpiece coming down a conveyor. The employee takes a part from a parts bin and attaches the part to a workpiece transported to his or her station by a conveyor. If we were to set up a robot to do this work instead, we would have an arrangement something like that shown in Figure 1.1.

In Fig. 1.1, the robot grabs a part from the parts feeder and attaches the part to a workpiece coming down the conveyor. Considering this work from the point of view of the robot, we would come up with a diagram like that of Figure 1.2. In this figure, the robot first moves straight down from Point B, and at Point A it grabs a part. After grabbing the part, the robot moves back up from Point A to Point B. From Point B, the robot moves the part to Point C, which is directly above the part attachment location Point D. The robot then drops down from Point C to Point D, and attaches the part to the workpiece. When the robot is finished attaching the part, it moves back up to Point C, and then finally back to Point B. This completes one work cycle.

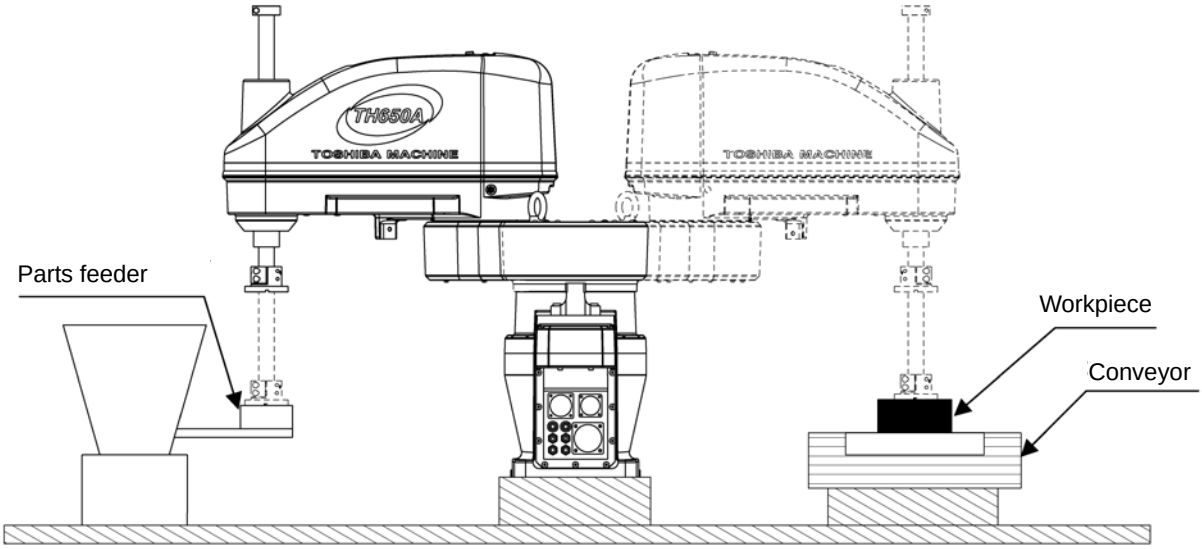


Fig. 1.1 Assembly work

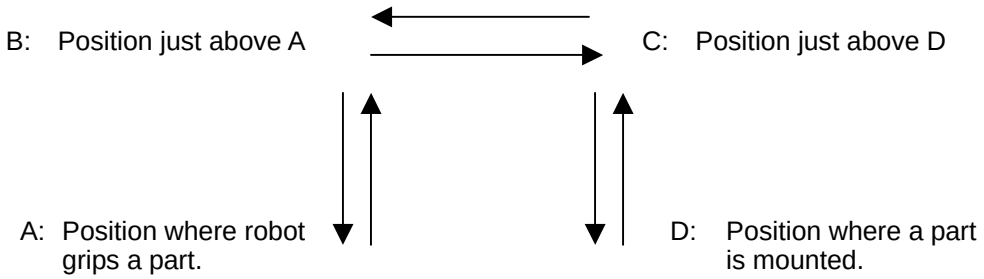


Fig. 1.2 Robot movement

1.2 Robot Language

Robots do assembly work and other tasks in place of people. However, someone still has to teach the robot what to do.

Robots will only do what you tell them to do, and it's important to tell them exactly what you want it to do.

Telling a robot what to do is called "teaching." Making a robot do what you taught it to do is called "playback." Of course, this only applies to what are called "playback robots," which repeat (or playback) the movements you instructed the robot when teaching.

There are various ways to teach a robot what to do. One is a method of making a robot do a job in order. For example, when carrying out painting and spot welding work, someone teaches a robot as he does it. The robot does the work as it is moving along a taught motion.

In order to achieve complicated work, however, we need change robot motion according to the states of peripheral equipment. As understood by the assembling example, there may be problems with change in attachment parts in response to types of workpieces carried by a conveyor, or with repeated attachment and detachment of the parts when mis-attachment occurs. The robot needs response to such circumstances that occur around the robot. Thus we must provide a method how the robot responds to various circumstances.

A language used for writing robot movement programs is called "Robot Language". A robot task expressed in the robot language is referred to as "Program", and a task to create the program is called "Programming". TS series employ the SCOL Language (which stands for Symbolic Code Language for robot) as our unique robot language. The example of the assembly work described in the previous paragraph can be expressed in the SCOL Language as shown below.

PROGRAM ASSEMBLY

```
    MOVE B          Move to Point B.
    OPEN1           Open Hand 1.
    MOVE A          Move to Point A.
    CLOSE1          Close Hand 1.
    DELAY  0.5      Wait 0.5 seconds before grabbing the part.
    MOVE B          Move to Point B.
    MOVE C          Move to Point C.
    MOVE D          Move to Point D.
    OPEN1           Open Hand 1.
    DELAY  0.5      Wait 0.5 seconds before letting go off the part.
    MOVE C          Move to Point C.
    MOVE B          Move to Point B.
END
```

The word PROGRAM marks the beginning of a program and the word END marks the end of a program. The name of this particular program is ASSEMBLY. MOVE A means to move to Point A. OPENi and CLOSE 1 mean to, respectively, open and close Hand 1. (There are two hands.) DELAY 0.5 means not to do anything for 0.5 seconds. Furthermore, the locations of Points A, B, C and D are defined (taught) beforehand by physically guiding the robot (in the teaching mode) to these points.

Thus we can express robot tasks in SCOL by arranging movement commands that are given to the robot in sequence for achieving the taught tasks.

1.3 Types of Commands

In the previous section, we saw how SCOL is used to express the action of the robot. Here, we explain a little bit more about SCOL commands themselves.
In addition to commands like "MOVE A" which actually move the robot, there are many other commands which do such things as send signals to external equipment or direct the robot to do the same thing over and over again. Table 1.1 presents a list of SCOL commands.
All SCOL commands can be roughly classified into one of six categories.

(1) Movement control commands

These commands move the robot. Commands which temporarily stop the robot, interrupt movement, or restart the robot are also included in this category. Commands which actually move the robot are called movement commands.

(2) Program control commands

Program control commands control the execution of the program by doing such things as executing certain parts of the program in accordance with external signals or causing portions of the program to be carried out repeatedly.

(3) I/O (Input/output) control commands

These commands are used to read in (input) or send out (output) signals to and from external equipment. Data input/output of hand open/close communication channel are included in the I/O control command.

(4) Movement condition commands

These commands are used to specify the configuration and speed of various joints of the robot while it is moving.

(5) Calculator commands

These commands are used to invoke (use) mathematical functions such as the trigonometric functions and the square root function.

(6) Movement reference commands

These commands are used to reference and check the movement of the robot. For example, these commands could be used to determine what percentage of a certain motion has been completed at a certain time. They also include commands for setting of the timer used during program execution and referencing to robot operating mode.

These commands, in combination with other commands, are used to output signals to the external equipment when the robot has completed up to 70% of the given task, or to branch the program when movement of the robot is recognized as an error if the movement is not finished within a specified time. By combining these commands, tasks suitable for the robot can be programmed.

Table 1.1 Functions of the SCOL language

Type	Purpose	Commands
Movement control commands	(1) Move the robot.	MOVE, MOVES, MOVEC, MOVEA, MOVE1, READY MOVEJ
	(2) Temporarily stop the robot.	DELAY
	(3) Move the robot hand.	OPEN1, OPENI1, OPEN2, OPENI2, CLOSE1, CLOSEI1, CLOSE2, CLOSEI2
	(4) Interrupt or restart operation.	BREAK, PAUSE, RESUME
Program control commands	(1) Monitor external signals, timers, etc.	ON ~ DO ~, IGNORE IF ~ THEN ~ ELSE, WAIT, TIMER
	(2) Control program execution.	PROGRAM, END, GOTO, RCYCLE, RETURN, FOR ~ TO ~ STEP ~ NEXT, STOP TASK, KILL, SWITCH, TID, MAXTASK
	(3) Remarks and comments on program	REMARK

Type	Purpose	Commands
I/O control commands	(1) Input and output of external signals.	DIN, DOUT, PULOUT, RESET, BCDIN, BCDOUT
	(2) Input and output of communication data.	PRINT, INPUT
Movement condition commands	(1) Specify conditions for controlling robot movement.	CONFIG, ACCUR, ACCEL, DECEL, SPEED, PASS, TORQUE, GAIN, ENABLE, SETGAIN, DISABLE, NOWAIT, PAYLOAD, FREELOAD, SWITCH, MOVESYNC
Palletizing command	(1) Load a library.	LOADLIB
	(2) Initialize a pallet.	INITPLT
	(3) Move a pallet to the specified position.	MOVEPLT
Calculator functions	(1) Perform calculations for real numbers.	SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SQRT, ABS, SGN, INT, REAL, LN, MOD, LOGIO, EXP, AND, OR, NOT
	(2) Perform calculations involving positional and coordinate data.	HERE, DEST, POINT, TRANS
	(3) Use an array.	DIM, AS
Movement reference commands	(1) Check robot movement.	MOTION, MOTIONT, REMAIN, REMAINT
	(2) Check system movement.	MODE, CONT, CYCLE, SEGMENT
	(3) Assign a coordinate system.	TOOL, BASE, WORK
Others	(1) Define a variable.	GLOBAL, DATA, END
	(2) Restore an updated value in the program file.	RESTORE
	(3) Save data at power OFF.	SAVEEND

Section 2

Writing Programs in Robot Language

In Section 1, we got a rough idea of what a robot language is and how it works. Now, in Section 2, we will describe how to write a program in robot language.

2.1 Program Configuration

Below we present a general outline of program configuration with the SCOL language.

2.1.1 Files

In order to get the robot to perform a task, you need both a program written in robot language and positional data for use by the program. That is, for the TS series, you have to have a matched set of a program (or programs) and positional data. This matched set is called a file. Program execution and editing are on the file basis.

2.1.2 Program

A program is an expression using robot language of a set of operations performed by the robot. A program can also be called and used by another program. It is possible preparing frequently-used operations and preset operations as a single program and calling these programs when needed. Programs that are called in this way are called subprograms, and the program that calls the subprogram is called a main program.

A single file can include multiple programs. When a program is executed, unless specified otherwise, the program at the start of the file is executed as the main program. When a subprogram is called, the subprogram must be located in the same file as the main program. Also, even if multiple programs are contained in a file, all of these programs are not executed in order. Instead, the programs are executed until the end of the main program as a single unit.

Also, multiple programs can be executed simultaneously using the TASK command (multitask execution). For details on multitask execution, see Para. 2.8 “Programs.” Programs are edited from the teach pendant using the program editor function of the controller. For details on how to use the editor, see the Operator’s Manual.

As far as the robot is concerned, its job is over when the main program is completed (i.e., when the robot reaches the final END statement of the main program), and if the other programs have not been called by that time they will never be called.
A plural number of programs can be executed at the same time, using the TASK command (multitask execution). For details of the multitask execution, see Para. 2.8.
Programs are edited with the teach pendant using the program editor function. For information on how to use the editor, see the "Operator's Manual."

2.1.3 Positional Data

Positional data for use in a program (or programs) must be placed in the same file as the program (or programs). Positional data in a file can be accessed (used) by all programs in that file. However, positional data in a file cannot be accessed by any programs not in that file.
Positional data is "fed" to the robot using the data editor function of the controller. See the Operator's Manual for information on how to use the data editor.

2.2 Character Set

The SCOL character set is made up of alphanumeric characters and the following special symbols.

Alphanumeric characters

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0

Special symbols

" ' ( ) + – * / , . < > =
! [ ] ( ) % ^ & ?

With the exception almost all of the small letters, these characters and symbols can all be input from the teach pendant. When executing a program, the robot makes no distinction between capital letters and small letters. For reading method of symbols, see "Appendix E."

2.3 Identifiers

In the SCOL robot language, identifiers are used to express commands, program names, variable names, and labels (which are used to specify program branches). Identifiers must start with an alphabetic character. There is no particular limit on length, although the robot will only differentiate the first ten alphanumeric characters. The robot does not care whether you use capital or small letters, since it will treat them the same anyway. Also, special symbols and spaces cannot be used in identifiers.

Special characters or spaces are used to separate identifiers.

Example:

TOSHIBAROB
Toshibarob
TOHIBAROBOT

The three identifiers above are all treated as the same identifier “TOSHIBAROB”.

Some identifiers have already been defined by the SCOL language itself. These are called reserved words, and you as the programmer cannot use them for any other purpose except for that already defined. A list of reserved words is shown in Appendix B. In addition to SCOL commands, you will find words used in the computer system and words set assigned for future expansion.

Do not use identifiers with the same name for different meanings. For example, when creating a program, do not use the same identifier for both the program name and variable name.

2.4 Variables and Constants

Not all data takes the same form, and these different forms of data are called data types. Scalar type (integer type, real number type and character string) and vector type (position type, coordinate type and load type) can be used in the SCOL language. Variables are divided into global variable and auto variable according to the definition method. All taught data and variable defined in the area between GLOBAL and END are called the global variable. These variables can be referred and changed from any part of the program. For all data types of global variables, the array can be declared. For descriptions of global variable and array, see Para. 2.8.5.

The work area in the controller is used for all data. The defined value is substituted for the global variables other than the array without a specific initial value at the start of the program. If the value is entered for the variable during program execution, only the work area is changed. If the power of controller is turned off, execution file is reselected or the file is edited, work area is reset by the variable's initial value saved in the file and the changed value is lost accordingly. This is also applicable for change of the taught data. If the data in the file is to be overwritten, the RESTORE command should be executed in the program.

2.4.1 Scalar Data

There are three types of scalar data, i.e., integers, real numbers and character strings. Scalar type auto variables can only be used in the program in which they were declared. That means that if you use a variable with the same name in another program, the two variables will be completely independent and have nothing to do with each other. Therefore, when passing data from one program to another, make it a point to, if possible, redefine the variable as the scalar type global variable or declare the arguments in the program. See Para. 2.8 "Programming."

(1) Integer data

(a) Constants

SCOL can handle integer values ("whole numbers") in the range of –2147483648 to + 2147483647. When an integer is used as a constant in a program, if it is positive, directly describe the value; if it is negative, describe the value following the – symbol. If a value of 11-digit or over is entered by the INPUT command, an error occurs.

Example:

0
234
–39208
5963

(b) Variables

Variables are distinguished by identifiers and can be in the range of –2147483648 to + ~147483647, just as above. The data type of a variable is determined by the data type of the first number you assign to that variable. For example, if the first thing you assign to a variable is an integer, all other numbers substituted into that variable will become integers. That means that if you later try to insert a real number into this variable, the controller will chop off all the decimal places and treat what is left as an integer.
The variable comes in two types; the global variable which is valid in the entire program and the general variable which is valid in a part of the program. The global variable can be changed from any part of the program.

(c) Logical values

Logical values are used in the program when making conditional judgments. Logical expressions and commands such as DIN (which check input signals) return logical values.
A logical value may have one of two values; TRUE or FALSE. Internally, logical values are treated as integers with 1 being TRUE and 0 being FALSE.

Note: Strictly speaking, 0 is considered as FALSE and everything else is considered as TRUE.

(2) Real data

With SCOL, numbers are treated as real types with the exception of certain special cases.

(a) Constants

SCOL can handle real numbers having an absolute value in the range of approximately 5.87×10^{-39} (2^{-127}) to 6.80×10^{38} $\{(2^{23}-1)\times2^{106}\}$. The number significant digits for the mantissa [the mantissa is the part of the number to the right of the decimal point) is approximately 7 in Base 10. (The precision is 2^{23}).

The number of allowable digits is 9 for the integer and 3 for the decimal. If a value consisting of more than these digits is entered by the INPUT command, an error occurs.

When a real number is used in the program, if it is positive, directly describe the value; if it is negative, describe the value following the – symbol.

When the decimal part is 0, it is omissible. However, when the decimal point is omitted, the data are treated as integer type data. In addition, since the integer part cannot be omitted, even if the absolute value of a numeric value is less than 1, it is necessary to designate 0 to the integer part.

Example:

1234.567
–28.16
0.00985
1234567.
–369.

As mentioned above, the precision of the computer is somewhat limited when handling decimal values. Usually this is no problem if the number of decimal places is reasonable. Therefore, when working with the robot, try to use the following as the minimum set units.

Distance (X, Y, and Z data)	0.001 mm
Angles (C data)	0.001 deg.
Time	0.01 sec.
Rates (Speed, torque, etc.)	1 %
Mass	0.01 kg
Inertia	0.01kg.m

(b) Variables

Variables are distinguished by identifiers and have the same range as listed above for constants. The data type of a variable is determined by the data type of the first number you assign to that variable. For example, if the first thing you assign to a variable is a real number, that variable will become a real type.

(3) Character strings

Character strings can only handle constants. They are expressed by placing one or more characters between quotation marks. In the example below, the character string is SCOL MESSAGE.

Example: "SCOL MESSAGE"

2.4.2 Vector Data

As opposed to scalar-type data which only holds one data element, vector-type data holds multiple data elements. There are three types of vector data in SCOL; positional vectors, coordinate vectors and load vectors.
Vector-type data can be expressed by enclosing one to five elements in brackets { }. In addition to positional vectors, coordinate vectors, and load vectors, vector-type data is also specified for TORQUE and GAIN commands by enclosing in brackets { }.
Vector type data other than the vector type global variable such as data taught by the data editor are temporarily stored in the working area of the controller. The data are not created in the file. The vector type variable can be used only in the declared program. Thus, even if the same variable is used in another program, the content of the former does not accord with that of the latter. When data are passed from one program to another program, the passed data should be redefined as the vector type global variable or it should be an argument. For details of arguments, see "2.8.2 Subprograms."

(1) Positional data

Positional data is used by the robot to describe positions. Positional vectors have the following format.

(X, Y, Z, C, T, <configuration>)

X, Y, Z, C and T are coordinate values represented by real numbers. Units are in millimeters or degrees.

<Configuration> holds an integer from 0 to 2 that describes the set-up configuration of the system.

0 ... Free (Set-up of the system is undefined)
1 ... Left hand system
2 ... Right hand system

(2) Coordinate data

Coordinate data is used by the robot to specify coordinate systems. Coordinate vectors have the following format:

(X, Y, Z, C)

X, Y, Z and C are coordinate values represented by real numbers. Units are in millimeters or degrees.

Coordinate vectors allow one to convert between different coordinate systems as shown in Figure 2.1. In the figure, we have an original coordinate system X, Y and Z. Then, with data provided by a coordinate vector (x, y, z, c), the original coordinate system is shifted parallel along its axes by the amounts x, y and z. This forms a new coordinate system centered about 0'. Once this is done, we twist the new coordinate system around the Z' axis by an amount c. We are now finished orientating our new coordinate system X', Y', and Z'.

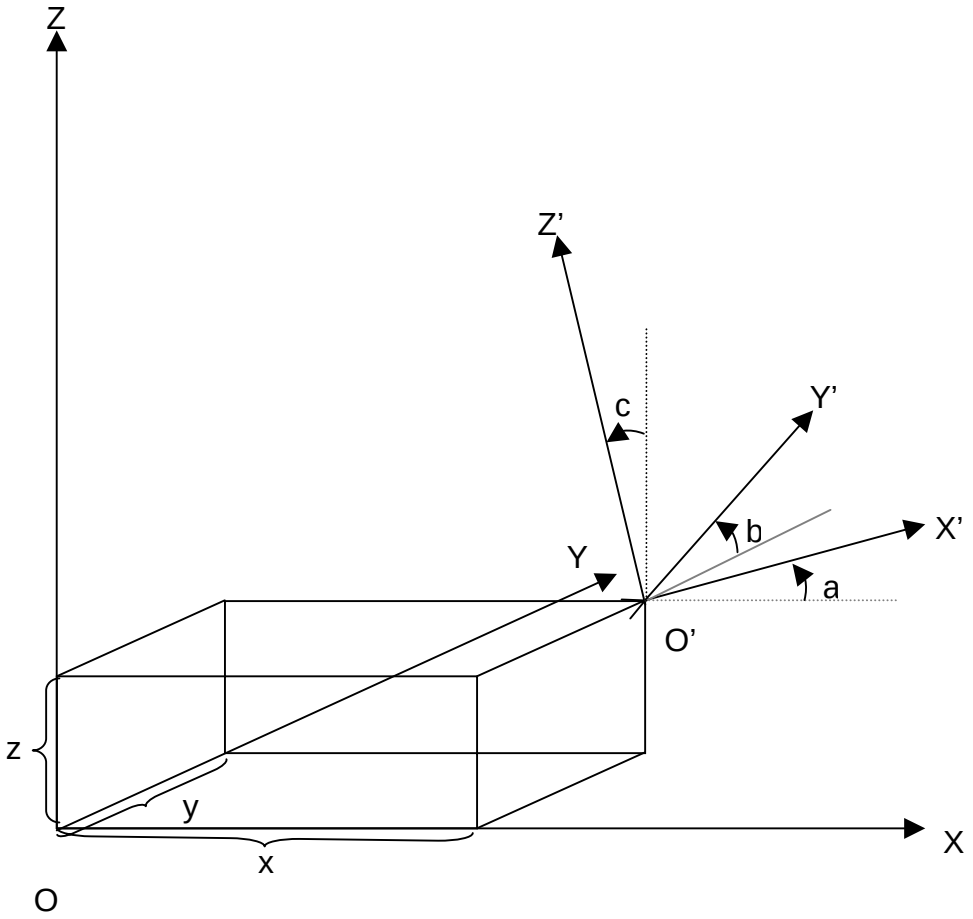


Fig. 2.1 Coordinate transformation

(3) Load data

Load data is used to define the physical loads acting on the end effector (hand) of the robot. Load vectors have the following format.

{<mass>, <center of gravity offset>}

<mass> is the mass of the load acting on the tip of the robot hand. Units are in kg.

<center of gravity offset> is the amount representing the distance between the center of gravity applied to the tip of the robot hand and the center of the tool flange of the robot (unit: mm).

2.4.3 System Variables

The SCOL language provides special variables that are used in the programs to specify and referent robot operating conditions. These variables are called system variables. Just like other variables, you can refer to these variables in the program, change their value, etc. However, you have to be careful when setting or substituting values into system variables since doing this will directly effect robot operating conditions.

A list of system variables is presented below in Table 2.1.

Table 2.1 List of system variables

Name	Description	Effective values	Initial value	Data type
CONFIG	Robot configuration	0, 1, 2	0	Integer type
ACCUR	Positioning accuracy	0, 1	1	Integer type
ACCEL	Acceleration (during acceleration)	0 ~ max %	100	Integer type
DECEL	Deceleration (during deceleration)	0 ~ max %	100	Integer type
SPEED	Speed of movement	0 ~ max %	100	Integer type
PASS	Short-cut movement parameter	0 ~ 100%	100	Integer type
TORQUE	Maximum torque on each axis	0 ~ max %	300	Vector type
GAIN	Servo gain on each axis	0.1	1	Vector type
TOOL	Tool coordinates		0	Coordinate type
BASE	Base coordinates		0	Coordinate type
WORK	Work coordinates		0	Coordinate type
TIMER	Timer	0 ~	–	Real type
ERROR	Error information	–		Integer type
PLAYLOAD	Load on the robot	0 ~	0	Load type
SWITCH	Multitask	0, 1	1	Integer type
TID	Task number	1 ~	–	Integer type
PLCDATAR1 ~ 8	Simplified PLC interface	0 ~ 65535	0	Integer type
PLCDATAW1 ~ 8	Simplified PLC interface	0 ~ 65535	0	Integer type

Note: Maximum values are set separately for each system.

Should you change the contents of a system variable related to movement control, that change will not take effect until the next motion; it will have no effect at all on a motion in progress at the time; However, by using a WITH construct, it is possible to temporarily set a system variable with regards to one motion command. For example:

MOVE Al WITH SPEED = 50

Furthermore, be warned that SCOL does not check to see whether a value substituted into a system variable is within the permissible range. Should the value not be in the permissible range, the system sets a value according to the following rules.

- Should you try to insert a value less than the minimum permissible value, the minimum permissible value will be entered in its place.
- Should you try to insert a value greater than the maximum permissible value, the maximum permissible value will be entered in its place.

See Section 3 for details on how to use system variables.

2.4.4 System Constants

In order to make programs easier to read (and thereby debug), SCOL provides the system constants shown in Table 2.2. These names can be substituted into the program in place of numbers in order to make it easier to see what you are doing. However, be sure to use them only in the locations specified in the Comments column of Table 2.2.

Table 2.2 List of system constants

Name	Value	Comments (Locations for use)
FREE LEFTY RIGHTY	0 1 2	In the system variable CONFIG In the POINT command
COARSE FINE	0 1	In the system variable ACCUR
OFF ON	0 1	In the system variable GAIN In the SETGAIN command
PAI	3.141593	Pi value
CONT CYCLE SEGMENT	0 1 2	In the MODE command

2.5 Expressions

This paragraph describes expressions provided by SCOL for substitution, calculation and judgement.

In SCOL language, expressions can be not only used independently to perform substitution and calculation, but also used within commands. Expressions include computational expressions where the calculation result is substituted into a variable and logical expressions that determine greater than/less than or true/false results. The operands shown below can be used. Please note that execution result of 0/0 is -1 and of 0 ^ 0 is 0 instead of an error as would normally be expected.

Table 2.3 Operands

Type	Operand	Function	Example
Arithmetic functions	^	Exponentiation	A ^ B (A to the B power)
	–	Minus sign	–A
	*, /	Multiplication, division	A * B, A / B
	+, –	Addition, subtraction	A + B, A – B
	MOD	Remainder	A MOD B (The remainder when A is divided by B.)
	=	Substitution	A = B (Puts the value of B into A.)
Relational function	= =	Equal	A = = B
	< >, > <	Not equal	A < > B, A > < B
	<	Less than	A < B
	>	Greater than	A > B
	< =, = <	Less than or equal	A < = B, A = < B
	> =, = >	Greater than or equal	A > = B, A = > B
Logical operands	AND	Logical product	A AND B
	OR	Logical sum	A OR B
	NOT	Negation	NOT A
Functions	SIN	Sine	SIN (A)
	COS	Cosine	COS (A)
	TAN	Tangent	TAN (A)
	ASIN	Arcsine	ASIN (A)
	ACOS	Arccosine	ACOS (A)

Type	Operand	Function	Example
Functions	ATAN	Arctangent	ATAN (A)
	ATAN2	Arctangent	ATAN2 (A, B) (Arctangent of A / B)
	SQRT	Square root	SQRT (A)
	ABS	Absolute value	ABS (A)
	SGN	Sign	SGN (A) (Extract and return the sign of A)
	INT	Changes number to an integer.	INT (A)
	REAL	Changes number to a real number.	REAL (A)
	LN	Natural logarithm	LN (A)
	LOG10	Common logarithm	LOG10 (A)
	EXP	Exponential base e	EXP (A)

Parentheses ( ) may be used inside the expressions.

2.5.1 Computational Expressions

In the SCOL language, the results of computations on the right side of an equal sign are placed on the left. Variables and constants may be used in the expressions.

(1) Order of computational priority

In SCOL language, calculation is performed in the same order of priority as regular computational operations. Specifically, operations are performed based on the rules below.

- If the expression contains brackets, the operations inside the brackets are performed first.
- Operations are performed in the order of assignment of negative signs, calculation of exponents, multiplication and division (*, /), and addition and subtraction (+, -).
- If two operations have the same priority, the operations are performed from the left to the right of the expression.

For example:

$$a = b + c * d / (e - f) - g,$$

The order of computation for the above expression is:

1.	Calculate e – f.	e–f
2.	Calculate c * d.	c * d
3.	Divide c * d by e – f.	(c * d) / (e – f)
4.	Add the above result to b.	b + (c * d) / (e – f)
5.	Subtract g from the above result.	(b + (c * d) / (e –f)) – g

Table 2.4 presents the order of computational priority for various operations.

Table 2.4 Order of computational priority

Priority	Operation	Operand	Grouping convention
High	Parenthesis	( )	Left to right
	Assignment of vector elements	.	Left to right
	Assignment of negative signs and negations	–, NOT	Right to left
	Exponentiation	^	Left to right
	Multiplication, division, remainder	*, /, MOD	Left to right
	Addition, subtraction	+, –	Left to right
	Comparison	<, >, < =, > =, = <, = > = =, < >, > <	Left to right
	Equality, inequality	AND, OR	Left to right
Low	Logical product, logical sum	=	Right to left
	Substitution		

Note: Explanation of grouping convention:

Left to right 1 + 2 – 3 is interpreted as (1 + 2) – 3.

Right to left NOT–3 is interpreted as NOT (–3).

(2) Computation of scalar type data

Scalar type data can be used in calculations in combination with computational operands. However, should even one number in an expression be a real number, the output of that expression will also be a real number. Also, the following functions will all return a real number.

SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SQRT, REAL, LN, LOG10, EXP

When the variable on the left side of the equation is an integer type and the output of the calculation is not an integer, the output will be converted into an integer before being assigned to the variable. Do not forget, however, that all decimal points are chopped off when a real number is converted to an integer. On the other hand, when converting from an integer to a real number, the number of significant digits is limited. When you want to make it clear what kind of data type you are dealing with, use the INT or REAL command.

Note that character strings cannot be used in calculations. Calculations may be carried out between the elements of vector-type variables and scalar data. In this case, an element specifier is appended to the end of a vector-type variable to specify the element which is involved in the calculation. The value of the element is then drawn out from the vector-type variable and used in the calculation.

As element specifiers, ".X", ".Y", ".Z", ".C" and ".T" may be used. You may also numerically specify the element position with ".1", ".2", ".3", ".4" and ".5."

Examples:

```
A = POINT1.X/25
GAIN={GAIN. 1,GAIN.2,0,0,0}
```

Note: You can only use this to return the value of an element from the inside of a vector-type variable. You cannot change the value of the element itself.

(3) Computation of vector-type data

You can add and subtract corresponding elements of two vectors. Computation is a possible only between the same type variables. The <CONFIG> element is not involved in the calculations but rather takes the value of the variable substituted into it.

Example: Given the following two position vectors and two coordinate vectors;

P1: (10, 20, 30, 40, 50, RIGHTY)
P2: (–5, 10, –15, 20, –25, LEFTY)
H1: (100, 50, –50, 0)
H2: (12, 34, 56, 78)

and performing the following operations,

P3 = P1 – P2
H3 = H1 – H2

we obtain:

P3: (15, 10, 45, 20, 75, RIGHTY)
H3: (88, 16, –106, –78)

Note: The <CONFIG> element in P3 is indeterminant.

(4) Substitution into vector data types

The following methods are available to substitute (insert) a constant, a variable or the result of a computation into an element of vector-type data.

(a) Commands to convert a row of scalar-type data into vector-type data

A POINT command and a TRANS command are available to convert rows of scalar data into a vector data. POINT converts scalar data into positional vector data, and TRANS converts scalar data into coordinate vector data. If an element is not included in the expression, that element is treated as 0 within the expression. For details on how to use these commands, see "Section 3."

Example:

```
P1 = POINT(P2.X, P2.Y, P2.Z + 50, 0, 0)
H1 = H2 + TRANS(100, 100)
```

Generally, vector-type data declarations are made as shown below.

Positional data POINT (X, Y, Z, C, T, <CONFIG>)
Coordinate data TRANS (X, Y, Z, C)

X, Y, Z, C and T are coordinate values represented by real numbers. Units are in millimeters or degrees.

<CONFIG> stands for "configuration" and holds an integer from 0 to 2 that is used to describe the set-up of the system.

0 ... Free (Set-up of the system is undefined)
1 ... Left hand system
2 ... Right hand system

Any omitted elements are taken as "0".

Note 1: In order to make it clear just what kind of data type you are using, always try to use the POINT command when creating positional type data and the TRANS command when creating coordinate type data.

Note 2: When position data which have not been taught are used in a program of the robot language, the position data are temporarily stored in the controller memory. Thus, when the program is reset, the position data are cleared. The position data are only valid in the program which uses data. Therefore, to use the position data in a subprogram, it is necessary to pass it as an argument. For details of arguments, see "2.8.2 Subprograms."

Note 3: The substitution and reference to the array type data (type of variable name [index number]) are dealt in the same manner as the original data type (scalar type and vector type) of the array type data.

2.5.2 Logical Expressions

With SCOL, logical expressions can be used in combination with the commands IF, WAIT and ON. Also, six relational operands are available (<, >, < = (or = <), > = (or = >), < > (or > <), and = =). Also, logical expressions may be combined using the logical operands AND, OR and NOT. Scalar constants, scalar variables and the results of calculations may be used as data in logical expressions.
When evaluating equivalence, use the "= =" sign and not the "=" sign. When comparing real numbers, differences of 0.001 or less will be ignored.
Logical expressions will return an integer value of 1 if true and 0 if false. The calculation result is an integer type number.

Examples:

1) IF K = =K2 * K3 THEN K = K2
   ON MOTION > = 50 DO DOUT (1,2)

2) When IF J1 THEN GOTO TRUE1 ELSE GOTO FALSE1 is executed, If J1 is an integer 0, or a real number with absolute value J1 is equal or less than 0.001, the comparison is considered as FALSE, then the program branches off to FALSE1.
   If J1 is a value other than an integer 0, or an real number with absolute value J1 is greater than 0.001, the comparison is considered as TRUE, then the program branches off to TRUE1.

2.6 Labels

With the SCOL language, program branches are specified by labels placed at the beginning of the branch destination. When labeling a statement as a branch, put a colon at the end of the identifier.

When directing the program to branch to another location with the GOTO command, do not put a colon at the end of the label.

Program branching may only be carried out within a single program. You cannot branch from one program to another. Also, you may use the same labels in different programs, but you cannot use the same label in a single program.

Examples:

```
LOOP1:  MOVE P1
GOTO    LOOP1
```

2.7 Remarks and Comments

The SCOL language allows you add comments to your program in order to make it easier to understand (and debug). Comments can be entered by using the teach pendant to type in whatever you want to say. Remarks and comments are written in any one of the following formats.

(1) REMARK command

You can write what you want to say after a REMARK command. In the REMARK command, everything written until the end of the line (until the [EXE] key is pressed) is treated as a comment, and it is not executed as a program. Because the REMARK command is an independent command, it cannot be written after another command.

Example:

```
REMARK SCOL SAMPLE PROGRAM
```

(2) Single quotation mark

Text written after a single quotation mark (') is treated as a comment. This method enables you to insert comments after other commands. Everything written after the single quotation mark (') until the end of the line (until the [EXE] key is pressed) is treated as a comment, and it is not executed as a program.

Example:

```
MOVE P1        'MOVES THE ROBOT TO P1
'*** SCOL COMMENT SAMPLE ***
```

2.8 Programs

This paragraph describes SCOL programs.

2.8.1 Program Declaration

Declaration of a combination program of SCOL commands is written in the following format.

```
PROGRAM <name of your program>
  Contents of your program
END
```

A program is made up of everything from the PROGRAM statement to the END statement. Write a program name after the PROGRAM statement. The program name is expressed using an identifier. Put the content of the program between the PROGRAM statement and the END statement.

Example:

```
PROGRAM SAMPLE    Program name "SAMPLE"
  REMARK SAMPLE   Comment
  SPEED=20        Set the movement speed to 20% of the maximum
                  speed.
  MOVE Al         Move the robot to position Al.
  DELAY 0.5       Wait for 0.5 sec.
  MOVE A2         Move the robot to position A2.
  DELAY 0.5       Wait for 0.5 sec.
END               End of program
```

As shown in the example, the body of the program is composed of statements made up of an arrangement of SCOL commands. A new line is created every time the EXE key is pressed. Up to 130 characters can be contained in a single line. You may add spaces as you wish in order to make the program neater and easier to read. You can write a remark and comment using the symbol (') in a statement.

Note: No spaces can be placed between characters structuring a word of a command and identifier.

2.8.2 Subprograms

You can call up a subprogram by just writing its name in the main program.

Example:

Main program

```
PROGRAM MAIN
   REMARK *** SAMPLE 1 ***
   SUB1
END
```

Sub program

```
PROGRAM SUB1
   REMARK *** SUBPROGRAM NO. 1 ***
   Body of subprogram
   RETURN
END
```

A RETURN command should be inserted in subprograms to send control back to the main program. If you forget to write RETURN, SCOL will forgive you and pretend that there is a RETURN command in front of the END statement.

When wishing to pass data between subprograms and the main program, you have to first specify arguments for the subprogram. When an argument is specified in a subprogram, the program statement should be written like this:

```
PROGRAM <program name> (<names of arguments>)
```

The argument is specified within parentheses ( ) following the program name of the subprogram. The argument values are inserted to the specified variable names in the subprogram. When using multiple arguments, write commas (,) to separate the variable names in parentheses ( ). The maximum number of arguments is 10.

In the main program, specify the data to be transferred when the subprogram is called in parentheses ( ) after the subprogram name. The data specified in parentheses ( ) is transferred in the specified order to the variable names of the subprogram. The argument data from the main program is substituted within the subprogram, and when the data is changed, the data corresponding to the variables in the main program is also changed at the same time.

Example:

Main program

```
PROGRAM MAIN
    REMARK *** SAMPLE 2 ***
    K1 = 15
    K2 = 28
    SUB2(K1, K2, K)
    PRINT K
END
```

Sub program

```
PROGRAM SUB2(N1, N2, N3)
    REMARK *** SUBPROGRAM NO. 2 ****
    N3 = N1 + N2
    RETURN
END
```

In the above example, three arguments are being passed off between the main program and subprogram. Specifically, K1 of the main program is passed over as N1 of the subprogram. Similarly, K2 of the main program is passed over as N2 of the subprogram. The subprogram adds N1 and N2, and puts the result in a variable called N3. When this happens, the value of K in the main program also changes.

When this program is executed, the values for K1=15 and K2=28 are added in the subprogram, and the result K=43 is displayed on the teach pendant using the PRINT command in the main program.

Note that subprograms may not call themselves. Also, should you call a subprogram which is in another file, the controller will not understand you and instead will treat the name of that subprogram as an error.

Note 1: An expression itself, result of vector data expression such as position data and vector data element cannot be designated as an argument.

Note 2: When a constant is used as an argument, it cannot be substituted into a variable according to a subprogram.

Note 3: For a variable which is an argument to a subprogram, a value should be substituted into the variable before the subprogram is executed.

2.8.3 Library

The SCOL language does not allow you to use subprograms which are not in the same file as the main program. However, by putting especially useful subprograms in the library file, you can access the subprogram from the main program.
When writing your own subprogram to add to the library file, enter the program in that file just like you would enter any other subprogram. For information on how to enter a program into a file, refer to the Operator's Manual.

The library comes in the following two (2) kinds.

[1] System library
This library is always loaded at program execution.
The file name is "SCOL.LIB" which cannot be changed, but the contents can be added or changed when necessary.
The OPEN1 and CLOSE1 commands are the subprograms which are included in this library file. The contents of the library file which is the standardly provided in the robot controller system disk are shown in Appendix C.
When you create a new subprogram in the system library, add it to the end of the current library file (SCOL.LIB).

[2] Dynamic link library
This is the library file that the user can load when necessary.
The name of library file is _____.LIB.
The dynamic link library is the user program and you should declare its loading.
You can declare the loading of a library file in the GLOBAL area of the user program in the following manner.

```
GLOBAL
     LOADLIB PALLET.LIB     ← Declaration of loading library.
END
PROGRAM SAMPLE1
     ~ ~ ~ ~ ~ ~
     Omitted
     ~ ~ ~ ~ ~ ~
END
```

Up to five (5) dynamic link libraries can be loaded at the same time.

Some SCOL commands use this dynamic link library.

Even when such a command is used, declaration of loading is necessary with the command of “LOADLIB + file name.”

When a program which uses the dynamic link library is executed, the controller creates a temporary file named “SCOLLIB.TMP (SCOL.LIB + dynamic link library).” When a sufficiently free space is not available in the user program area, the program may not be executed.

Caution:
If a name of program to be executed is the same name as the subprogram included in the selected library file, the subprogram included in the library file is executed in the automatic operation.

Operation of dynamic link library function

When a program which requires the dynamic link library (i.e., program including the LOADLIB command) is selected, the system operates in the following manner.

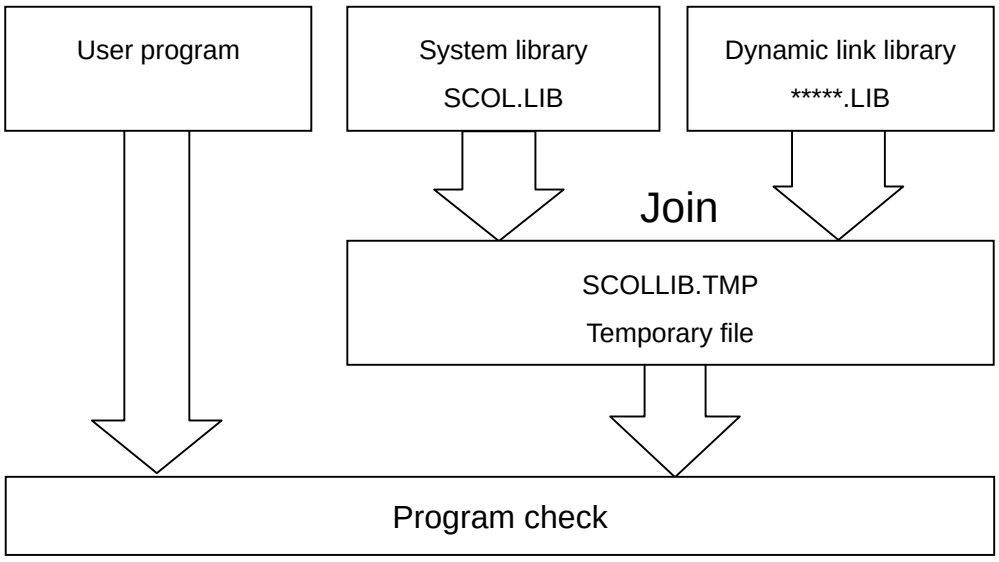
[1] Opening the user program.

[2] Check of the LOADLIB command (i.e., check of the file name).

[3] Joining SCOL.LIB with the dynamic link library to make SCOLLIB.TMP.

[4] Program check. Unless there is a problem, the SELECT command finishes.

[5] When the SELECT command has finished normally, SCOLLIB.TMP is deleted automatically. (When an error occurs, SCOLLIB.TMP is not deleted.)



If an error occurs in the library due to some cause, the system displays the message saying “LINE???:LIB>ERROR–***”.
Shown in “LINE???” is the line number of SCOLLIB.TMP. Confirm the contents of SCOLLIB.TMP, then modify the library file.

2.8.4 Multitask Processing

This paragraph describes how to use the multitask function of the SCOL language together with the relevant commands and system variables.

Program execution of single task and multitask operation is shown in Fig. 1 and Fig. 2. The number in the figure designates the order of the program execution.

Specific timing of change-over from program to program (task change) is described later.

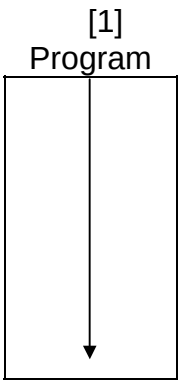


Fig. 1 Single task operation

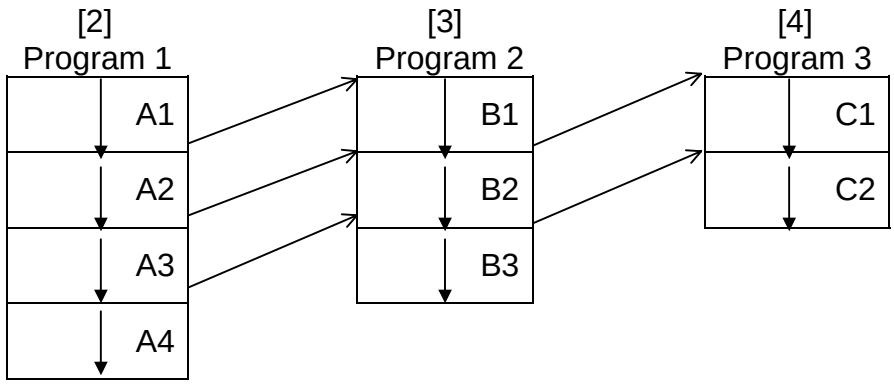


Fig. 2 Multitask operation

In Fig. 1, program A is executed continuously from the start to the end (single task operation and no subroutine call).

A program which uses no multitask command is executed in the manner as shown in Fig. 1 (no subroutine call).

Execution of a program which uses the multitask command is shown in Fig. 2.

As shown in Fig. 2, the multitask operation is realized, changing over a plural number of individual programs by time sharing, as if the programs were executed in parallel.

The order of program execution is shown in the following table.

Order	Program to be executed	
1	A1	Program 1 start
2	B1	Program 2 start
3	C1	Program 3 start
4	A2	
5	B2	
6	C2	1-cycle end of program 3
7	A3	
8	B3	1-cycle end of program 2
9	C1	Program 3 start
10	A4	1-cycle end of program 1
11	B1	Program 2 start
12	C2	
13	A1	Program 1 start
:	:	

Next, the start of multitask is described.
A program that can be treated as multitask is the program block containing no arguments. The program block means an area between the PROGRAM command and END command, which consists of the SCOL language statements. The subroutine without argument can be dealt with as a task. The argument cannot be kept in the task.
To deal with a program as task, use the TASK command. The TASK command executes a program specified in the argument as a task. Unless the program starts by the TASK command, the program is not performed as a task.
The program block (statements between the PROGRAM command and the END command) described at the head of the program file is an exception. Even if the TASK command is not used, the program is performed as a task.
To execute the program 2 as a task in the Fig. 2, the TASK (“PROG2”) is required to be executed in the program 1. (The program 1 is described at the head of the file, and the program starts as a task without TASK command.)
To execute the program 3 as a task, a new task (“PROG 3”) is required to be executed in the task (in the program 1 or 2 in this case) which has been already started.
If the task and program which have been started are reset or the task operation is released by the SCOL language, the task is kept active.

The task ID (the number assigned to the task) is described.
The characteristic numbers (task ID) are assigned to the tasks which have been started by the TASK command respectively. In the example of Fig. 2, “1” is assigned to the program 1, “2” is assigned to the program 2 and “3” is assigned to the program 3. This task ID starts from 1 in sequence and this ID increases one by one every time the task starts (every time the task executes). If the task is managed by the SCOL language, this task ID is used.
To get the task ID, see the following examples.

Example: I1 = TASK (“PROG 2”)

“I1” is a desired variable of integer type. The task ID of PROG 2 can be obtained. This command is executed in the program 1. The task ID of its own cannot be referred to in the program 2 in this example.

Example: I2 = TID

“I2” is a desired variable of integer type. If the system variable TID is referred to, the task ID of its own can be acquired. If this command is executed in the program 2, the task ID of its own can be seen in the program 2 (“2” in this occasion).

If this command is executed in the program 1, the task ID of program 1 (“1” in this occasion) is substituted for “I2”.
If the task ID other than the own task is referred to from other tasks, variables of examples 1 and 2 are required to be defined as the global variable.

Change-over of task is described.
As shown in the Fig. 2, the system executes the program 1 ~ 3 by time sharing. When this happens, timing of program change-over depends on the following three conditions.

(1) When the program change-over is specified clearly by the SWITCH command of the SCOL.
The SWITCH command is used if the task is changed over clearly by the SCOL language. Even if the task change-over conditions specified in the system are not satisfied while the SWITCH command is used, the task can be changed over.

(2) When a new task starts by the TASK command of the SCOL.
If a new task starts by the TASK command, the program control is changed over to the newly started task.

(3) When the task terminates by the KILL command of the SCOL.
If the task of its own terminates by the KILL command, the program control is changed over to the next task.

(4) When the predetermined conditions specified in the system are satisfied and the program is changed over by the system.

The task change-over conditions specified in the system are as follows:

(1) A program in a task is executed for more than 100 msec.

(2) When the data area for movement command becomes full.
Up to four data can be read beforehand by the movement command. If this internal area for prior reading becomes full, the task is changed over.

(3) When a command required for communication with the external device has been executed.
The INPUT, PRINT and RESTORE commands are not executed by the active SCOL program alone. They take time to execute the TP and RAM file operation by an operator. The active program waits for the reply of processing finish from the operator and changes over the task.
To cancel the task changeover by the system, specify DISABLE for system variable SWITCH.

Note: During step execution or when task changeover has been cancelled, only the program executed currently continues and the program of another task already started will not run. (The single task operation becomes effective.)

Cautions on creating a multitask program

(1) Motion commands can be used in the main task only. If they are used in the subtask, an error is generated.
The motion commands are MOVEI, MOVEA, MOVE, MOVES, MOVEC, MOVEJ and DELAY.

(2) As each task has system variables TIMER, TID and NOWAIT separately, they can be set arbitrarily in each task, or referred to.
For NOWAIT, separate or common use of it can be selected by means of the user parameter [U02].

It is recommended to use NOWAIT separately and set ENABLE NOWAIT in the subtask.
The following system variables are commonly used among the tasks. A value set in one task remains effective in another task. Careful precautions should be taken, therefore, when setting a plural number of tasks.

CONFIG, ACCUR, ACCEL, DECEL, SPEED, PASS, TORQUE, GAIN, TOOL, BASE, WORK, PAYLOAD, NOWAIT

(3) When a command for waiting for the finish of a movement as shown below has been executed while the motion command was currently executed in the main task, the subtask waits for the finish of the active motion command in the main task.

1) Execution of input/output commands in DISABLE NOWAIT mode
The input/output commands includes INPUT, PRINT, DIN, DOUT, BDCIN, BCDOUT and POUT.

2) Execution of WAIT MOTION command.

(4) If execution of an INPUT or PRINT command has been commanded in another task while the same command was executed in one task for different communication channel, the INPUT or PRINT command started first is executed to the last and the another task should wait until the command has been executed.

2.8.5 Global Variable Definition

If the global variable which can be referred to from the entire program is defined, obey the following rules.

(1) Global variable declaration

If the global variable is used, the type and identifier (variable name) of the variable to be used is required to be defined.
This definition must be performed before the first PROGRAM statement.
To define the variable A of real number type and the variable B of integer type, the definition is as follows:

```
GLOBAL
A = 1.0     (This value is the initial value of the variable.)
B = 2
END

PROGRAM
:
END
```

(2) Global variable declaration by type

To define the global variable of each type, use the following formats.

Integer type:	A = 1	
Real number type:	B = 1.0	
Array type:	DIM D(10) AS INT	Array of ten integer type elements is defined. (Note 1)
	DIM E(10, 3) AS REAL	Array of 10 × 3 real number type elements is defined.
	DIM F(5) AS POINT	Array of five position type elements is defined.

In the global block between reserved words GLOBAL and END, only variables of a scalar type and array type should be described, and the variables of a vector type should be described in the data block between reserved words DATA and END which are edited by the data editor.

(3) Setting of initial value of array type variable

Like the global variable other than the array, specify the initial value of the scara type array variable by the global block, and specify the initial value of vector type array variable by the data block.

Note 1: The initial value of the array type global variable, which is not set clearly, is indefinite. The variable is required to be initialized by the user program.

2.8.6 Array Type Variable

For the variables of a scalar type and vector type, each name represents one (1) data. If one (1) name can signify multiple data, however, programming becomes easier, and the array type variable can be used for this purpose.
To use the array type variable, the type and number of all elements should be predetermined by the DIM command. The array type variable bearing the same name cannot have a type with different elements.
For details, see the descriptions on the DIM command.
The variable which is declared as the array type by the DIM command should be described in the program according to the following format.

Variable name (<element> [. <element>] ···)

For the element number following the variable name, any value ranging from 1 to a number specified by the DIM command can be selected. However, if the total number of "variable name + "(" + element + ")" exceeds ten (10) characters, an execution error is generated.
When the SCOL program is executed, specific one (1) data is selected from multiple data based on the variable name and element number.
When 25 positions (5 × 5) are processed as the array variable named "P" and each axis is moved to respective positions in turn, the program is as follows:

```
GLOBAL
    DIM P(5, 5) AS POINT
END
PROGRAM SAMPLE
FOR I = 1 TO 5
    FOR J = 1 TO 5
        MOVE P(I, J)
```

```
    NEXT J
NEXT I
END
DATA
    POINT P(1,1) = 650, 0, 100, 0, 0 / LEFTY
    POINT P(1,2) = 650, 0, 100, 0, 0 / LEFTY
                  :
                  :
    POINT P(5,5) = 650, 0, 100, 0, 0 / LEFTY
END
```

Direct access to or substitution of elements X and Y of vector type array variable is not possible. In the example above, the following command cannot be executed for array type variable P as given above.

```
PRINT P(1,1).X, CR
```

When this happens, copy the value to the normal vector type variable, then execute the command.

```
PP=P(1,1)
PRINT PP.X
```

Also, if move to position P (I, J) is commanded by mistake while I = 5 and J = 6, an execution error is generated because access to or substitution of an element other than specified by the DIM command is not possible. The type of array index is only integer. If the data of a real number type and vector type is used, an error occurs at execution.

Section 3

Explanation of Robot Commands

Here we describe in detail what each SCOL command means and does. These commands are listed in alphabetical order.

3.1 Command Explanations

Commands are explained as follows.

(1) Purpose
This paragraph gives a simple explanation of what the command does.

(2) Format
This paragraph describes how to write down the command. The symbols used in the paragraph have the meaning that follows:

[ ]	Indicates that the content therein can be omitted. The commands are specified as necessary.
< >	Indicates the content of the data to be described.
{ }	Indicates that one among the data in this bracket should be selected.
...	Indicates that a plurality of data elements can be specified.

The above symbols are used here for purposes of explanation, and they are not actually written in the program. If these symbols need to be used in a particular case, this will be explained.

(3) Examples
This paragraph presents samples showing how to use the command.

(4) Analysis and advice
This paragraph presents an analysis of the command and describes warnings and restrictions for its use.

(5) Sample program

This paragraph presents a short program example using the command.

The meaning of the data in the command format is shown below. An expression can also be used as data.

<Position>	Specifies positional data.
<Axis>	Specifies a joint-controlled axis. The data must be an integer from 1 to 5.
<Absolute position>	Specifies the absolute position of each axis. Data is in units of 0.001 mm or 0.001 degrees. Variables or expressions may also be used as data.
<Relative position>	Specifies the travel of each axis in terms of relative position. Data is in units of 0.001 mm or 0.001 degrees. Variables or expressions may also be used as data.
<Time>	Specifies time in units of 0.01 seconds. Variables or expressions may also be used.
<Logical expression>	Specifies a logical expression.
<Statement>	Specifies a statement to be executed. As long as it is a normal SCOL statement, you can specify anything you want.
<Monitoring condition>	With an ON statement, specifies condition(s) to be monitored.
<Label>	Specifies a label for branching the program.
<Comment>	Shows comments written in the program.
<Variable>	Indicates a variable.
<Expression>	Indicates a calculation. Individual variables may also be substituted for a calculation.
<Signal name>	Specifies the name of an I/O (Input/Output) signal. The signal name is to be given as an integer. A positive value shows that the signal is ON and a negative value shows that the signal is OFF. Variables and expressions may also be used.

<Mass>	Specifies the mass of the load acting on the robot hand.
<Center of gravity offset>	Specifies the distance between the center of gravity of the load applied to the tip of robot hand and the center of the tool of the hand.
<Configuration>	Specifies the robot configuration with an integer. "0" means the configuration is undefined (not fixed), "1" specifies a left hand configuration, and "2" specifies a right hand configuration.
<Switch>	Specifies the system switch. There are two system switches available.
PASS	This system switch specifies shortcut movement.
NOWAIT	This system switch directs signal I/o to be performed without waiting for the completion of a previous movement command.
<State>	Specifies what is to be reset by the RESET command.

Table 3.1 presents a list of commands classified by purpose.

Table 3.1 SCOL commands

Type	Command	Purpose
Movement control commands	BREAK	Suspends movement immediately.
	CLOSE1, CLOSE2	Closes hand after completion of movement.
	CLOSEI1	Closes hand.
	CLOSEI2	Closes hand.
	DELAY	Pauses for specified time.
	MOVE	Synchronous movement
	MOVES	Linear interpolation movement
	MOVEC	Circular interpolation movement
	MOVEA	Absolute single axis movement
	MOVEI	Relative single axis movement
	MOVEJ	Arch movement
	OPEN1, OPEN2	Opens hand after completion of movement.
	OPENI1, OPENI2	Opens hand.
	PAUSE	Suspends a movement.
	READY	Moves to machine coordinate origin.
	RESUME	Restarts an interrupted movement.
Program control commands	FOR ~ TO ~ STEP ~	Repeats movement.
	GOTO	Branches unconditionally.
	GOTO ()	Branches in accordance with the value of an expression
	IGNORE	Cancels monitoring.
	IF ~ THEN ~ ELSE ~	Judges conditions.
	NEXT	Repeats movement.
	ON ~ DO ~	Registers conditions monitor.
	PROGRAM	Marks beginning of program.
	RCYCLE	Label for cycle reset
	RETURN	Returns to main program.
	STOP	Stops the program.
	WAIT	Waits for establishment of conditions.
	END	End of program
	KILL	Task standstill
	MAXTASK	Maximum number of tasks
	REMARK	Comments
	SWITCH	Task change-over
	TASK	Task start
	TID	Task ID

Type	Command	Purpose
I/O control commands	BCDIN	Inputs a BCD signal.
	BCDOUT	Outputs a BCD signal.
	CR	Outputs a CR code
	DIN	Reads an input signal.
	DOUT	Outputs a signal.
	HEXIN	Reads signals in hexadecimal notation.
	HEXOUT	Outputs signals in hexadecimal notation.
	PULOUT	Outputs a pulse signal.
	RESET	Resets the controller.
	PRINT	Outputs communication data.
	INPUT	Inputs communication data.
Movement condition commands	ACCEL	Specifies acceleration (during acceleration).
	ACCUR	Specifies positioning accuracy.
	CONFIG	Specifies configuration.
	DECEL	Specifies acceleration (during deceleration).
	DISABLE	System switch off
	ENABLE	System switch on
	FREELOAD	Cancels load data.
	GAIN	Each axis gain
	NOWAIT	Does not wait for the completion of positioning for previous movement.
	PASS	Short-cut movement parameter
	PAYLOAD	Sets load data.
	SMOOTH (option)	Smooth movement
	SPEED	Specifies speed.
	MOVESYNC	Specifies movement command synchronization/unsynchronization mode
	SWITCH	Prohibits or allows task change-over.
	WITH	Specifies operating conditions.

Type	Command	Purpose
Calculator commands	COS	Cosine
	SIN	Sine
	TAN	Tangent
	ABS	Absolute value
	ACOS	Arccosine
	AND	Logical product
	ASIN	Arcsine
	ATAN	Arctangent
	ATAN2	Arctangent
	DEST	Destination position
	EXP	Exponent to power e
	HERE	Present position
	INT	Changes number to an integer.
	LN	Natural logarithm
	LOG10	Common logarithm
	MOD	Remainder
	NOT	Negation
	OR	Logical sum
	POINT	Creates positional type data.
	REAL	Changes number to a real number.
	SGN	Extracts and returns the sign.
	SQRT	Square root
	TRANS	Creates coordinate type data.
Movement reference commands	BASE	Base coordinate system
	MODE	System operating mode
	MOTION	Amount of movement which has been executed
	MOTIONT	Time expended for a motion
	REMAIN	Amount of movement remaining to be executed
	REMAINT	Time remaining for a motion
	TIMER	Timer
	TOOL	Tool coordinate system
	WORK	Work coordinate system

Type	Command	Purpose
Data definition commands	DATA	Starts data definition.
	DIM ~ AS	Array variable definition
	GLOBAL	Global variable definition
	RESTORE	Saves an initial value of the global variable to a file.
	SAVEEND	Saves data at power OFF.
Palletize command	INITPLT	Initializes a pallet.
	MOVEPLT	Moves to pallet specified position.
Positional data latch function (Options of TS3000)	LATCH	Position latch function ON/OFF
	LATCHTRG 1 ~ 8	Detected edge direction
	LATCHSIG 1 ~ 8	Signal state
	LATCHPSN 1 ~ 8	Latched position
System constants	COARSE	Coarse positioning accuracy
	COM0, TP	Communication channel (teach pendant)
	COM1	Communication channel 1
	CONT	Continuous operation mode
	CYCLE	Cycle operation mode
	FINE	Fine positioning accuracy
	OFF	Each axis gain OFF
	ON	Each axis gain ON
	PAI	Pi
	SEGMENT	Segment operation mode
Simplified PLC	PLCDATAR 1 ~ 8	Simplified PLC interface
	PLCDATAW 1 ~ 8	Simplified PLC interface

Type	Command	Purpose
Mathemati-cal symbols	^	Exponentiation
	–	Negative sign
	*, /	Multiplication and division
	+, –	Addition and subtraction
	=	Substitution
	= =	Equal
	< >, > <	Not equal
	<	Less than
	>	Greater than
	< =, = <	Less than or equal
	> =, = >	Greater than or equal
	‘	Comments
	.	Designation of vector element

3.2 Explanation of Commands

SCOL commands are explained in the following pages. Commands are arranged in alphabetical order.

ABS

Purpose

The ABS function will return the absolute value of a number.

Format

ABS(<expression>)

Examples

```
AK = ABS(–20.345)
K  = ABS(K1)
J1 = K – ABS(N – 28.5)
```

Analysis and advice

This function returns the absolute value of the <expression>.

You may use a constant, variable or result of calculation for the <expression> term. However, you may not use vector data.

This command must be used in an expression.

Sample program

```
PROGRAM MAIN
    ABSSAMPLE (3, 5, K)
    PRINT TP, K, CR
END
PROGRAM ABSSAMPLE(K1, K2, K)
    K = ABS(K1 – K2)
    RETURN
END
```

This program takes the arguments K1 and K2, subtracts K2 from K1, finds the absolute value of the result, calls it K, and sends program execution back to the main program.

ACCEL

Purpose

This command sets the fraction of full acceleration for the robot.

Format

ACCEL = (<expression>)

Examples

ACCEL = 80
ACCEL = 0. 8*ACCEL
MOVE A1 WITH ACCEL = 90

Analysis and advice

ACCEL is a system variable used to specify the acceleration of the robot when the robot is accelerating.
Acceleration is expressed as a percentage of the standard (full) acceleration. In the SCOL language, acceleration during acceleration and acceleration during deceleration are set separately. In order to set the acceleration during deceleration, use the DECEL commands.

This is used to lower the acceleration as needed when the robot is carrying a heavy load. In this case, change the acceleration during both acceleration and deceleration. For the setting value of the acceleration according to the load, see the "Transportation and Installation Manual."

You may use a constant, a variable or a calculation for the <expression>. However, you may not use vector-type data.

This command must be used in an expression.
An upper limit on acceleration is set in the controller to protect the robot. The robot will not exceed this limit even if you enter a value larger than the upper limit.

Values of 0 or less are taken as 1.

The current acceleration during acceleration can be viewed by viewing this system variable.

This initial value for acceleration is 100%.

Sample program

```
PROGRAM ACCELSAMPL
    FOR K = 1 TO 100
        ACCEL = K
        DECEL = K
        MOVE A1
        MOVE A2
    NEXT K
END
```

This program increases the acceleration from 1% to 100% in steps of 1%.

ACCUR

Purpose

This command specifies the positioning accuracy of the robot.

Format

ACCUR = (<expression>)

Examples

```
ACCUR = 1
N = ACCUR
MOVE A1 WITH ACCUR = COARSE
```

Analysis and advice

ACCUR is a system variable used to specify the positioning accuracy of the robot. A coarse positioning accuracy is marked by 0 and a fine positioning accuracy is marked by 1.

When the positioning accuracy is set to coarse, the robot executes the subsequent command before the positioning of the robot is completed. The robot tact time can be reduced by setting the positioning accuracy to COARSE for operations where waiting for high positioning accuracy is unnecessary.
The system constants FINE and COARSE can be used to specify the positioning accuracy. The positioning accuracy is set to fine by ACCUR=FINE and set to coarse by ACCUR=COARSE.
You may use a constant, a variable, or a calculation for the <expression>. However, you may not use vector-type data.

The ACCUR command must be used in an expression.

When specifying the positioning accuracy, anything entered that is less than 0 will be taken as 0 and anything greater than 1 will be taken as 1.

You can find the positioning accuracy under which the system currently is operating by referring to ACCUR. An ACCUR value of 0 means coarse, and a value of 1 mean fine.

The initial setting for the positioning accuracy is FINE.

Sample program

```
PROGRAM ACCURSMPL
    ACCUR = COARSE
    MOVE A1
    MOVE A2
    MOVE A3   WITH ACCUR = FINE
    MOVE A4
END
```

The robot will move with fine positioning accuracy only for movement to point A3. It will move with coarse positioning accuracy for other movements increasing cycle time.

ACOS

Purpose

This function returns the arccosine of an entered value.

Format

ACOS(<expression>)

Examples

K = ACOS (0.577)
J1 = 90 – ACOS (X/L)

Analysis and advice

This function returns the arccosine of the value in the brackets ( ). The returned value is in units of degrees.

You may enter a constant, variable or calculation for the <expression>. However, you may not enter vector-type data.

This command must be used in an expression.

Sample program

```
PROGRAM MAIN
    ACOS2 (1.0, 2.0, K)
    PRINT TP, K, CR
END
PROGRAM ACOSSAMP (L, X, K)
    K = ACOS (X/L)
    RETURN
END
```

This program takes the arguments L and X, divides X by L, takes the arccosine of the result, calls it K, and returns to the main program.

AND

Purpose	AND calculates the logical product of expressions.
Format	<Logical expression> AND <Logical expression>
Examples	IF DIN (1) AND K < = 3 THEN J = 0 WAIT DIN (5) AND TIMER==0
Analysis and advice	The AND statement is used to find the logical product of two logical expressions. If both logical expressions are true, a TRUE value will be returned. This command must be used in a logical expression.
Sample program	PROGRAM ANDSAMPLE FOR K=1 TO 50 IF K==50 AND DIN (1) THEN J=1 ELSE J=0 PRINT TP, J, CR NEXT K END

ASIN

Purpose

This function finds the arcsine of the value entered.

Format

ASIN ((expression>)

Examples

K = ASIN (0.577)
J1 = 90 – ASIN (Y/L)

Analysis and advice

This function returns the arcsine of the value in the brackets ( ). The returned value is in units of degrees.

You may enter a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data.

This command must be used in an equation.

Sample program

```
PROGRAM MAIN
    ASIN2 (5.0, 2.0, K)
    PRINT TP, K, CR
END
PROGRAM ASIN2 (L, Y, K)
    K = ASIN (Y/L)
    RETURN
END
```

This program takes the arguments L and Y, divides Y by L, takes the arcsine of the result, calls it K, and returns to the main program.

ATAN/ATAN2

Purpose

This function returns the arctangent for the value(s) entered.

Format

ATAN (<expression>)
ATAN2 (<expression>, <expression>)

Examples

K = ATAN (0.577)
J1 = 90 – ATAN (Y/X)
N = ATAN2 (0.3, 0.5)
D = ATAN2 (100/K, 50/J)
L3 = ABS (180 – ATAN2 (A1, Y, A1, X))

Analysis and advice

This function returns the arctangent of the value(s) in the brackets ( ).
The ATAN2 takes the first expression in the brackets, divides it by the second expression in the brackets, and finds the arctangent of the result.

For both functions, the returned value is in units of degrees. (Be warned that ATAN2 (0, 0) will return a 0 instead of an error.)

You may enter a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data. This command must be used in an equation.

Sample program

```
PROGRAM MAIN
    ATANSAMPLE (5.0, 3.0, K)
    PRINT TP, K, CR
END
PROGRAM ATANSAMPLE (X, Y, K)
    K = ATAN (Y/X)
    RETURN
END
```

This program takes the arguments X and Y, divides Y by X, takes the arctangent of the result, calls it K, and returns to the main program.

```
PROGRAM MAIN
   ATAN2SMPL (K)
   PRINT TP, K, CR
END
PROGRAM ATAN2SMPL (K)
   K = ATAN2 (A1. Y, A1. X)
   RETURN
END
```

This program takes the elements X and Y from taught point A1, divides Y by X, takes the arctangent of the result, calls it K, and returns to the main program.

BASE

Purpose

BASE is a system variable used to specify the base coordinate system.

Format

BASE

Examples

```
BASE = TRANS (0, 0, 0, 0)
BASE1 = BASE
MOVE A1 WITH BASE = BASE + TRANS (,, 100)
```

Analysis and advice

BASE is a system variable used to specify the base coordinate system. It can be handled as normal coordinate-type data. By referring to the BASE, you can find the values (location) of the present base coordinate system.

You can directly designate values for the base coordinates with one of the following two methods:

```
BASE = TRANS (X, Y, Z, C)
BASE = {X, Y, Z, C}
```

In order to make it clear just what kind of data type you are using, always try to use the TRANS command.

The BASE variable has the following format:

X, Y, Z, C: X, Y, Z and C are real numbers representing the position of the base coordinate system. Units are of millimeters or degrees.

The BASE coordinate system is created by "sliding" a distance of X, Y and Z along the respective axes of the WORLD coordinate system and then twisting the new Z axis by an amount C.

BASE must be used in an expression.

Be aware that if you change base coordinate systems within a program, there may be some misalignment between the positions as taught and the positions where the robot actually moves.

Sample program

```
PROGRAM BASESAMPLE
    MOVE A1
    MOVE A2
    BASE = BASE + TRANS (,, 200)
    MOVE A1
    MOVE A2
    BASE = TRANS ()
END
```

The BASE command shifts the BASE coordinate system 200 mm along the Z axis and after that, the robot moves to a point below 200 mm from the taught position.

BCDIN

Purpose

This command is used to read in signals as BCD (Binary Coded Decimal) notation.

Format

BCDIN (<signal name>, <signal length>)

Examples

K = BCDIN (1, 2)
J2 = BCDIN (N, N + 2)
GOTO (BCDIN (20, 1)) L1, L2, L3

Analysis and advice

The command causes an input signal to be read in as BCD notation. The signal will start from the signal name and continue to a place 4 times the value of the specified signal length. For example, "K = BCDIN (1, 2)" tells the controller to read in, as BCD notation, an eight unit signal starting from the 1st unit (bit) and continuing to the 8th unit (bit).

Higher signal numbers correspond to bits with a higher digit.

Input signals are divided into units of 4 bits (in order of low to high place value (signal number)) and converted into Base 10.

Signals are coded as 1 for ON and 0 for OFF.

Example:
In the case where input signals 1 to 12 indicate the state shown below, the value of BCDIN(1,3) is 329.

Input signal number	12	11	10	9	8	7	6	5	4	3	2	1
Input signal state	OFF	OFF	ON	ON	OFF	OFF	ON	OFF	ON	OFF	OFF	ON
Base 2 expression	0	0	1	1	0	0	1	0	1	0	0	1
Base 10 exp0ression				3				2				9

You may use constants, variables, expressions or calculations for <signal name> and <signal length>. However, you may not use vector-type data.

This command must be used in an equation.

When an input signal is coded based on 4 bits and if it exceeds 9 (1001 in binary notation), the signal is coded in the binary notation. In other words, if signal “1111” is input in the binary notation, it is read as “15”.

Sample program

```
PROGRAM BCDINSAMPL
    K = BCDIN(l, 2)
    SPEED = K
    MOVE A1
    MOVE A2
END
```

This program reads in a single made up of eight bits, sets the movement speed in accordance with that signal.

BCDOUT

Purpose	This command will put a signal into BCD notation and output the result.
Format	BCDOUT(<signal name>, <signal length>, <expression>)
Examples	BCDOUT(1, 4, 3) BCDOUT(N, N+4, K)
Analysis and advice	This command will take the value of the expression and change it into a BCD signal having the name <signal name> and a signal length four times the value of <signal length>. For example, the command BCDOUT(1, 1, 3) will produce an output signal which is four bits long and starts from signal position 1. The output signal will express the value 3 as a binary number. Therefore, the first two bits (output signal numbers 1 and 2) will be ON. Higher signal numbers correspond to bits with a higher digit. Each digit of <expression> is broken down one at a time and converted into a 4 bit code. The 4 bit code itself is built up in order of smaller to larger output signal number. Should there not be enough room to hold all the bit code corresponding to the Base 10 <expression> (i.e., should the signal length be too short), the excess digits of the Base 10 <expression> will be ignored. Signals are considered as 1 when ON and 0 when OFF.

Example:
The command BCDOUT(1, 3, 952) will create an output signal like that shown below.

Base 10 expression				9				5				2
Base 2 expression	1	0	0	1	0	1	0	1	0	0	1	0
Output signal number	12	11	10	9	8	7	6	5	4	3	2	1
Output signal state	ON	OFF	OFF	ON	OFF	ON	OFF	ON	OFF	OFF	ON	OFF

You may use constants, variables, expressions or calculations for <signal name>, <signal length> and <expression>.
However, you may not use vector-type data.
If the same signals are output consecutively, the signal output last becomes valid.
Be careful of agreement's there being in the signal range by which it is possible to guarantee simultaneous-ness.
The simultaneous-ness of BCDOUT to the range of the 16 bit carving from DOUT1, DOUT101, DOUT301 can be guaranteed but the simultaneous-ness of BCDOUT which strides the boundary can not be guaranteed.

Sample program

```
PROGRAM BCDOUTSAMPL
    J = 0, 0
    FOR K = 1 TO 4
    J = 2 ^ (K = 1)
    BCDOUT (1, 1, J)
    TIMER = 0.5
    WAIT TIMER = = 0
    BCDOUT (1, 1, 0)
    NEXT K
END
```

This program will output a number from 1 to 4 in steps of 1 at intervals of 0.5 seconds.

BREAK

Purpose

This command immediately suspends robot operation.

Format

ON <monitoring condition> [{BREAK | PAUSE}] DO <statement>

Examples

ON DIN(l) BREAK DO SUB

Analysis and advice

The BREAK command will stop robot movement immediately when the specified monitoring condition has been satisfied, and execute the statement following the DO statement. The robot will decelerate and stop the movement.
For more information, see the ON command.

The RESUME command can be used to restart operation interrupted by the BREAK command.
This command allows you to stop the robot and take appropriate reaction should any problems occur with the system.

Sample program

```
PROGRAM BREAKSMPL
    REMARK *** MAIN PROGRAM ***
    ON DIN(24) BREAK DO BREAKSUB
    MOVE A1
    MOVE A2
    MOVE A3
    WAIT MOTION >= 100
    IGNORE DIN(24)
END
```

Should something go wrong with the system and Input Signal 24 turn ON, the robot will stop immediately and control will shift to BREAKSUB, a subroutine designed specifically for such a case.

```
PROGRAM BREAKSUB
    REMARK *** SUBROUTINE ***
    WAIT DIN(–24)
    RESUME
END
```

The subprogram BREAKSUB will sit and wait until Input Signal 24 turns OFF.

After the problem has been removed, the program resumes the movement.

CLOSE1, CLOSE2, CLOSEI1, CLOSEI2

Purpose

These commands close the robot hand.

Format

CLOSE1
CLOSE2
CLOSEI1
CLOSEI2

Examples

CLOSE1
CLOSEI2

Analysis and advice

These commands are used to close the hand. The numbers 1 and 2 refer to Hand1 and Hand2. These commands close the hand by changing the state of the output signal which controls the robot hand.

The CLOSE command directs the robot to close its hand after it completes the motion in progress.

The CLOSEI command directs the robot to close its hand immediately.

Note that these commands will not work if the file SCOL.LIB is not in the controller RAM drive.

Also, keep in mind that there is a slight delay from when a CLOSE command is executed until the robot actually closes its hand.
Corresponding commands OPEN1, OPEN2, OPENI1 and OPENI2 are provided in order to open the hand.

These commands execute a program written in the system library (SCOL. LIB). The data of SCOL. LIB should be changed according to the robot hand specifications.

Sample program

```
PROGRAM CLOSESMPL
    OPENI1
    MOVE A1
    CLOSE1
    DELAY 0.5
    MOVE A2
END
```

This program closes the hand1 after the robot has finished moving to point A1. The robot waits 0.5 seconds until the hand is closed completely after the CLOSE command has been executed.

```
PROGRAM CLOSEISMPL
    ENABLE NOWAIT
    OPENI1
    DELAY 0.5
    MOVE A1
    CLOSET 1
    DELAY 0.5
    MOVE A2
END
```

Here, the robot will close the hand1 while moving to point A1.

COARSE

Purpose

COARSE is a system constant used to set the positioning accuracy to coarse.

Format

COARSE

Examples

```
ACCUR = COARSE
MOVE A1 WITH ACCUR = COARSE
```

Analysis and advice

The COARSE statement sets positioning accuracy to COARSE.

As a system constant, COARSE has a value of 0. If you wanted to, you could use it in your program as a constant having the value 0. However, you should not do things like that since it makes your program extremely hard to read.

You cannot substitute into system constants such as COARSE.

For information on positioning accuracy, refer to the ACCUR command.

Sample program

```
PROGRAM COARSESMPL
    MOVE A1
    ACCUR = COARSE
    MOVE A2
    MOVE A3
END
```

This program sets the positioning accuracy to coarse before moving around the robot.

COM0, COM1

Purpose

These commands specify the communication channel to be taken by a PRINT or INPUT command.

Format

PRINT [{COM0 | COM1 | TP},]
{<character string> | <expression>}][<character string>|
<expression>]]...[, CR]
INPUT [(COM0 | COM1 | TP},]
<variable> [, <variable>] ...

Examples

```
PRINT COM0, "*** INPUT N ***"
PRINT COM1, N, N * 10
INPUT COM1, K
```

Analysis and advice

COM statements are used to designate a communications channel when using a PRINT or INPUT command.

COM0 is a communications channel used solely for the teach pendant.

COM1 corresponds to the communication channel of controller connector COM1.

If you do not specify a communication channel for a PRINT or INPUT command, the controller inputs or outputs data to or from the communication channel used solely for the teach pendant.

See the PRINT and INPUT commands for communication processing.

Sample program

```
PROGRAM COMSAMPLE
    PRINT COM0, "*** INPUT N ***"
    INPUT COM0.N
    PRINT COM1.N, CR
END
```

This program inputs a value from the teach pendant and sends it out on the No. 1 communications channel.

CONFIG

Purpose

This command is used to specify the robot configuration.

Format

CONFIG = <expression>

Examples

CONFIG = 1
MOVE A1 WITH CONFIG = RIGHTY

Analysis and advice

CONFIG is a system variable used to express the configuration of the robot. You should specify the robot configuration when there is a chance of peripheral equipment interfering with the robot motion.

The robot configuration is undefined at 0, left handed at 1 and right handed at 2. In order to specify the system configuration, you may use these numbers or the system constants FREE, LEFTY and RIGHTY.

As you would probably guess, the configuration is undefined at CONFIG = FREE, left handed at CONFIG = LEFTY, and right handed at CONFIG = RIGHTY.

The configuration is included in positional data fed into the robot while teaching. Therefore, when CONFIG = FREE, the robot will move with the same configuration it had when it was being taught.

Unless there is a good reason otherwise, you should leave the configuration undefined, i.e. CONFIG = FREE. This is the initial value that the robot will assume.

The robot configuration may change upon executing a movement command.

When conducting linear or circular interpolation (with the MOVES or MOVEC command), the robot configuration cannot be changed and an error will result if you try.

Designation of the configuration for an orthogonal coordinate robot is ignored.

A constant, a variable or a calculation can be used for the <expression> term. However, you may not use vector-type data.
Even if it refers to this system variable, the posture of the present robot can not be referred to.
It is possible for the posture of the present robot to be acquired by the HERE direction.
N=HERE.6
The value of the present posture is stored in N.
For SCARA type robots, the value of HERE.6 is undefined at 0, left handed at 1, and right handed at 2. When the starting point position of the robot isn't correctly set, it works in the position where a robot was shifted with the position to have instructed in when changing the posture of the robot.
Therefore, when instructing a robot in the position, the robot go actually in the working posture.

Sample program

```
PROGRAM CONFIGSMPL
    CONFIG = RIGHTY
    MOVE A1
    MOVE A2
    MOVE A3 WITH CONFIG = LEFTY
    MOVE A4
END
```

The robot moves with a left hand configuration only when moving to A3, and moves with a right hand configuration for other movements.

CONT

Purpose

CONT is a system constant which is used to refer to the system operating mode.

Format

CONT

Examples

IF MODE < > CONT THEN STOP

Analysis and advice

CONT is used along with the MODE command to refer to the system operating mode. When MODE = = CONT, the system is operating in the continuous operation mode.

As a system constant, CONT has a value of 0. If you wanted to, you could use it in your program as a constant having the value 0.
However, you should not do it since it will make your program hard to understand.

You cannot substitute into system constants.

For information on operating modes, refer to the MODE command.

Sample program

```
PROGRAM CONTSAMPLE
    IF MODE <>CONT THEN STOP
    MOVE A1
    MOVE A2
    MOVE A3
END
```

If the system changes out of the continuous operation mode, program execution will stop and the robot will not move.

COS

Purpose

This function returns the cosine of an entered value.

Format

COS (<expression>)

Examples

K = COS(60)
J1 = 1 – COS(180 – D)

Analysis and advice

This function returns the cosine of the value in the brackets ( ). Calculations are handled in units of degrees.

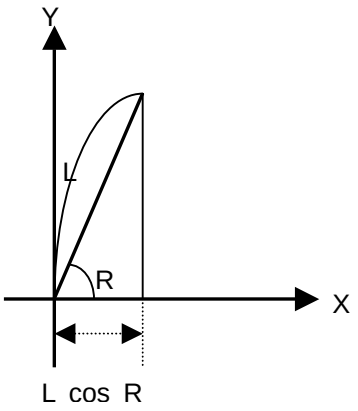
You may enter a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data.

This command must be used in an equation.

Sample program

```
PROGRAM MAIN
  COSSAMPLE (2, 30, K)
  PRINT TP, X, CR
END
PROGRAM COSSAMPLE (L, R, X)
  LOOP:
    IF R > 180 THEN R = R – 360
    IF R < –180 THEN R = R + 360
    IF R > 180 OR R < –180 THEN GOTO LOOP
    X = L * COS(R)
    RETURN
END
```

Given (as arguments) a line segment with a length L and forming an angle R with the X-axis, this program finds the length of the x-component of the line segment and sends it back to the main program as argument X.



CR

Purpose	This function outputs a CR (carriage return) code to the communication channel.
Format	PRINT [{COM0 | COM1 | TP},] {<character string> | <expression>} [, {<character string> | <expression>}] ··· [, CR]
Examples	PRINT COM1, K, CR PRINT TP, CR
Analysis and advice	This function outputs a CR (carriage return) code to the communication channel. For the communication channel, select only one (1) from COM0, COM1, and TP. COM0 and TP are the communication channels exclusively used for the teach pendant. COM1 corresponds to the COM1 communication channel of the controller. Unless the communication channel is specified by the PRINT command, data is output to the communication channel exclusive to the teach pendant. When CR is specified at the end of the PRINT command, a CR code (0DH) is attached to the last of the data. When the data has been output to COM0 (TP), it is displayed by line feed.

Sample program

```
PROGRAM COMSAMPLE
    PRINT TP, "*** INPUT N ***", CR
    INPUT TP, N
```

The characters are displayed on the teach pendant, which are then subject to line-feed.

```
    PRINT TP, N, CR
END
```

The value of "N" is displayed on the teach pendant, which is then subject to line-feed.

CYCLE

Purpose

CYCLE is a system constant which is used to refer to the system operating mode

Format

CYCLE

Examples

IF MODE < > CYCLE THEN GOTO LOOP

Analysis and advice

CYCLE is used along with the MODE command to refer to the system operating mode. When MODE = = CYCLE, the system is operating in the cycle operation mode.

As a system constant, CYCLE has a value of 1. If you wanted to, you could use it in your program as a constant having the value 1. However, this is not a good idea since it makes your program unnecessarily hard to understand.

You cannot substitute into system constants such as CYCLE.

For information on operating modes, refer to the MODE command.

Sample program

```
PROGRAM CYCLESMPL
  LOOP:
    MOVE A1
    MOVE A2
    MOVE A3
    IF MODE <> CYCLE THEN GOTO LOOP
END
```

If the system is not operating in the cycle operation mode, execution of the program will keep on returning to the beginning of the loop and the robot will move over and over again.

.

DATA

Purpose

This function designates the start of a data block which defines the position and coordinate data for the taught point. For details of the data block, see Para. 5.3.5.

Format

DATA

Examples

DATA

Analysis and advice

The data block is edited by the data editor rather than the program editor. If a format error occurs, the program editor is used for editing.

Sample program

```
PROGRAM MAIN
  MOVE HOME
END

DATA
  POINT HOME = 650, 0, 100, 0, 0 /RIGHTY
END
```

DECEL

Purpose

This command sets the fraction of full deceleration for the robot.

Format

DECEL = <expression>

Examples

DECEL = 80
DECEL = 0.8 * DECEL
MOVE A1 WITH DECEL = 90

Analysis and advice

DECEL is a system variable used to specify the acceleration of the robot when the robot is decelerating. Acceleration is expressed as a percentage of the standard (full) acceleration.

In the SCOL language, acceleration during acceleration and acceleration during deceleration are set separately. In order to set the acceleration during acceleration, use the ACCEL command.

This is used to lower the acceleration as needed when the robot is carrying a heavy load. In this case, change the acceleration during both acceleration and deceleration. For the setting value of the acceleration according to the load, see the "Transportation and Installation Manual."

You may use a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data.

This command must be used in an equation.

An upper limit on deceleration is built into the controller to protect the robot. The robot will not go over this limit even if you enter a value larger than the upper limit. Should you enter such a value, the robot will operate at the upper limit.

Values of 0 or less are taken as 1.

The current acceleration during acceleration can be viewed by viewing the system variables.
The initial value for acceleration is 100%.

Sample program

```
PROGRAM DECELSMPL
  FOR K = 1 TO 100
    DECEL = K
      MOVE A1
      MOVE A2
  NEXT K
END
```

This program increases the acceleration from 1% to 100% in steps of 1 %.

DELAY

Purpose

The DELAY command stops the movement of the robot arm for a specified time.

Format

DELAY <time>

Examples

DELAY 0.5
DELAY T * 0.2

Analysis and advice

The DELAY command stops the movement of the robot arm for a specified length of time.

The <time> designation is specified in units of seconds.

Since the execution precision is limited, try to keep your time designation in units of 0.01 second or more. A constant, variable or calculation may be used for the <time> designation. However, you may not use vector-type data.

When the program stop operation is conducted while the DELAY command is being executed, after the specified time elapsed, the automatic operation is stopped. On the other hand, when the automatic operation is cancelled with the servo off operation or the emergency stop operation while the DELAY command is being executed, the DELAY command is executed again when the program is restarted.

DELAY is a movement control command in that it stops the movement of the robot arm for a specified period of time. Keep in mind that the DELAY command stops movement of the robot and does not stop execution of the program itself.
When you want to delay the program itself, use the TIMER or WAIT command.

Sample program

```
PROGRAM DELAYSMPL
    MOVE A1
    DELAY 2
    MOVE A2
    DELAY 2
    MOVE A3
    DELAY 2
END
```

The robot will stop moving for 2 seconds after it completes each move.

DEST

Purpose

The DEST command returns the destination of the present robot command.

Format

DEST

Examples

A1 = DEST
X = DEST. X

Analysis and

The DEST command is used to refer to the destination of the movement being executed on the world coordinate system at that time.

DEST can be used just like any other positional vector-type. However, you can only refer to the values it contains and cannot change the values themselves.

Should the robot have come to rest after having positioned itself, DEST will return the location of that position.

Sample program

```
PROGRAM DESTSAMPLE
    AA = A
    MOVE A
    ON DIN(1) DO AA = DEST
    MOVE A1
    MOVE A2
    MOVE A3
    MOVE A4
    IGNORE DIN(1)
    PRINT "POSITION DATA = ", AA.X, AA.Y, AA.Z
END
```

This program moves the robot from point A1 to point A4 while monitoring Input Signal 1. Should the signal turn on, the target position (at that time) will be displayed on the teach pendant.

DIM …. AS ….

Purpose	This function defines an array variable.
Format	DIM <array variable> (<number of elements>, <number of elements>, …) AS <type>
Examples	DIM A (5) AS INT
Analysis and advice	This command is used to define the type and number of elements of array variable. The array variable can be defined only as the global variable which can be accessed and modified from any position of the defined program. The value of index of the array is 1 ~ No. of elements. That is, in this example, it is 1 ~ 5. If the initial value set is outside the index range, an error occurs at the execution of SELECT command. Also, access and substitution outside the index range cause an error at program execution. A total of five (5) types can be specified; INT (integer type), REAL (real number type), POINT (position type), TRANS (coordinate type) and PAYLOAD (load type). The initial value of INT and REAL type array variables should be described in the global block, and the initial value of POINT, TRANS and PAYLOAD type array variables in the data block.

Sample program

```
GLOBAL
    DIM ICHI (3) AS POINT      One dimensional array of position
                               type is declared.
END

PROGRAM DIMSAMPLE
    ICHI (1) = P0
    ICHI (2) = POINT (500.0, 0.0, 100.0, 0.0, 0.0, 0)
    ICHI (3) = {500.0, –200.0, 100.0, 0.0, 0.0, 0}
    FOR I = 1 TO 3
       MOVE ICHI (I)
    NEXT I
END
     DATA
     POINT P0 = 500.0, 200.0, 100.0, 0.0, 0.0 / LEFTY
     POINT ICHI (I) = 650.0, 0.0, 100.0, 0.0, 0.0 / LEFTY
END
```

DIN

Purpose

The DIN command reads in the state (ON or OFF) of an input signal (or signals).

Format

DIN(<signal name> [,<signal name>]...)

Examples

IF DIN(1) THEN GOTO LOOP
WAIT DIN (1, –2, 3)
ON DIN (J, J + 1, J + 2) DO RETURN

Analysis and advice

The DIN command reads in the state (ON or OFF) of an input symbol. DIN is used in conjunction with an IF, WAIT or ON command to judge external signals.

<signal name> specifies the signal number of a signal to be read into the controller. A positive signal is considered to be ON and a negative signal is considered to be OFF. Up to 10 signal names can be specified. (Extra signal names exceeding 10 signal names are ignored.)

When the state of all the signals becomes as specified, DIN will return a value of TRUE (1). If even one of the signals is not as specified, DIN will return FALSE (0).

A constant, variable or calculation may be used for the <signal name> specification. However, you may not use vector type data.

Sample program

```
PROGRAM DINSAMPLE
    WAIT DIN(1)
    MOVE A1
    MOVE A2
    MOVE A3
END
```

The robot will wait until Input Signal 1 turns on before starting to move.

DISABLE

Purpose

The DISABLE command is used to disable system switches.

Format

DISABLE <switch> [, <switch>]...

Examples

DISABLE PASS
DISABLE PASS, NOWAIT

Analysis and advice

The DISABLE command is used to disable system switches related to robot movement. There are three system switches.

(1) PASS

PASS is used to specify short-cut movement. Short-cut movement is an operating mode in which the robot is directed to begin its next move before completing the positioning of its previous move. The timing for switching over from the present movement to the next movement is specified with the system variable PASS command.

Short-cut movement allows you to reduce the time it takes the robot to get from one place to another. For more information, refer to Section 5.

A DISABLE PASS statement will cancel short-cut movement. The initial setting for the controller is DISABLE PASS.

(2) NOWAIT

NOWAIT specifies whether the controller should wait for the robot to finish positioning itself before sending out (or taking in) external signals. Signal output timing is explained in Section 5.

A DISABLE NOWAIT statement directs the controller to wait for the robot to finish positioning itself before send out (or taking in) an external signal. The initial setting for the controller is DISABLE NOWAIT.

(3) SWITCH

SWITCH determines whether the task change-over is performed or not in the multitask operation.
The task change-over is prohibited by the DISABLE SWITCH. In the initial setting, ENABLE SWITCH is effective.

(4) MOVESYNC

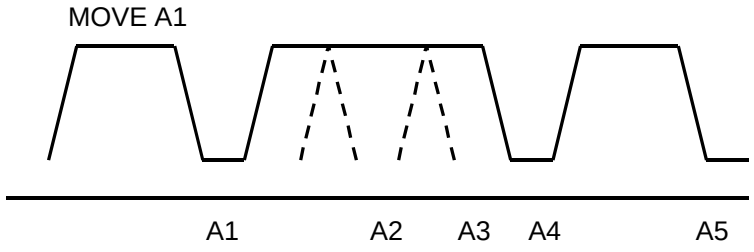
Specifies the motion command synchronous mode or motion command asynchronous mode. In the DISABLE MOVESYNC state (i.e., motion command asynchronous mode), the system pre-executes commands all the way to just before four (4) (max.) motion commands ahead and waits for the finish of positioning. If the system variable PASS is set to “ENABLE”, short-cut (pass) motion is allowed. The initial status is specified by the user parameter [U03].

To make the system switch effective, use the ENABLE command.

Sample program

```
PROGRAM DISABLESPL
    MOVE A1
    PASS = 80
    ENABLE PASS
    MOVE A2
    MOVE A3
    DISABLE PASS
    MOVE A4
    MOVE A5
END
```

The robot will move from point A1 to point A4 with short-cut movement. From point A4 onward, the robot will move normally.



DO

Purpose

The DO command is used in conjunction with the ON command to monitor conditions.

Format

ON <monitoring condition> [{BREAK | PAUSE}] DO <statement>

Examples

ON DIN (1) DO RETURN
ON TIMER DO MOVE A1

Analysis and advice

Should the <monitoring condition> be satisfied, the statement following the DO command will be executed.
Condition monitoring is carried out no matter what kind of movement the robot happens to be doing at the time.

The ON command is processed in parallel with robot motion commands. Should a MOTION, MOTIONT, REMAIN or REMAINT command be used as the monitoring condition, monitoring of conditions for subsequent movement commands will be performed. Should TIMER or ERROR be used as the monitoring condition, conditions will be monitored independently of robot movement.

When monitoring input signals with DIN or other such commands, the timing with which monitoring begins will vary depending on the setting of the NOWAIT system switch. When an ENABLE NOWAIT statement is in effect, signals will be monitored independently of robot movement.

When the DISABLE NOWAIT statement is in effect, monitoring of the signal will start after the robot has completed the movement it was executing at the time.

The execution of the statement following the DO command will start immediately after the execution of the command in effect when the monitoring condition was satisfied. However, if you happened to be executing a WAIT command, the WAIT command will be cancelled immediately and program control will shift to the statement following the DO command.

There are three types of execution timing you can specify for the robot while in operation:

BREAK: BREAK will immediately stop all robot movement and shift control to the statement following the DO command.

PAUSE: The statement following the DO command is executed after the movement now in progress finishes. During arm movement, however, normal program execution continues, except for the subprogram call command, return command to main program and motion command. At execution of these commands, program execution stops until the arm has stopped.

Default: The default setting will cause the movement in progress to be completed while simultaneously executing statements following the DO command.

When the statement following the DO command is a movement command, always include a BREAK or PAUSE statement in the DO command line.

If the statement following the DO command (i.e., DO statement) and the motion command in the DO statement were executed, after the arm movement has finished, program execution will restart in accordance with conditions just before the condition for the ON command was satisfied.

(Should a WAIT command have been interrupted, program execution will restart from the position where the WAIT command was interrupted. However, should a program branch to a label have been carried out with the statement following the DO command, execution will start from the statement having that label.

Ten sets of conditions can be monitored at once. Furthermore, a maximum of four input signals may be specified with a single ON command.

When multiple monitoring conditions become true at once, the DO statement corresponding to the ON command having the highest priority is executed. This priority is determined by the order in which the ON commands were encountered in the program, with the first ON command encountered having the highest priority. DO statements corresponding to all other ON commands are ignored.

Monitoring of a condition specified by one ON command will be cancelled should execution shift to a DO statement corresponding to another ON command. Also, conditions are not monitored while program execution is halted due to a STOP command or an error.

When the system timer is specified as the monitoring condition, the condition is checked only when the monitoring conditions have changed. When monitoring an external signal, an error condition or a movement reference command (such as the amount of a motion remaining to be performed), the controller monitors the state, not the change, of that signal.

The IGNORE command will cancel the monitoring of conditions specified by an ON command. Monitoring of conditions will also stop when a condition is satisfied and a statement following a DO command is executed.

[Note 1]

At present, ON and DO commands may be combined only in the ways shown below:

ON TIMER DO <statement>
When the timer becomes 0, execute the statement.

ON DIN ( ) DO <statement>
When the state of the input signal(s) in the brackets ( ) becomes as specified, execute the statement. You cannot monitor more than four signals at once with one such statement. Up to four input signals can be specified. Extra input signals exceeding four signals are ignored.

ON MOTION > = <expression> DO <statement>
Execute the statement when the amount of a motion which is to be executed next to this command exceeds the specified value. The only relational operand you can use with MOTION is > =.

ON MOTIONT > = <expression> DO <statement>
Execute the statement when the time required for a motion which is to be executed next to this command exceeds the specified time. The only relational operand you can use with MOTIONT is > =.

ON REMAIN < = <expression> DO <statement>
Execute the statement when the remaining amount of a motion which is to be executed next to this command is smaller than the specified value.

The only relational operand you can use with REMAIN is < =.

ON REMAINT < = <expression> DO <statement>
Execute the statement when the remaining time required for a motion which is to be executed next to this command is smaller than the specified time.

The only relational operand you can use with REMAINT is < =.

[Note 2]

In a statement following the DO statement, the following commands relating to the task control cannot be used.

TASK, KILL, SWITCH

If these commands are used in the DO statement and after, they are inoperative. Condition monitor by the ON command is not possible in the subtask.

[Note 3]

If a motion monitored under the condition of ON MOTION, ON MOTIONT, ON REMAIN or ON REMAINT has been stopped, or if the slow speed command has been specified during execution of a monitored motion, the ON condition is cancelled.

Sample program

```
PROGRAM MAIN
    DOSAMPLE
    MOVE P
END
PROGRAM DOSAMPLE
    ON DIN(1) PAUSE DO RETURN
    MOVE A1
    MOVE A2
    MOVE A3
      WAIT MOTION >= 100
    IGNORE DIN(1)
    RETURN
END
```

Should signal 1 turn ON while a movement is being executed, control will be returned to the main program after that movement has been completed.

Cautions on DO statement:

For ON ~ DO command, the ON conditions to be monitored and the DO statement which starts when the conditions are satisfied are registered.

```
PRGRAM MAIN
  SIG = 1
  ON DIN (1) DO INPUT SIG
  SUB
  IGNORE DIN(1)
  PRINT SIG
END

PROGRAM SUB
  MOVE P
  WAIT MOTION >= 100
END
```

In the above SCOL program, if DIN(1) is set ON during traverse to P, the DO statement cannot be executed because the variable SIG is not defined in the program SUB and there is no space for saving the variable as input by the INPUT command. In this case, the relevant DO statement can be executed normally by defining the variable SIG as the global variable.

```
GLOBAL
  SIG = 0
END
PROGRAM MAIN
  SIG = 1
  ON DIN(1) DO INPUT SIG
  SUB
  IGNORE DIN(1)
  PRINT SIG
END
PROGRAM SUB
  MOVE P
  WAIT MOTION >= 100
END
```

In the DO statement, even if the task changeover conditions are established or the SWITCH command is executed, the task cannot be changed over. If the TASK command or KILL command is executed, an error occurs.

DOUT

Purpose

DOUT is used to output external signals.

Format

DOUT (<signal name> [,<signal name>] ...)

Examples

DOUT (1, 2, –3)
DOUT (J, J + 1, J + 2)

Analysis and advice

DOUT is used to output external signals.

<signal name> specifies the number (name) of a signal to be output from the controller.

A positive signal is considered to be ON and a negative signal is considered to be OFF. Up to ten signal names may be specified in a signal DOUT command.

A constant, variable or calculation may be used for the <signal name> specification. However, you may not use vector-type data.

When the same signal is output consecutively after execution of the DO command, signal output is not guaranteed. When output of multiple signals is specified, the simultaneousness of signal output timing is not guaranteed.

DOUT(-4,3,-2,1) Robot program in turn from the head DOUT(-4) DOUT(3), DOUT(-2), DOUT(1) Resolving and being executed by the robot program
In other words, it changes and the signal becomes a value for the purpose with "1010"→"0010"→"0011"→"0001"→"0101" at the number - the interval degree of hundreds of ms, being final when executing the robot program which is called DOUT(-4,3,-2,1) behind DOUT(4,-3,2,-1).

Sample program

```
PROGRAM DOUTSAMPLE
    FOR K = 1 TO 16
    DOUT(K)
    TIMER = 0.5
    WAIT TIMER == 0
    DOUT(–K)
    NEXT K
END
```

This program will send out (turn on) output signals 1 to 16 in order and in 0.5 second intervals.

ELSE

Purpose

The ELSE statement is used in combination with IF ~ THEN constructions in order to judge conditions.

Format

IF <logical expression> THEN <statement> [ELSE <statement>]

Examples

```
IF DIN (1)    THEN K = K + 1    ELSE K = 0
```

Analysis and advice

ELSE is used in an IF statement to specify a statement to be executed if the IF condition is not satisfied.

An ELSE statement is not mandatory in an IF construction. If the IF condition is not satisfied and there is no ELSE statement, program
execution will shift to the next command following the IF command.

The <statement> following the THEN and ELSE statements cannot contain PROGRAM, END, IF, FOR, NEXT or WAIT. For more information on judging conditions, refer to the IF command.

Sample program

```
PROGRAM ELSESAMPLE
    IF DIN (1) THEN SPEED = 100 ELSE SPEED = 50
    MOVE A1
    MOVE A2
    MOVE A3
END
```

Should Input Signal 1 be ON, the robot will operate at full (100%) speed. If OFF, the robot will operate at half (50%) speed.

ENABLE

Purpose

The ENABLE command is used to put system switches into effect.

Format

ENABLE <switch> [, <switch>]...

Examples

ENABLE PASS
ENABLE PASS, NOWAIT

Analysis and advice

The ENABLE command is used to put system switches related to robot movement into effect. There are four (4) system switches.

(1) PASS

PASS is used to specify short-cut movement. Short-cut movement is an operating mode in which the robot is directed to begin its next move before completing its previous move. The timing for switching over from the present movement to the next movement is specified with the PASS command.

Short-cut movement allows you to reduce the time it takes the robot to get from one place to another. For more information, see Section 5.

An ENABLE PASS statement specifies short-cut movement. The initial setting for the controller is DISABLE PASS.

(2) NOWAIT

NOWAIT specifies whether the controller should wait for the robot to finish positioning itself before sending out (or taking in) external signals. Signal output timing is explained in Section 5.

An ENABLE NOWAIT statement directs the controller to send out (or take in) external signals without waiting for the robot to finish positioning itself. The initial setting for the controller is DISABLE NOWAIT.

(3) SWITCH

SWITCH determines whether the task change-over is performed or not in the multitask operation.
The task change-over is prohibited by the DISABLE SWITCH. In the initial setting, ENABLE SWITCH is effective.

(4) MOVESYNC

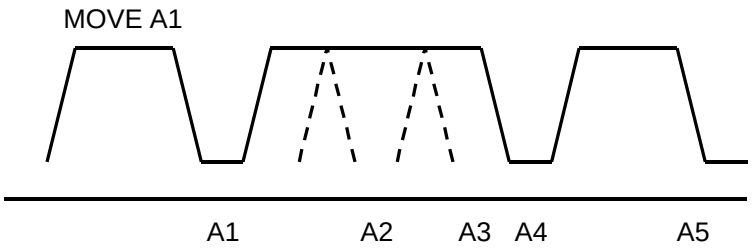
Specifies the motion command synchronous mode or motion command asynchronous mode.
In the ENABLE MOVESYNC state (i.e., motion command synchronous mode), the system executes all the way to just before the next motion command and waits for the finish of positioning. The initial status is specified by the user parameter [U03].
In the MOVESYNC mode, short-cut (pass) motion is not allowed, irrespective of the status of system variable PASS. If the system variable PASS is set to “DISABLE” in the SCOL program, short-cut (pass) motion is allowed.

To make the system switch ineffective, use the DISABLE command.

Sample program

```
PROGRAM ENABLESMPL
    MOVE A1
    PASS = 80
    ENABLE PASS
    MOVE A2
    MOVE A3
    DISABLE PASS
    MOVE A4
    MOVE A5
END
```

This program directs the robot to move A1 to point A4 with short-cut movement point A3 to point A5 without short-cut movement.



END

Purpose	The END statement marks the end of a program.
Format	END
Examples	END
Analysis and advice	The END statement marks the end of a program. When operating in the cycle operation mode, the program will stop by the END statement of the main task generated automatically at the start of the program. In the continuous operation mode, program execution will be returned to the start of the program and the program will repeat itself. When executing a program as a subprogram, you should have a RETURN command in the line above the END statement. Even if you forget to put in a RETURN command, however, control will still be sent back to the main program when the END statement is encountered. PROGRAM marks the beginning of a program and END marks the end. The program itself is sandwiched between the two. If you do not have an END statement, you will get an error message. After the END statement in the main program has been executed, the values of internal variables are cleared.

Sample program

```
PROGRAM ENDSAMPLE
    MOVE A1
    MOVE A2
    MOVE A3
END
```

From PROGRAM to END, this program will be executed as a single program.

EXP

Purpose

The EXP function returns the exponent of a number to the power e.

Format

EXP (<expression>)

Examples

K = EXP (3.5)
J1 = N * EXP (L – K)

Analysis and advice

This function is used to calculate the exponent of an <expression> to the power e. (e = 2.71828...)

A constant, variable or calculation may be used for the <expression> term. However, you may not use vector-type data. This command must be used in an equation.

Sample program

```
PROGRAM MAIN
    EXPSAMPLE (5, K)
    PRINT TP, K, CR
END
PROGRAM EXPSAMPLE (N, K)
    K = EXP (N)
    RETURN
END
```

This subprogram takes an argument N, finds the exponent of that argument to the base e, calls the result K, and sends control back to the main program.

FINE

Purpose

FINE is a system constant used to set the positioning accuracy to fine.

Format

FINE

Examples

ACCUR = FINE
MOVE A1 WITH ACCUR = FINE

Analysis and advice

The FINE statement is used with ACCUR command to set positioning accuracy to FINE. The system constant has a value of 1. This value can be used as constant 1 in the expression. However, do not do it since it makes your program unnecessarily complicated.

You cannot substitute into system constants.
For information on positioning accuracy, refer to the ACCUR command.

Sample program

```
PROGRAM FINESAMPLE
    ACCUR = FINE
    MOVE A1
    MOVE A2
    MOVE A3
END
```

This program sets the positioning accuracy to FINE before moving around the robot.

FOR

Purpose

FOR directs a section of the program to repeat itself.

Format

FOR <variable> = <expression 1> TO <expression 2>
[STEP<express ion 3>)
• • • •
NEXT [<variable>)

Examples

FOR K = 1 TO 4
• • • •
NEXT K

FOR N = K1 TO K1 + K2 STEP K3
• • • •
NEXT N

Analysis and advice

The FOR commands directs a part of the program to repeat itself.
The program block between the FOR command and the NEXT command is executed repeatedly. The block will keep on repeating itself until the condition specified by the FOR statement is satisfied.

When a FOR statement is executed, the value of <expression 1> is substituted into the <variable>. When the NEXT statement is executed, the value of <expression 3> specified by the STEP statement is added on to the <variable>.

Should the value of the <variable> become greater than the value of <expression 2> at this time, the execution of the program will shift to the statement following the NEXT command. If <variable> is not greater than <expression 2>, the program execution will branch (go back) to the statement following the FOR statement.

The values of <expression 1>, <expression 2> and <expression 3> used in the FOR construct are those in effect when the FOR statement was first executed. Therefore, even should these values be changed while executing the loop, the number of times the loop is repeated will not change.

The <variable> should be used only to control the number of times the loop is repeated. Therefore, do not change the value of the <variable> while executing the loop.

If the value of <expression 3> is 1, you may omit statements after the STEP statement.

A constant, variable or calculation may be used for <expression 1>, <expression 2> or <expression 3>. However, you may not use vector-type data.

For the corresponding NEXT statement, you should specify the variable specified by the FOR statement.

If you do not specify <variable> in the NEXT statement, a loop is made between the nearest FOR statement (executed finally) and the NEXT statement.

When another FOR command (being nested) is used in a FOR – NEXT loop, the number of nesting levels should be 32 levels or less.
When the number of nesting levels exceeds 32 levels, an error occurs.

Note 1: A FOR loop is ended by the NEXT command. Therefore, no matter what, the loop will be executed at least once.

Note 2: Real numbers may be used for the <variable>, <expression 1>, <expression 2> or <expression 3>. However, since there is a certain imprecision when handling real numbers, try to use integers when telling the FOR statement how many times to repeat itself. Furthermore, values substituted into the <variable>, <expression 1>, <expression 2> or (expression 3> are converted into the data type when the loop is executed. Data types are converted as shown below.

(1) When the <variable> data type is undefined when a FOR statement is executed:
The data type of a variable will be undefined should the identifier used for the <variable> appear for the first time in the program in the FOR command. In such a case, all data will be converted to and processed as the data type of <expression 1>.

(2) When the <variable> data type has been defined before a FOR statement is executed:
When the identifier (used for the variable) has been used in the program beforehand, the data type of the <variable> will be the same as the data type of the data which was first entered into that <variable>.

Example:

```
PROGRAM SAMPLE
    J1 = 5
    FOR J1 = 0.1 TO 0.9 STEP 0.01
        MOVE A1
        MOVE A2
    NEXT J1
END
```

When the above program is executed, the data type of J1 is defined as the integer type before the FOR command is executed. Therefore, all variables used in the FOR command are converted into the integer type.
Therefore, the controller will interpret the above FOR statement as:
FOR J1 = 0 TO 0 STEP 0
Consequently, the FOR loop will only be executed once.

Note 3: When the FOR statement is used in the following manner, an error will occur.

(1)
```
FOR J1 = ...
FOR J2 = ...
 ...
NEXT J1
```

Here, you are missing a NEXT statement corresponding to the FOR J2 statement.

(2)
```
FOR K = ...
...
IF DIN(1) THEN GOTO L1
...
NEXT K
...
L1:
```

A branch command from the inside to the outside of the loop or vise versa is not allowed.

Sample program

```
PROGRAM FORSAMPLE
    FOR K = 1 TO 100
        MOVE A1
        MOVE A2
    NEXT K
END
```

The robot will make 100 trips back and forth between A1 and A2.

FREE

Purpose

FREE is a system constant used to make the robot configuration as undefined.

Format

FREE

Examples

```
CONFIG    FREE
MOVE A1 WITH CONFIG = FREE
```

Analysis and advice

The FREE system constant is used along with the CONFIG command in order to specify the robot configuration as undefined.

FREE has the value of 0. If you wanted to, you could use FREE in your program as a constant having the value 0. However, do not do this since it makes your program hard to read.

You cannot change (substitute into) system constants including FREE.
For information on robot configuration, see the "CONFIG command."

Sample program

```
PROGRAM FREESAMPLE
    CONFIG = FREE
    MOVE A1
    MOVE A2
END
```

This program will make the robot configuration undefined before beginning to move the robot.

FREELOAD

Purpose	The FREELOAD command will zero the data for the load acting on the end of the robot hand.
Format	FREELOAD
Examples	FREELOAD
Analysis and advice	The FREELOAD command will zero the data for the load acting on the end of the robot hand. In order that the robot operate effectively under various loads, the SCOL language makes it possible to set load data acting on the end of the robot hand. It is important that the robot know this information so that it does not damage itself by swinging around too quickly. Loads acting on the robot hand are set with the system variable PAYLOAD. The controller uses these values to calculate control constants for robot acceleration and deceleration that are appropriate for the load. Load data consists of values for the load mass and the load moment of inertia. When a FREELOAD command is executed, all load data will become 0. This is true for both mass load data and inertia load data. This command will not work if you do not have the file SCOL.LIB in the controller RAM drive.

Sample program

```
PROGRAM FREELOADSL
    PAYLOAD=HAND
    MOVE A1
    CLOSE1
    DELAY 0.5
    MOVE A2 WITH PAYLOAD=HAND + MOTOR
    OPEN1
    DELAY 0.5
    MOVE A3
    FREELOAD
    MOVE A2
    MOVE A3
END
```

This program sets the load data to zero before moving around the robot.

GAIN

Purpose

The GAIN command is used to specify whether the gain (for servo control) is ON or OFF for each axis.

Format

GAIN = {<expression>, <expression>, <expression>, <expression>, <expression>}

Examples

GAIN = {0, 0, 1, 0, 0}
MOVE A1 WITH GAIN = {,, ON}

Analysis and advice

The GAIN command is used to specify whether the gain (servo control) of each axis is to be ON or OFF.

Should the GAIN be specified as OFF, servo control for that axis will stop the next time a movement command is executed. Axes for which servo control has been stopped are in the servo free state (in which positioning control is not carried out) and can be moved around freely by external forces.

Positioning checks are not performed for axes for which the gain is OFF. Furthermore, should you give a movement command to one of those axes, the axis will not move.

This command is used when, for example, one is fitting a part into a hole in a workpiece.

By turning off all the gains except for the Z-axis (which must be on so you can move the hand up and down), all the axes (except for the Z-axis) are free (i.e., not locked up) and the robot hand can move freely in the horizontal direction. This allows the robot to move horizontally along with the workpiece and fit in the part in the hole even should the workpiece and part not be aligned exactly.

The GAIN command is treated as a system variable having five values (that correspond to the five axes). The programmer specifies which gains are to be on and off for each axis in the { } brackets following the GAIN commands.

Specifications are divided by commas with the first specification corresponding to Axis 1, the second to Axis 2, and so on. A "0" means that the gain is to be OFF and a "1" means that the gain is to be ON. Should specifications be abbreviated (for example, {,, ON}), the controller will assume all non-specified gains to be OFF.

You may use constants, variables or calculations for the (expression> terms. You may also use the system constants ON and OFF. (ON sets the gain on, and OFF sets the gain off.) However, you may not use vector-type data.

You must always synchronize the execution of a GAIN command with the movement of the robot. That is, be sure that the GAIN command is executed after the previous movement command is completed (as seen in the sample program).

Should you execute a GAIN command while the robot is still moving, the robot may move incorrectly. For more information on setting gains in synchronization with the robot movement, see the "SETGAIN command."

Even should you complete automatic operation and switch over to the manual mode, the GAIN settings will not change. In order to turn on gains which have been turned off in the manual mode, push the corresponding guide keys for those axes while in the guide mode.

When a selected gain value is not more than 0 for ON and not less than 1 for OFF, 0 and 1 are considered to be specified.

Sample program

```
PROGRAM GAINSAMPLE
    MOVE A1
    WAIT MOTION > = 100
    GAIN = {OFF, OFF, ON, OFF, OFF}
    MOVE A2
    OPEN1
    DELAY 0.5
    MOVE A1
    WAIT MOTION > = 100
    GAIN = {ON, ON, ON, ON, ON}
    READY
END
```

This program turns off all gains except that for the Z-axis (Axis 3) before moving the robot to point A2.

```
PROGRAM GMOVE
    G = GAIN
    GAIN = {,, ON}
    MOVE P1
    WAIT MOTION > = 100
    GAIN = G
    MOVE P2
END
```

This program turns off the gains on all axes except Axis 3 (the Z-axis), and later restores the gains to their original state after moving.

GLOBAL

Purpose

This function specifies the global variable area.

Format

GLOBAL

Examples

```
GLOBAL
   A = 10
END
```

Analysis and advice

The global variable which can be referred to and substituted from any part of the program is defined. To identify the type of a variable other than the array, setting of an initial value in the substitution format is required.

As the array variable defines the type and number of elements by DIM command and sets the initial value separately, some may not have an initial value.

The following types can be used for the global variable area.

Integer type, real number type, position type, coordinate type and load type.

Combination of the above variables with the array.

Specify the variable definition of integer type and real number type, and initial values of integer type and real number type arrays in global blocks, and the variable definition of position type, coordinate type and load type, and initial values of position type, coordinate type and load type arrays in data blocks.

Sample program

```
GLOBAL
    A = 1
END

PROGRAM TEST
    A = A + 1
    PRINT “A=”, A, CR
END
```

GOTO

Purpose

The GOTO command specifies that the execution of the program is to be branched to the location marked by the label of the GOTO command.

Format

GOTO <label>

Examples

IF DIN(1) THEN GOTO L1
GOTO LOOP
GOTO RESTART

Analysis and advice

The GOTO command specifies that the execution of the program is to be branched to the location marked by the label of the GOTO command.

Branching locations for the GOTO command are limited to statements in the same program. Should there be no location in the program corresponding to the GOTO label, you will get an error. Furthermore, should you have several statements with the same label in the same program, the controller will not know where to go and you may get an error.

In order to label a branching location, put the label name (identifier) at the beginning of the statements you wish to execute. Be sure to put a colon after the identifier.

Sample program

```
PROGRAM GOTOSAMPLE
    MOVE A1
    LOOP:
        MOVE A2
        MOVE A3
    GOTO LOOP
END
```

When the program is executed finally, the program goes back to the label LOOP.

GOTO ( )

Purpose	This command will cause the execution of the robot to branch off depending on the value of the expression in the brackets.
Format	GOTO (<expression>) <label> [, <label>]….
Examples	GOTO (K) LABEL1, LABEL2, LABEL3 GOTO (N1 – N2) L1, L2, L3, L4, L5
Analysis and advice	The GOTO ( ) command will cause the execution of the program to be branched off in accordance with the value in the brackets ( ). When the value in the brackets (i.e., the <expression>) is 1, the program will branch off to the <label> furthest to the left. When the value in the brackets is 2, the program will branch off to the <label> second furthest to the left, and so on. Up to 10 labels can be specified. Extra labels exceeding 10 labels are ignored. Should the value in the brackets be greater than the number of labels or should the value be zero or less, program execution will proceed to the statement following the GOTO statement. Should the value in the brackets be a real number, all decimal points will be cut off and what remains will be treated as an integer. You may use constants, variables or equations for the <expression> term However, you may not use vector-type data.

Branching locations for the GOTO ( ) command are limited to statements in the same program. Should there be no location in the program corresponding to the specified GOTO ( ) label, you will get an error. Furthermore, should you have several statements with the same label in the same program, the controller will not know where to go and you may get an error.

In order to label a branching location, put the label name (identifier) at the beginning of the statements you wish to execute. Be sure to put a colon after the identifier.

Sample program

```
PROGRAM MAIN
    INPUT N
    GOTOSAMPL2 (N)
END
PROGRAM GOTOSAMPL2 (N)
    GOTO (N) L1, L2, L3
    RETURN
    L1: MOVE A1
    RETURN
    L2: MOVE A2
    RETURN
    L3: MOVE A3
    RETURN
END
```

If the argument N is 1, 2 or 3, the robot will move to point A1, A2 or A3 respectively. If N is not 1, 2 or 3, program execution will be sent back to the main program without moving the robot.

HERE

Purpose

The HERE statement returns the current position of the robot.

Format

HERE

Examples

MOVE HERE
A1 = HERE
X = HERE. X

Analysis and advice

The HERE command returns the current position of the robot on the world coordinate system.
HERE can be handled just like any other positional- type data with the exception that you can only refer to the values contained inside and cannot change the values themselves.
When a HERE command is executed while the robot is moving, the commanded position HERE at the time of HERE command execution is returned.

Note: The position returned with the HERE command is the position commanded to the robot. Note that while the robot is moving, the actual current position of the robot has a delay from the commanded position.
It is possible to acquire the posture of the present robot by the HERE command.

N=HERE.6

The value of the present posture is stored in N.
With a SCARA robot, the value of HERE.6 is 0 for Free (undefined), 1 for Left-handed system, or 2 for Right-handed system.

HEXIN

Purpose

This function reads input signals in the hexadecimal notation.

Format

HEXIN (<signal name>, <signal length>)

Examples

K = HEXIN (1, 2)
J2 = HEXIN (N, N+2)
GOTO (HEXIN (20, 2)) L1, L2, L3

Analysis and advice

This function reads the input signals of <signal length> in the hexadecimal notation, starting with the signal specified by <signal name>.
When K = HEXIN (1, 2), for instance, the state of two (2) input signals 1 and 2 is read as the HEX code.
The signal length can be specified in the range of 1 ~ 32. As the input signal number increases, it corresponds to higher-order bit accordingly. Each signal is coded as follows.
ON = 1, OFF = 0

Example: When the state of input signals 1 ~ 12 is as shown in the table below, the value of "HEXIN (1, 12)" is 809 (i.e., 329 in the hexadecimal notation).

Input signal number	12	11	10	9	8	7	6	5	4	3	2	1
Input signal state	OFF	OFF	ON	ON	OFF	OFF	ON	OFF	ON	OFF	OFF	ON
Binary expression	0	0	1	1	0	0	1	0	1	0	0	1
Hexadecimal expression				3				2				9

You can use a constant, variable, expression and calculation for <signal name> and <signal length>. You cannot use the vector-type data, however.
This command is used in the expression.

Sample program

```
PROGRAM HEXINSAMPL
    K = HEXIN (1, 7)
    SPEED = K
    MOVE A1
    MOVE A2
END
```

The state of input signals 1 ~ 7 is coded, the motion speed is set according to such a value, then the robot moves at the set speed.

HEXOUT

Purpose

This function outputs signals by coding them in the hexadecimal notation.

Format

HEXOUT (<signal name>, <signal length>, <expression>)

Examples

HEXOUT (1, 4, 3)
HEXOUT (N, N+4, K)

Analysis and advice

This function codes the value of <expression> in the hexadecimal notation and outputs the digits specified by <signal length> to the number of the signals as specified by <signal length>, starting with the signal specified by <signal name>.
When HEXOUT (1, 4, 3) is commanded, for instance, the value of "3" which is coded in the hexadecimal notation is output to the four signals (1 ~ 4), and output signals 1 and 2 become ON.
As the output signal number increases, it corresponds to higher-order bit accordingly. The signal length can be specified in the range of 1 ~ 32. If the value of <expression> exceeds the value specified by <signal length>, the high-order digit or digits are ignored. As the input signal number increases, it corresponds to higher-order bit.
Each signal is coded as follows.
ON = 1, OFF = 0

Example: When HEXOUT (1, 12, 952) is commanded, the output signals are as shown below.

Decimal expression												952
Hexadecimal expression				3				B				8
Binary expression	0	0	1	1	1	1	0	1	1	0	0	0
Output signal number	12	11	10	9	8	7	6	5	4	3	2	1
Output signal state	OFF	OFF	ON	ON	ON	ON	OFF	ON	ON	OFF	OFF	OFF

You can use a constant, variable, expression and calculation for <signal name>, <signal length> and <expression>. You cannot use the vector-type data, however. If the same signals are output consecutively, the signal output last becomes effective. Be careful of agreement's there being in the signal range by which it is possible to guarantee simultaneous-ness.
The simultaneous-ness of HEXOUT to the range of the 16 bit carving from DOUT1, DOUT101, DOUT301 can be guaranteed but the simultaneous-ness of HEXOUT which strides the boundary can not be guaranteed.

Sample program

```
PROGRAM HEXOUTSMPL
    FOR K = 1 TO 4
    J = 2 ^ (K–1)
    HEXOUT (1, 4, J)
    TIMER = 0.5
    WAIT TIMER == 0
    HEXOUT (1, 4, 0)
    NEXT K
END
```

Output signals 1 ~ 4 are output in turn at intervals of 0.5 second.

IF

Purpose	The IF statement is used for judging conditions.
Format	IF <logical expression) THEN <statement> [ELSE <statement>]
Examples	IF DIN (1) THEN K = K + 1 ELSE K = 0
Analysis and advice	If the conditions of the <logical expression> following IF are satisfied, the <statement> following THEN will be executed. If the conditions are not satisfied, the statement following ELSE will be executed. An ELSE statement is not mandatory in an IF construction. If the IF condition is not satisfied and there is no ELSE statement, program execution will shift to the next step following the IF command. The <statement> following the THEN or ELSE statement may not contain PROGRAM, END, IF, FOR, NEXT or WAIT. IF ~ THEN ~ ELSE constructs are considered as a single unit, and for that reason they all have to be on the same line. (That means you must write everything following the ELSE statement on that line.)

Sample program

```
PROGRAM IFSAMPLE
    IF DIN (1) THEN K = 1 ELSE K = 0
    MOVE A1
    MOVE A2
    MOVE A3
    PRINT TP, K, CR
END
```

Should Input Signal 1 be ON, K will equal 1.
Should Input Signal 1 be OFF, K will equal 0.

IGNORE

Purpose	The IGNORE command is used to cancel the monitoring of a condition specified by a previous ON command.
Format	IGNORE <monitoring condition>
Examples	IGNORE DIN (1) IGNORE TIMER
Analysis and advice	The IGNORE command is used to cancel the monitoring of a condition specified by a previous ON command. In the <monitoring condition> specification, use the exact same logical expression as you used for the corresponding ON statement. However, the ON condition statement that can be ignored should be specified before the IGNORE command line. At this time, the same ON condition should not be specified on two (2) or more lines. When the IGNORE statement is executed, monitoring of the condition will cease. For more information on condition monitoring, refer to the ON command.

Sample program

```
PROGRAM MAIN
    IGNORESMPL
    MOVE P
    MOVE P2
END
PROGRAM IGNORESMPL
    ON DIN (1) PAUSE DO RETURN
    MOVE A1
    MOVE A2
    WAIT MOTION > = 100
    IGNORE DIN (1)
    RETURN
END
```

Should Input Signal 1 go on, execution will return to the main program after the motion in progress at the time is completed. Monitoring of Input Signal 1 will cease when the movement from point A1 to point A2 is completed.

INITPLT

Purpose

Initializes a pallet.

Format

INITPLT (<pallet number>, <i>, <j>, <k>)

Examples

INITPLT (1, 5, 4, 3)

Analysis and advice

This function is used to initialize the pallet to execute the palletizing command (MOVEPLT).

Pallet number : Number assigned to the pallet, starting with number "1" (i.e., any integer larger than "1").

i : Number of elements from pallet home point to point I (i.e., any integer larger than "1").

j : Number of elements from pallet home point to point J (i.e., any integer larger than "1").

k : Number of elements from pallet home point to point K (i.e., any integer larger than "1").

If the value of i, j or k is zero (0) or less, the program stops with an error message saying **"ERR!! ELEMENT IS TOO SMALL"** shown on the teach pendant display.

The INITPLT command is available in the dynamic link library. When executing this command, library build-in and global variable should be declared in the GLOBAL area.

For further information, see Appendix G–1.

Sample program

```
GLOBAL
   LOADLIB PALLET.LIB          Library build-in declaration.
   DIM PLTP (1, 7) AS POINT    Global variable declaration.
END

PROGRAM SAMPLE
   INITPLT (1, 3, 4, 2)        Pallet initialization to "3 × 4 × 2"
                               with teach points PLTP (1, 1) ~
                               PLTP (1, 4).
   MOVEPLT (1, 1, 0, 0, 0, 0)  Move to pallet No. 1, element No.
                               1.
END
```

INPUT

Purpose

The INPUT command reads in data from a specified communications channel.

Format

INPUT[{COM0 | COM1 | TP},] <variable> [, <variable>]…

Examples

INPUT K1, K2, K3
INPUT COM1, K

Analysis and advice

The INPUT command is used to read in data from a communication channel. This data may be either real or integer numbers.

Specify one (1) communication channel from COM0, COM1, and TP. COM0 and TP are channels used solely for the teach pendant. COM1 corresponds to controller COM1 communication channel.

If you do not specify a communication channel in your INPUT statement, data will be read in from the teach pendant communication channel.

When an INPUT command is executed, the program will wait until the data is read in from the communication channel.

Data which has been read in will be placed in the assigned variable(s). If there is more data than there are variables, excess data will be ignored. If there is less data than there are variables, the program will wait until the remaining data comes in.

When inputting data from the teach pendant, keep real numbers separate with commas. When you are done entering the numbers, push the EXE key.

When inputting data from anywhere other than the teach pendant, the data will be processed when transmission is completed. For information on data communication, refer to the Communication Manual.

After the moving arm has stopped, this command is not executed.

Sample program

```
PROGRAM INPUTSMPL
PRINT COM0, "*** INPUT N1, N2, N3 ***"
INPUT COM0, N1, N2, N3
PRINT (N1 + N2 + N3)/3, CR
END
```

This program will read in three values (N1, N2 and N3) from the teach pendant, find the average, and display the average on the teach pendant.

INT

Purpose

The INT command changes a numerical value into an integer.

Format

INT (<expression>)

Examples

AK = INT (–20.345)
N = INT (K)
J1 = K – INT (N – 28.5)

Analysis and advice

The INT command converts the number or calculation result in the brackets ( ) to an integer.
Note that the INT command simply cuts off real numbers to the right of the decimal point and converts them into integer.

This command is used when one wants to specify the data type of a variable as an integer-type.

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

The INT command must be used in an expression.

Sample program

```
PROGRAM MAIN
    INTSAMPLE (2, 30, K)
    PRINT TP, K, CR
END
PROGRAM INTSAMPL (L, R, K)
    K = INT (L * COS (R))
    RETURN
END
```

This program takes in arguments L and R, finds the value of (L * COS (R)), cuts off any decimal places and returns the result as argument K to the main program.

KILL

Purpose	This function determines the multitask operation.
Format	KILL (<expression>)
Examples	KILL (TASKID)
Analysis and advice	The KILL command terminates the task which has the task number specified by the calculation result of the expression in brackets ( ). If the task ID (settled on 1) of main task generated automatically at the start of program or the non-existing task ID is specified, NOP operation is effective. The task terminated by the KILL command starts by the TASK command. When this happens, a new number is assigned to the task number. The constant, variable and arithmetic expression can be used for the <expression>. The vector type data cannot be used. The task for which stop is specified is to be deleted from the system at the time when it is executed. That is, if system variable SWITCH is DISABLE, task changeover will not occur and other tasks cannot be stopped. To execute the KILL command, set system variable SWITCH to ENABLE. This command is invalid during step execution

Sample program

```
GLOBAL
    MAXTASK=2
    K=0
END

PROGRAM MAIN
    TID1=0
    LOOP:
    IF DIN(1) AND TID1==0 THEN TID1=TASK("SUB1")
    IF DIN(–1) AND TID1<>0 THEN KILL(TID1)ELSE GOTO
     LOOP1
    TID1=0
    LOOP1:
    MOVEA 1, –90
    MOVEA  1,90
    GOTO LOOP
END
PROGRAM SUB1
    K=K+1
    PRINT K,CR
END
```

A task is created when the input signal 1 is turned on, which is cleared when the same signal is turned off.

LATCH (Option of TS3000)

Purpose	This function specifies ON/OFF of the position latch function, using the exclusive input port signals.
Format	LATCH
Examples	DISABLE LATCH ENABLE LATCH
Analysis and advice	This function specifies whether the exclusive input port signals should be monitored or not monitored to latch the position. To turn on and off the system switch, use the ENABLE and DISABLE commands, respectively. When the ENABLE LATCH command is specified, the position latch function, using the exclusive input port signals, becomes effective. When the DISABLE LATCH command is specified, the same function becomes ineffective. In the initial state, the DISABLE LATCH command takes effect. If the start edge detection of the exclusive input port signal is specified and the ENABLE LATCH command is used during the exclusive port signal ON, the operation of the position latch function cannot be guaranteed. The same is also applicable if the fall edge detection of the exclusive input port signal is specified and the ENABLE LATCH command is used during the exclusive port signal OFF. When this happens, however, no error is generated and the processing continues. Only after confirming the state of the exclusive input port signal by means of the DIN command, program the ENABLE LATCH command. The exclusive input port signals are assigned to 53 through 56.

Also, after confirming the state of LATCHSIG1 and 2, acquire the latch position.
When you use this function, you should provide an exclusive board. If the ENABLE LATCH command is executed while the system is not provided with an extension board, an error occurs.

Sample program

```
PROGRAM LATCHSMP
    DISABLE NOWAIT
    MOVE A0
    LATCHTRG1 = 1
    IF DIN (49) THEN GOTO FINI
    ENABLE LATCH
    MOVES A1
    WAIT MOTION >= 100
    IF LATCHSIG1 == 1 THEN LP = LATCHPSN1
        ELSE LP = HERE
    DISABLE LATCH
    FINI:
    MOVE LP
END
```

LATCHTRG1 ~ 8 (Option of TS3000)

Purpose

This function specifies the detected edge direction of the position latch function, using the exclusive input port signals.

Format

LATCHTRG1 = {0 | 1}
LATCHTRG3 = {0 | 1}

Examples

LATCHTRG1 = 1
A = LATCHTRG8

Analysis and advice

These are the system variables for specifying the detected edge direction which serves as the trigger of the position latch, using the exclusive input port signals.
LATCHTRG1 ~ 8 specify the detected edge direction of the exclusive input ports.
When "0" is specified, the signal falls (ON → OFF). Likewise, when "1" is specified, the signal starts up (OFF → ON).
A numeric value other than the integers cannot be specified for LATCHTRG1 ~ 8. If a value other than "0" is specified, the system takes it as "1".
When you refer to this system variable, you can refer to the current edge direction detected. The default of the detected edge direction is "1" (OFF → ON).
If the detected edge direction is changed during execution of the ENABLE LATCH command, the operation of the position latch function cannot be guaranteed. When this happens, however, no error is generated and the processing continues.
To identify the programmed operation, execute the DISABLE LATCH command, then change the detected edge direction.
When you use this function, you should provide an exclusive board. If LATCHTRG1 ~ 8 is specified in the system without an extension board, however, the operation is not affected at all.

Sample program

```
PROGRAM LATCHSMP
    DISABLE NOWAIT
    MOVE A0
    LATCHTRG1 = 1
    IF DIN (49) THEN GOTO FINI
    ENABLE LATCH
    MOVES A1
    WAIT MOTION >= 100
    IF LATCHSIG1 == 1 THEN P1 = LATCHPSN1
        ELSE P1 = HERE
    DISABLE LATCH

    LATCHTRG2 = 0
    IF DIN (–50) THEN GOTO FINI
    ENABLE LATCH
    MOVES A2
    WAIT MOTION >= 100
    IF LATCHSIG2 == 1 THEN P2 = LATCHPSN2
        ELSE P2 = HERE
    DISABLE LATCH

    FINI:
    MOVE A0
END
```

LATCHSIG1 ~ 8 (Option of TS3000)

Purpose	This function refers to the position latch state, using the exclusive input port signals.
Format	LATCHSIG1 LATCHSIG5
Examples	A = LATCHSIG1 IF LATCHSIG2 ==0 THEN GOTO ERR
Analysis and advice	This function refers to whether the position has been latched, using the exclusive input port signals. LATCHSIG1 ~ 8 refer to the position latch state of the exclusive input ports. When the position is latched, the system returns "1". Otherwise, it returns "0". During execution of the DISABLE LATCH command, the system returns "0", irrespective of the exclusive signal state. Also, if the exclusive signal has turned off during edge detection at signal startup, the state of LATCHSIG1 ~ 8 becomes "0". Likewise, if the exclusive signal has turned on during edge detection at signal fall, the state of LATCHSIG1 ~ 8 becomes "0". LATCHSIGN1 ~ 8 can only be referred to and cannot be substituted. Also, LATCHSIGN1 ~ 8 cannot be used as the ON condition. If you wish to refer to the signal under the ON condition, you should refer to 49 ~ 56, using the DIN command. When you use this function, you should provide an exclusive I/O board. If LATCHSIG1 ~ 8 is referred to in the system without extension I/O board, however, the system always returns "0".

Note: The latch state identified by LATCHSIG1 ~ 8 is the current robot state. Real robot motion is delayed due to processing of the SCOL program.
It is recommended to get the latch state and latch position only after execution of "WAIT MOTION >= 100".

Sample program

```
PROGRAM LATCHSMP
    DISABLE NOWAIT
    MOVE A0
    LATCHTRG1 = 1
    IF DIN (49) THEN GOTO ERR
    ENABLE LATCH
    MOVES A1
    WAIT MOTION >= 100
    IF LATCHSIG1 == 0 THEN GOTO ERR
    LP = LATCHPSN1
    DISABLE LATCH
    GOTO FINI
    ERR:
    PRINT "LATCH ERROR", CR
    LP = HERE
    FINI:
    MOVE LP
END
```

LATCHPSN1 ~ 8 (Option of TS3000)

Purpose

This function gets the latch position, using the position latch function.

Format

LATCHPSN1
LATCHPSN2

Examples

P1 = LATCHPSN1
AX = LATCHPSN2. X

Analysis and advice

This function gets a position in the world coordinate system when the edge of the exclusive input port signal has been detected.
LATCHPSN1 ~ 8 get the detected edge position of exclusive input ports 1 ~ 8 in the world coordinate system.
Specify the detected edge direction by LATCHTRG1 ~ 8.
Processing of edge detection becomes effective in the ENABLE LATCH mode, which is ineffective in the DISABLE LATCH mode.
LATCHPSN1 ~ 8 can be referred to when the state of corresponding LATCHSIG1 ~ 8 is "1". Even if the state of LATCHSIG1 ~ 8 is "0", LATCHPSN1 ~ 8 can be referred to, but the value cannot be guaranteed.
When you use this function, you should provide an exclusive board. If LATCHPSN1 ~ 8 is referred to in the system without an extension board, however, the origin of the world coordinate system is indicated.

Note: The latch state identified by LATCHSIG1 ~ 8 is the current robot state. Real robot motion is delayed due to processing of the SCOL program.
It is recommended to confirm the latch state and latch position only after execution of "WAIT MOTION >= 100".

Sample program

```
PROGRAM LATCHSMP
    DISABLE NOWAIT
    MOVE A0
    LATCHTRG1 = 1
    IF DIN (49) THEN GOTO ERR
    ENABLE LATCH
    MOVES A1
    WAIT MOTION >= 100
    IF LATCHSIG1 == 0 THEN GOTO ERR
    LP = LATCHPSN1
    DISABLE LATCH
    GOTO FINI
    ERR:
    PRINT "LATCH ERROR", CR
    LP = HERE
    FINI:
    MOVE LP
END
```

LEFTY

Purpose

LEFTY is a system constant used to change over the configuration of the robot to a left handed system.

Format

LEFTY

Examples

```
CONFIG = LEFTY
MOVE A1 WITH CONFIG = LEFTY
```

Analysis and advice

LEFTY is used in conjunction with CONFIG in order to set the robot configuration to a left handed system.

As a system constant, LEFTY has the value of 1. If you wanted to, you could use it in your program as a constant having the value 1.
However, this is not a good idea since it makes your program unnecessarily complicated.

You cannot substitute into system constants.

For Cartesian coordinate robots, designation of robot configuration is ignored.

For information on robot configuration, refer to the CONFIG command.

Sample program

```
PROGRAM LEFTYSMPL
    CONFIG = LEFTY
    MOVE A1
    MOVE A2
END
```

This program will set the robot configuration to a left handed system before moving the robot on its way.

LN

Purpose

This function calculates the natural logarithm of a number.

Format

LN (<expression>)

Examples

K = LN (100)
J1 = 1 – N (50 – D)

Analysis and advice

The LN command will return the natural logarithm of the number in the brackets ( ). However, be warned that the result of LN (0) will be returned as 0 (when in fact it is undefined and would be expected to be returned as an error).

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

The LN command must be used in an expression.

Sample program

```
PROGRAM MAIN PROGRAM LNSMPL (N, K)
    LNSMPLE (3, K)
    PRINT TP, K, CR
END
PROGRAM LNSMPL (N,K)
    K = 10 ^ N
    K = LN (K)
    RETURN
END
```

This program will take the value of a constant logarithm given by the argument N, convert this value into a natural logarithm, and send the result back to the main program as argument K.

LOADLIB

Purpose

This function reads a dynamic link library.

Format

LOADLIB <file name>

Examples

LOADLIB PALLET.LIB

Analysis and advice

Be sure to declare the LOADLIB command in the GLOBAL area. The library file name is ********.LIB.

Up to five (5) libraries can be read in the same program.

For the dynamic library, see Para. 2.8.3 and Appendix G.

Sample program

```
GLOBAL
    LOADLIB PALLET.LIB        Library build-in declaration
END
PROGRAM MAIN
    INPUT N
    GOTOSAMPL2 (N)
END

PROGRAM GOTOSAMPL2 (N)    Example of library file
    GOTO (N) L1, L2, L3
    RETURN
    L1:
    MOVE A1
    RETURN
    L2:
    MOVE A2
    RETURN
    L3:
    MOVE A3
    RETURN
END
```

LOG10

Purpose

This function calculates the common logarithm of a number.

Format

LOG10 (<expression>)

Examples

K = LOG10 (100)
J1 = 1 – LOG10 (50 – D)

Analysis and advice

The LOG10 command will return the common logarithm of the number in the brackets ( ). However, be warned that the result of LOG10 (0) will be returned as some undefined number (when in fact it would be expected to be returned as an error).

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

The LOG10 command must be used in an expression.

Sample program

```
PROGRAM MAIN
    LOG10SAMPLE (3, K)
    PRINT TP, K, CR
END
PROGRAM LOG10SMPL (N,K)
    K= EXP (N)
    K= LOG10 (K)
    RETURN
END
```

This program will take the value of a natural logarithm given by the argument N, convert this value into a common logarithm, and send the result back to the main program as argument K.

MAXTASK

Purpose	This function specifies the maximum number of tasks that can be executed at the same time in the program containing the multitask function.
Format	MAXTASK
Examples	MAXTASK = 4
Analysis and advice	This variable can be used only in the global data block. Normally, specify the value of "No. of TASK commands + 1" for this variable. The maximum value is four (4). If a plural number of tasks are used, only the last value takes effect. Unless the multitask function is used, this variable need not be used. When this happens, the default value is 1 and the work area of only the main task is maintained. The work area of the controller is assigned to each task, divided equally by this variable. If a large value is specified, the work area that can be used by one (1) task reduces and a large-sized program cannot be executed.

Sample program

```
GLOBAL
  A=0
  MAXTASK=2                          As one (1) subtask is
END                                  used, "2" is set.
PROGRAM MAIN
  ID=0
  ID=TASK ("SUB")                    A loop is formed in the
  LOOP:                              GOTO statement to
  IF DIN(1) THEN A=1 ELSE A=0        prevent repeated call
  GOTO LOOP                          of the task command.
END
PROGRAM SUB
  ENABLE NOWAIT
  IF A==0 THEN PRINT "A", A, CR
END
```

MOD

Purpose

The MOD function returns the remainder of a division operation.

Format

<expression> MOD <expression>

Examples

N = K MOD 3
J = K + (L MOD M)

Analysis and advice

The MOD function will take the <expression> on the left, divide it by the <expression> on the right, and send back the remainder.

You may use constants, variables or equations for the <expression>. However, you may not use vector-type data.

The MOD command must be used in an expression.

Sample program

```
PROGRAM MAIN
    MODSAMPLE (5.0, 3.0, K)
    PRINT TP, K, CR
END
PROGRAM MODSAMPLE (N1, N2, K)
    K = N1 MOD N2
    RETURN
END
```

This program argument N2, main program takes argument N1, divides it by and sends the result back to the as argument K.

MODE

Purpose

MODE is used to refer to the system operating mode

Format

MODE

Examples

IF MODE < > CONT THEN STOP

Analysis and advice

MODE is used to refer to the system operating mode.

Should the value of MODE be 0, the system is in the continuous operation mode. If 1, the system is in the cycle operation mode, and if 2, the system is in the segment operation mode.

When referring to the system operating mode, you may use the system constants CONT, CYCLE and SEGMENT. As you would expect, MODE = = CONT puts the system in the continuous operation mode, MODE = = CYCLE puts the system in the cycle operation mode, and MODE = = SEGMENT puts the system in the segment operation mode.

The monitor command MODE MOTION can be used to specify segment operation.

Sample program

```
PROGRAM MODESAMPLE
    MOVE A1
    MOVE A2
    IF MODE <> CONT THEN STOP
    MOVE A3
END
```

This program will stop executing itself should the system change out of the continuous operation mode.

MOTION

Purpose

The MOTION statement is used to refer to the amount of a motion that has been completed.

Format

MOTION

Examples

K = MOTION
ON MOTION > – 50 DO DOUT (1)

Analysis and advice

The MOTION statement can be used to see what percentage of a robot motion has been completed.

The "amount of motion" is defined as the percentage of a motion completed by the robot with respect to the total distance to be covered by that motion. Calculations for the amount of motion are carried out for the axis that has the greatest distance to travel.

The amount of motion is returned as a real number.
By combining the MOTION statement with an ON command, the robot can be made to send out signals while a motion is still in progress. This statement must be used in an expression.

Note: The amount of motion referenced with this command is the position commanded to the robot. Note that while the robot is moving, the current position of the robot has a delay from the command position.
Be careful because == can't be used for the comparative operator.

With ENABLE PASS when using, be careful because an infinite loop is worked depending on how to use.
Because the path orbit formation wait to P2 has occurred from P1, the following example becomes an infinite loop.
By replacing with the WAIT sentence, it is possible to avoid an infinite loop.

```
ENABLE PASS                                  ENABLE PASS
PASS=50                                      PASS=50
MOVE P1                          →           MOVE P1
LOOP1:                                       WAIT MOTION >= 95
IF MOTION < 95 THEN GOTO LOOP1               MOVE P2
MOVE P2
```

Sample program

```
PROGRAM MOTIONSMPL
    ENABLE NOWAIT
    ON MOTION > = 50 DO DOUT (1)
    MOVE A1
    ON MOTION > = 80 DO DOUT (2)
    MOVE A2
END
```

When the robot hand is 50% of the way to point A1, Signal 1 will be output. When the robot hand is 80% of the way to point A2, Signal 2 will be output.

MOTIONT

Purpose

The MOTIONT statement is used to refer to the amount of time passed since a motion has begun.

Format

MOTIONT

Examples

K = MOTIONT
ON MOTIONT > = 5 DO DOUT (1)

Analysis and advice

The MOTIONT statement can be used to see how much time has passed since a certain motion has started.

Execution time is given as a real number in units of seconds. The execution time will change to 0 when the robot has completed final positioning for that movement.

By combining the MOTIONT statement with an ON command, the robot can be made to send out signals while a motion is still in progress. When this statement monitors travel time per one movement of the robot over the specified time, it can handle the error.
The MOTIONT command must be used in an expression.

Be careful because == can't be used for the comparative operator.

Note: The amount of motion referred with the MOTION command is the position commanded to the robot. Note that while the robot is moving the current position of the robot has a delay from the command position.
Be careful because == cannot be used for the comparative operator.
With ENABLE PASS when using, be careful because an infinite loop is worked depending on how to use.
Because the path orbit formation wait to P2 has occurred from P1, the following example becomes an infinite loop.

By replacing with the WAIT sentence, it is possible to avoid an infinite loop.

```
ENABLE PASS                                  ENABLE PASS
PASS=50                                      PASS=50
MOVE P1                          →           MOVE P1
LOOP1:                                       WAIT MOTION >= 5
IF MOTION < 5 THEN GOTO LOOP1                MOVE P2
MOVE P2
```

Sample program

```
PROGRAM MOTIONTSMPL
    ENABLE NOWAIT
    ON MOTIONT > = 10 DO DOUT (1)
    MOVE A1
    MOVE A2
END
```

Should the robot take more than 10 seconds to complete a motion to A1, Signal 1 will be output immediately.

MOVE

Purpose

The MOVE command moves the robot to a specified position.

Format

MOVE <position> [WITH clause]

Examples

MOVE A1
MOVE A1 WITH SPEED = 50

Analysis and advice

The MOVE command moves the robot to the specified position in synchronous motion.

All the robot joints will start and stop moving at the same time. The controller will adjust the speeds of the joints relative to the slowest joint for that motion accordingly. This is called synchronous motion (or sometimes joint angle interpolation).

You may use a positional vector for <position>. Also, you may directly specify the coordinate values for <position> in either of the two ways shown below.

You cannot use either coordinate type data or load type data for <position>.

MOVE POINT (X, Y, Z, C, T, <configuration>)
MOVE {X, Y, Z, C, T} WITH CONFIG = <configuration>

(You should try to use the POINT command whenever possible to make it clear what data type you are handling.)

For both methods shown above:

X, Y, Z, C, T : Coordinate values X, Y, Z, C and T are specified with real numbers (in units of millimeters or degrees).

<Configuration>: The configuration of the robot is specified by an integer value of 0, 1 or 2.
(0 = undefined (FREE); 1 = left hand system; 2 = right hand system)

You may use a constant, a variable or a calculation for each individual element. However, you may not use vector-type data for an element. Anything less than 0 which is entered as the <configuration> will be treated as 0, and anything greater than 2 will be treated as 2.
Individual data elements may be omitted, but these omitted elements will all be treated as 0. For example, the two statements below mean the same thing:

MOVE POINT (100, 100, 0, 0, 0, RIGTHY)
MOVE POINT (100, 100)

(Everything from Z to <configuration> will be taken as 0.)

When entering positional data from the teaching pendant, work coordinate system data specified at the time of teaching will also be recorded. When a movement command is executed, the work coordinate system will change over to that specified at the time the positional data was taught. Note, however, that base and tool coordinates will stay as they were before the command was executed.

When directly specifying the coordinate values of <position> (sometimes along with creating or manipulating positional data with commands such as DEST, HERE and POINT), movements are performed with the work coordinate system in effect before the command was executed.

The controller will figure out movement conditions such as speed and acceleration using the system variable values in effect at the time. Should you wish to change a movement condition for one operation, use a WITH command to specify that condition.
Refer to the WITH command for more information.

Sample program

```
PROGRAM MOVESAMPLE
    MOVE A1
    MOVE A2
END
```

This program will move the robot to point A1 with synchronous motion.

MOVEA

Purpose	MOVEA moves a specified robot joint to a specified position.
Format	MOVEA <axis>, <absolute position> [WITH clause]
Examples	MOVEA 1, 60 MOVEA 3, 0 WITH GAIN = {,, ON}
Analysis and advice	The MOVEA command moves a specified robot joint to a specified position. Such movement is called "absolute single axis motion." The <axis> designation contains an integer from 1 to 5 and specifies the robot joint to be moved. All other axes besides that specified will not move. The <absolute position> designation specifies the destination of that movement relative to the origin of that axis. For rotary joints, <absolute position> is in terms of degrees. For linear (direct drive) joints, <absolute position> is in terms of millimeters. Should you specify an <absolute position> outside of the range of that joint, the robot will move to the position just before the end of that limit. Constants, variables or calculations may be used for the <axis> and <absolute position> designations. However, you may not use vector-type data. Should you use anything other than 1 to 5 for the <axis> specification, or should you designate an axis which your robot does not have, the robot does not move.

The controller will figure out movement conditions such as speed and acceleration with the system variable values in effect at the time. Should you wish to change a movement condition for one operation, use the WITH command to specify that condition. Refer to the WITH command for more information.

Sample program

```
PROGRAM MOVEASAMPL
    MOVEA 1, 0
    MOVEA 2, 90
END
```

This program will move Axis 1 to its 0 degree position.

MOVEC

Purpose

MOVEC moves the robot hand to a specified position through a specified passing position in circular interpolation.

Format

MOVEC <passing position> <position> [WITH clause]

Examples

MOVEC A1 A2
MOVEC A1 A2 WITH SPEED = 10

Analysis and advice

The tip of the robot hand is moved in a circular path connected among the current position, <passing position> and <position>. The tip of the hand is moved in the direction from the current position to <position> at a constant angular velocity.

Specify position type data to <passing position> and <position>. Like the MOVE and MOVES commands, with the POINT command, a coordinate value can be directly specified.

You cannot use either coordinate type data or load type data for <passing position> and <position>.

[Work coordinate system]
In the work coordinate system, the tip of the robot hand is moved as taught for both <passing position> and <position>.
When the work coordinate system is specified with the WITH clause, the tip of the hand will be moved work coordinate system. When a coordinate value is directly specified for <passing position> and <position>, the tip of the hand will be moved in the work coordinate system when the command is executed.

[WITH clause]
When the robot is moved, the moving conditions such as speed and acceleration will be determined depending on the setting values of the system variables at the time. To change the moving conditions for one motion, use the WITH clause.
Refer to the WITH command for more information.

[Limitation of restoration after canceling circular interpolation]
After a motion of circular interpolation is cancelled by "ON ~ BREAK ~ DO ~," feed hold, emergency stop, or a trouble, when the execution is resumed without resetting the program (when a motion is cancelled by "ON ~ BREAK ~ DO ~" and resumed with the RESUME command), the cancelled command is re-executed. At that time, the motion of circular interpolation works as linear motion of interpolation to <position>.

[Limitation of position relationship of three points]
When three or two positions of three points forming an arc (present position, <passing position>, and <position>) are the same or very close, the tip of the robot hand may be moved along an arc which differs from that expected.
When a motion of circular interpolation is used during a short-cut motion, the path of the robot should be connected to the tangent of the circular. When the angle becomes sharp, at a joint of circular interpolation of the short-cut motion, an abrupt acceleration may be applied to the robot.

[Tool offset]
When the hand is moved linearly or in circular interpolation with a tool being offset while changing the orientation of the tool, unless the tool offset is properly set, the specified motion may not be obtained.
The tool offset is used in teaching positions. Before teaching the positions, it is necessary to check that the tool offset is correctly set.

(For the selecting method of the tool coordinate system, see "6.7.6 Tool Coordinate Selection" of the "Operator's Manual.")
When a tool being offset is used, before teaching the positions, it is necessary to set the tool offset. In addition, at the beginning of the robot language program, with the TOOL command, securely specify the correct tool offset.

```
Example:  PROGRAM MAIN
          TOOL=TOOL1    Use "TOOL1" as tool offset after
                        that.
            ...
          END
```

[About the 5th axis addition (the option)]
When adding (the option) the 5th axis, be careful because it isn't possible for an arc interpolation to be worked about the 5th axis. Among the other axes, an arc interpolation is worked.

Sample program

```
PROGRAM MOVECSAMPL
    MOVES A1
    MOVEC A2 A3
END
```

Moves the hand from A1 to A2 in arc interpolation.

MOVEI

Purpose

MOVEI moves a specified robot joint by a specified amount from its present position.

Format

MOVEI <axis>, <relative position) [WITH clause]

Examples

MOVEI 1, 60
MOVEI 3, 10 WITH SPEED = 50

Analysis and advice

The MOVEI command moves a specified robot joint by a specified amount from its position at the time. Such movement is called "relative single axis motion."

The <axis> designation contains an integer from 1 to 5 and specifies the robot joint to be moved. All other axes besides that specified will not move.

The <relative position> designation specifies the amount of that movement relative to the position of that joint at the time. For rotary joints, <relative position> is in terms of degrees. For linear (direct drive) joints, <relative position> is in terms of millimeters. Should you specify an <relative position> outside of the range of that joint, the robot will move to the position just before the end of that limit.

Constants, variables or calculations may be used for the <axis> and <relative position> designations. However, you may not use vector-type data.

Should you use anything other than 1 to 5 for the <axis> specification, or should you designate an axis which your robot does not have, the robot does not move.

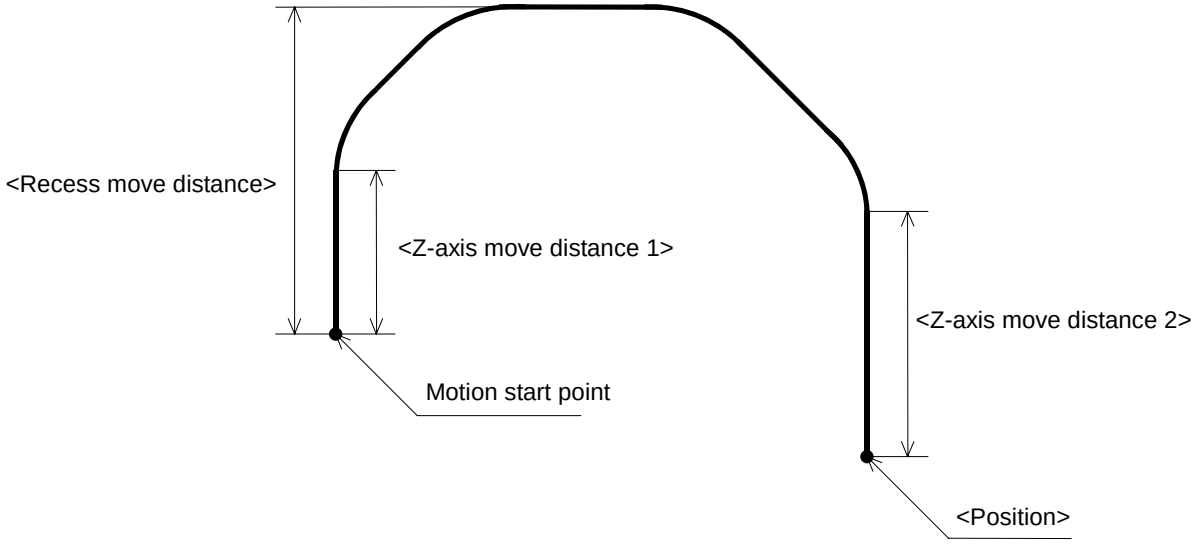
The controller will figure out movement conditions such as speed and acceleration with the system variable values in effect at the time. Should you wish to change a movement condition for one operation, use the WITH command to specify that condition. Refer to the WITH command for more information.

Sample program

```
PROGRAM MOVEISAMPL
    MOVEI 1, 30      This program will move Axis 1 to a
                     position 30 degrees from its current
                     position.
    MOVEI 2, 30      This program will move Axis 2 to a
                     position 30 degrees from its current
                     position.
    MOVEI 3, 30      This program will move Axis 3 to a
                     position 30 degrees from its current
                     position.
    MOVEI 4, 30      This program will move Axis 4 to a
                     position 30 degrees from its current
                     position.
END
```

MOVEJ

Purpose	Moves the robot to the specified position along an arch.
Format	MOVEJ <position> <definition of arch>
Examples	MOVEJ A1 AC WITH SPEED=30 MOVEJ A1 {50.0, 20.0, 30.0}
Analysis and advice	This function is used to move the robot to the specified position along an arch. Like the MOVE command, the robot moves by PTP (point-to-point) control. That is, at horizontal movement in the arch motion, the robot will not move along a straight line. <Position> specifies the final target position in the MOVEJ motion. The positional data can be used for <position>. <definition of arch> defines the profile of the arch motion by value. The positional data is used for <definition of arch>. Only three (3) elements are valid for <definition of arch>. (Values of elements 4 ~ 6 are ignored.) <definition of arch> = {<recess move distance>, <Z-axis move distance 1>, <Z-axis move distance 2>} The coordinate data and load data cannot be used for <position> and <definition of arch>.



<recess move distance> signifies the distance from the motion start point to the highest position in the Z-axis direction in units of "mm". The value of <recess move distance> should be a real number larger than 0.0. If a negative value is specified for <recess move distance>, an error is generated.

<Z-axis move distance 1> designates the move distance of axis 3 only in the up direction in units of "mm". The value of <Z-axis move distance 1> should be a real number larger than 0.0. If a negative value is specified for <Z-axis move distance 1>, an error is generated.
If <Z-axis move distance 1> is larger than the move distance in the up direction, the system interprets that the move distance in the up direction is designated.

<Z-axis move distance 2> designates the move distance of axis 3 only in the down direction in units of "mm". The value of <Z-axis move distance 2> should be a real number larger than 0.0. If a negative value is specified for <Z-axis move distance 2>, an error is generated.

If <Z-axis move distance 2> is larger than the move distance in the down direction, the system interprets that the move distance in the down direction is designated.

It is also possible to directly designate the values for <position> and <definition of arch> in the following manner.

MOVEJ POINT(X,Y,Z,C,T, <configuration>) POINT (Z1,Z2,Z3)
MOVEJ {X,Y,Z,C,T} {Z1,Z2,Z3} WITH CONFIG=<configuration>

(To identify the type of data, use the POINT command.)

X, Y, Z, C, T	:	Specify a real number for each coordinate of X, Y, Z, C, T. (Unit: mm, degree)
<Configuration>	:	Specify the robot configuration by an integer of 0 ~ 2. (0: Undefined, 1: Lefty, 2: Righty)
Z1, Z2, Z3	:	Specify a real number for defining the arch profile. (Unit: mm)

The MOVEJ command composes the movements in the up, horizontal and down directions, and will not draw an arc. Also, to give priority to the highest position in the Z-axis direction, the movements in the up and down directions are not composed. If the move distance in the up or down direction is smaller than the movement in the horizontal direction, the Z-axis move distance may be larger than the specified value.

If the configuration at the motion start point of the MOVEJ command differs from the target configuration, the configuration changes in the horizontal movement.

The value of already moved distance shown by the MOTION command at the execution of the MOVEJ command is the percentage of the lapse of time to the total movement time of the MOVEJ command. Likewise, the value of distance to go shown by the REMAIN command is the percentage of the remaining time to the total movement time of the MOVEJ command.

When the MOVEJ command has been interrupted by BREAK, the distance to go in the up direction, distance to go in the down direction, distance to go of Z-axis in each direction and target position are maintained.

To resume the operation, the MOVEJ command is created again based on these data. Therefore, if the robot has been moved by manual guide, etc. during interruption by BREAK, the midway pass cannot be assured.

The MOVEJ command cannot allow a short-cut movement in the interval with other motion command (including the MOVEJ command).

Sample program

```
PROGRAM MOVEJSAMPL
    P1 = POINT (300.0, 350.0, 50.0, 0.0, 0.0)
    P2 = POINT (300.0, –350.0, 100.0, 0.0, 0.0)
    ARCH = POINT (100.0, 40.0, 50.0)
    MOVE P1
    MOVEJ P2 ARCH
END
```

MOVEPLT

Purpose	Moves the robot to the specified position on the pallet.
Format	MOVEPLT (<pallet number>, <element number> X, Y, Z, C)
Examples	MOVEPLT (1, 10, 0, 0, 50, 0)
Analysis and advice	This function is used to move the robot to the position which is specified by the pallet number and element number and includes X, Y, Z, C offsets. The offset value zero cannot be omitted. Before executing the MOVEPLT command, appropriate **pallet should be initialized** by means of the INITPLT command. The MOVEPLT command is available in the dynamic link library. When executing this command, library build-in and global variable should be declared in the GLOBAL area. If the element number is zero (0) or less or larger than the maximum number of elements, the program stops with the following error message shown on the teach pendant display. When element number < 1: "ERR!! ELEMENT NO. IS TOO SMALL" When element No. > i × j × k of INITPLT: "ERR!! ELEMENT NO. IS TOO LARGE" For further information, see Appendix G–1.

Sample program

```
GLOBAL
  LOADLIB PALLET.LIB          Library build-in declaration.
  DIM PLTP (1, 7) AS POINT    Global variable declaration.
END

PROGRAM SAMPLE
  INITPLT (1, 3, 4, 2)        Pallet initialization to "3× 4×2"
                              with teach points PLTP (1, 1) ~
                              PLTP (1, 4).
  MOVEPLT (1, 1, 0, 0, 50, 0) Move to position of 50 mm of
                              pallet 1, element No. 1.
  OPEN1
  MOVEPLT (1, 1, 0, 0, 0, 0)  Move to pallet 1, element No. 1.
  CLOSE1
  MOVEPLT (1, 1, 0, 0, 50, 0) Move to position of 50 mm of
                              pallet 1, element No. 1.
END
```

MOVES

Purpose

The MOVES command moves the robot by linear interpolation to a specified position.

Format

MOVES <position> [WITH clause]

Examples

MOVES A1
MOVES A1 WITH GAIN = {,, ON}

Analysis and advice

The MOVES command moves the robot to the specified position along a path determined by linear interpolation.

This command will cause the robot to move from its current position to a specified position along a straight line which ties the two together. This kind of movement is called linear interpolated motion. Under such motion, the linear speed of the robot hand will remain constant (except during acceleration and deceleration).

You may use a positional vector for <position>. Also, you may directly specify the coordinate values for <position> in either of the two ways shown below.

You cannot use the coordinate type data or load type data for <position>.

MOVES POINT (X, Y, Z, C, T, <configuration>)
MOVES (X, Y, Z, C, T) WITH CONFIG = <configuration>

(You should try to use the POINT command whenever possible to make it clear what data type you are handling.)

X, Y, Z, C, T: Coordinate values X, Y, Z, C and T are specified with real numbers (in units of millimeters or degrees).

<configuration>: The configuration of the robot is specified by an integer value of 0, 1 or 2.

(0 undefined (FREE); 1 = left hand system; 2 right hand system)

You may use a constant, a variable or a calculation for each individual element. However, you may not use vector-type data for an element. Anything less than 0 which is entered as the <configuration> will be treated as 0, and anything greater than 2 will be treated as 2.

Individual data elements may be omitted, but these omitted elements will all be treated as 0. For example, the two statements below mean the same thing:

MOVES POINT (100, 100, 0, 0, 0, 0)

When entering positional data from the teach pendant, data for the work coordinate system in effect at the time will also be recorded. When a movement command is executed, the current work coordinate system will change over to that in effect at the time the positional data was taught. Note, however, that base and tool coordinates will stay as they were before the command was executed.

When directly specifying the <position> values, (sometimes along with creating or manipulating positional data with commands such as DEST, HERE and POINT), movements are performed with the work coordinate system in effect before the command was executed.

The controller will figure out movement conditions such as speed and acceleration with the system variable values in effect at the time. Should you wish to change a movement condition for one operation, use the WITH command to specify that condition. Refer to the WITH command for more information.

[About the 5th axis addition (the option)]
When adding (the option) the 5th axis, be careful because it isn't possible for a straight line interpolation to be worked about the 5th axis. Among the other axes, a straight line interpolation is worked.

Sample program

```
PROGRAM MOVESSAMPL
    MOVES A1
    MOVES A2
END
```

This program will move the robot to point A2 with linear motion.

MOVESYNC

Purpose	Specifies the motion command synchronous mode or motion command asynchronous mode.
Format	MOVESYNC
Examples	DISABLE MOVESYNC
Analysis and advice	Assume a motion during ENABLE NOWAIT in the SCOL program where multiple motion commands and signal input/output commands line up alternately. In the ENABLE MOVESYNC status (motion command synchronous mode), the system executes up to just before the next motion command and waits for the completion of positioning. Therefore, the second signal input/output is executed immediately after the second motion command starts and the third signal input/output command immediately after the third motion command starts. In this mode, however, the system does not get into the state of the system variable PASS, and short-cut motion cannot be executed. In the DISABLE MOVESYNC state (motion command asynchronous mode), the system pre-executes up to just before the maximum four motion commands ahead and waits for the completion of positioning. Therefore, the second and subsequent signal input/output commands may be executed during the first motion By enabling the system variable PASS, pass motion becomes possible. The value of this system variable when the SCOL program is actuated, is set by the user parameter [U03].

Sample program

```
PROGRAM MAIN
  ENABLE NOWAIT
  ENABLE MOVESYNC
  MOVEA 1, 90
  DOUT (1)                  D1 turns on while axis 1 is
  MOVEA 1, – 90             moving to +90° position.
  DOUT (2)                  D2 turns on while axis 1 is
  DISABLE MOVESYNC          moving to –90° position.
  MOVEA 2, 90
  DOUT (3)                  D3 turns on while axis 2 is
  MOVEA 2, –90              moving to +90° position.
  DOUT (4)                  D4 turns on while axis 2 is
END                         moving to +90° position (not
                            – 90° position).
```

NEXT

Purpose	NEXT is used in combination with the FOR statement to direct a section of the program to repeat itself for a specified number of times.
Format	NEXT [<variable>]
Examples	NEXT K
Analysis and advice	The NEXT statement is used with the FOR statement to direct a part of the program to repeat itself. The part of the program to be repeated is contained in a block starting with the FOR command and ending with the NEXT command. The block will keep on repeating itself until the condition specified by the FOR statement is satisfied. Specify the variable specified by the corresponding FOR statement for <variable>. If you do not specify <variable> in the NEXT statement, a loop is made between the nearest FOR statement (executed finally) and the NEXT statement. For the repeat conditions of the program, see the FOR command.

Sample program

```
PROGRAM NEXTSAMPLE
    FOR K = 1 TO 100
        MOVE A1
        MOVE A2
    NEXT K
END
```

The robot will repeat 100 times a shuttle operation between A1 and A2.

NOT

Purpose	NOT reverses the judgement of a logical expression.
Format	NOT <logical expression>
Examples	IF NOT DIN (1) THEN STOP
Analysis and advice	NOT reverses the judgement of a logical expression. The NOT statement must be used in an expression.
Sample program	PROGRAM NOTSAMPLE IF NOT DIN (1) THEN DOUT(1) END If input signal 1 is OFF, output signal 1 is ON.

NOWAIT

Purpose

NOWAIT is a system switch which directs the controller to continue processing I/O signals without waiting for the robot to finish positioning itself.

Format

NOWAIT

Examples

DISABLE NOWAIT
ENABLE NOWAIT

Analysis and advice

NOWAIT is a system constant used to tell the controller not to wait for the robot to finish positioning itself before processing I/O (input/output) signals.

Signal output timing is described in detail in Section 5.

The ENABLE and DISABLE commands are used to turn system switches (such as NOWAIT) on and off.

ENABLE NOWAIT tells the controller not to wait for the robot to finish positioning itself before sending out (or taking in) signals. DISABLE NOWAIT tells the controller to wait for the robot to finish positioning itself before sending out (or taking in) signals.

The initial setting for the system is DISABLE NOWAIT.

Sample program

```
PROGRAM NOWAITSMPL
    ENABLE NOWAIT
    MOVE A1
    DOUT (1)
    MOVE A2
    DOUT (2)
    MOVE A3
END
```

Here, the controller will send out external signals without waiting for the robot to finish positioning itself.

OFF

Purpose

OFF is a system constant used to specify axes for which the gain (servo control) is to be OFF.

Format

OFF

Examples

GAIN = {OFF, OFF, ON, OFF, OFF}
MOVE A1 WITH GAIN = {,, OFF}

Analysis and advice

OFF is a system constant used in conjunction with GAIN or SETGAIN in order to specify the gain (servo control) of a specific axis as off.
Should the GAIN be specified as OFF, servo control for that axis will stop the next time a movement command is executed.
Axes for which servo control has been stopped are in the "servo free state" (in which positioning control is not carried out).
As a system constant, OFF has a value of 0. If you wanted to, you could use it in your program as a constant having the value 0. However, this is not unnecessarily hard to understand.
You cannot substitute into system constants.
For information on gains, refer to the GAIN command.

Sample program

```
PROGRAM OFFSAMPLE
    MOVE A1
    WAIT MOTION > = 100
    GAIN = {OFF, OFF, ON, OFF, OFF}
    MOVE A2
    OPEN 1
    DELAY 0.5
    MOVE A1
    WAIT MOTION >= 100
    GAIN = {ON, ON, ON, ON, ON}
    READY
END
```

This program turns off all gains except that for the Z-axis (Axis 3) before the robot moves to point A2.

ON

Purpose

ON is a system constant used to specify axes for which the gain (servo control) is to be ON.

Format

ON

Examples

GAIN = {OFF, OFF, ON, OFF, OFF}
MOVE A1 WITH GAIN = {,, ON}

Analysis and advice

ON is a system constant used in conjunction with GAIN or SETGAIN in order to specify the gain (servo control) of a specific axis as ON.
Should the GAIN be specified as ON, servo control for that axis will start the next time a movement command is executed.
Axes for which servo control has been stopped are in the "servo free state" (in which positioning control is not carried out).
As a system constant, ON has a value of 1. If you wanted to, you could use it in your program as a constant having the value 1. However, this is not a good idea since it makes your program unnecessarily hard to understand.
You cannot substitute into system constants.
For information on gains, refer to the GAIN command.

Sample program

```
PROGRAM ONSAMPLE
    MOVE A1
    WAIT MOTION >= 100
    GAIN = {OFF, OFF, ON, OFF, OFF}
    MOVE A2
    OPEN 1
    DELAY 0.5
    MOVE A1
    WAIT MOTION >= 100
    GAIN = {ON, ON, ON, ON, ON}
    READY
END
```

Before moving to point A2, this program turns off all gains except that for the Z-axis (Axis 3). Then, when all motions have been completed, the program will turn the gains back on for all axes.

ON

Purpose

ON is used for monitoring conditions. (For the gain ON/OFF designation, see the descriptions on [ON] above.)

Format

ON <monitoring condition> [{BREAK | PAUSE}] DO <statement>

Examples

ON DIN (1) DO RETURN
ON TIMER DO MOVE A1

Analysis and advice

Should the <monitoring condition> be satisfied, the statement following the DO command will be executed.

Condition monitoring is carried out no matter what movement the robot happens to be doing at the time.

The ON command is processed in parallel with robot motion commands. Should a MOTION, MOTIONT, REMAIN or REMAINT command be used as the monitoring condition, monitoring of conditions for subsequent movement commands will be performed. Should TIMER be used as the monitoring condition, conditions will be monitored independently of robot movement.

When monitoring input signals with DIN or other such commands, the timing with which monitoring begins will vary depending on the setting of the NOWAIT system switch. When an ENABLE NOWAIT statement is in effect, signals will be monitored independently of robot movement.

When the DISABLE NOWAIT statement is in effect, monitoring of the signal will start after the robot has completed the movement it was executing at the time.

The execution of the statement following the DO command will start immediately after the execution of the command in effect when the monitoring condition was satisfied. However, if you happen to be executing a WAIT command at the time the monitoring condition was satisfied, the WAIT command will be cancelled immediately and program control will shift to the statement following the DO command.

There are three types of execution timing you can specify for the robot while in operation:

BREAK : BREAK will immediately stop all robot movement and shift control to the statement following the DO command.

PAUSE: The statement following the DO command is executed after the movement now in progress finishes. During arm movement, however, normal program execution continues, except for the subprogram call command, return command to main program and motion command. At execution of these commands, program execution stops until the arm has stopped.

Default: The default setting will cause the movement in progress to be completed while simultaneously executing statements following the DO command. When the statement following the DO command is a movement command, always include a BREAK or PAUSE statement in the ON command line.

If the statement following the DO command (i.e., DO statement) and the motion command in the DO statement were executed, after the arm movement has finished, program execution will restart in accordance with conditions just before the condition for the ON command was satisfied.

(Should a WAIT command have been interrupted, program execution will restart from the beginning of that WAIT command, i.e., the WAIT command will be executed again). However, should a program branch to a label have been carried out with the statement following the DO command, execution will start from the statement having that label.

Ten sets of conditions can be monitored at once. Furthermore, a maximum of four input signals may be specified with a single ON command.

When multiple monitoring conditions become true at once, the DO statement corresponding to the ON command having the highest priority is executed. This priority is determined by the order in which the ON commands were encountered in the program, with the first ON command encountered having the highest priority. DO statements corresponding to all other ON commands are ignored.

Monitoring of a condition specified by an ON command will be cancelled should execution shift to a DO statement corresponding to another ON command. Also, conditions are not monitored while program execution is halted due to a STOP command or an error.

When a subprogram is specified with a statement following DO, two or more processes described in the subprogram can be executed when the condition is satisfied. When an ON statement is used in the executing program as a statement following DO, the monitoring of the condition becomes valid just after the subprogram is returned.

When the system timer is specified as the monitoring condition, the condition is checked only when the state of the timer changes.

When monitoring an external signal, an error condition or a movement reference command (such as the amount of a motion remaining to be performed), the controller monitors the state, not the change, of that signal.

The IGNORE command will cancel the monitoring of conditions specified by an ON command. Monitoring of conditions will also stop when a condition is satisfied and a statement following a DO command is executed.

Note 1: At present, ON and DO command combinations may only be used in the ways shown below:

- ON TIMER DO <statement>
  When the timer becomes 0, execute the statement.
- ON DIN ( ) DO <statement>
  When the state of the input signal(s) in the brackets ( ) becomes as specified, execute the statement. You cannot monitor more than four signals at once with one such statement. When four or more points are specified, the extra points exceeding four points are ignored.
- ON MOTION > = <expression> DO <statement>
  Execute the statement when the amount of a motion which is to be executed next to this command exceeds the specified value. The only relational operand you can use with MOTION is >=.
- ON MOTIONT > = <expression> DO <statement>
  Execute the statement when the time required for a motion which is to be executed next to this command exceeds the specified time. The only relational operand you can use with MOTIONT is > =.
- ON REMAIN < = <expression> DO <statement>
  Execute the statement when the remaining amount of a motion which is to be executed next to this command is smaller than the specified value. The only relational operand you can use with REMAIN is < =.

- ON REMAINT <= <expression> DO <statement>
  Execute the statement when the remaining time required for a motion which is to be executed next to this command is smaller than the specified time.
  The only relational operand you can use with REMAINT is < =.

Note 2: The following command relating to the task control cannot be used in the area following the DO statement.
TASK, KILL, SWITCH

When using these commands after the DO statement, they are not executed. Note that monitoring of the conditions specified by the subtask ON command cannot be executed.

Note 3: If a motion monitored under the condition of ON MOTION, ON MOTIONT, ON REMAIN or ON REMAINT has been stopped, or if the slow speed command has been specified during execution of a monitored motion, the ON condition is cancelled.

Sample program

```
PROGRAM MAIN
    DOSAMPLE
    MOVE P
END
PROGRAM ONSAMPLE
    ON DIN (1) PAUSE DO RETURN
    MOVE A1
    MOVE A2
    MOVE A3
    WAIT MOTION > = 100
    IGNORE DIN (1)
    RETURN
END
```

Should Signal 1 turn ON while a movement is being executed, control will be returned to the main program after that movement has been completed.

Cautions on DO statement:

For ON ~ DO command, the ON conditions to be monitored and the DO statement which starts when the conditions are satisfied are registered.

```
PRGRAM MAIN
  SIG = 1
  ON DIN (1) DO INPUT SIG
  SUB
  IGNORE DIN(1)
  PRINT SIG
END
PROGRAM SUB
  MOVE P
  WAIT MOTION >= 100
END
```

In the above SCOL program, if DIN(1) is set ON during traverse to P, the DO statement cannot be executed because the variable SIG is not defined in the program SUB and there is no space for saving the variable as input by the INPUT command. In this case, the relevant DO statement can be executed normally by defining the variable SIG as the global variable.

```
GLOBAL
  SIG = 0
END
PROGRAM MAIN
  SIG = 1
  ON DIN(1) DO INPUT SIG
  SUB
  IGNORE DIN(1)
  PRINT SIG
END
PROGRAM SUB
  MOVE P
  WAIT MOTION >= 100
END
```

In the DO statement, even if the task changeover conditions are established or the SWITCH command is executed, the task cannot be changed over. If the TASK command or KILL command is executed, an error occurs.

ONGAIN

Purpose	Turns on the gain of each axis (servo control).
Format	ONGAIN (<integer>, <integer>, <integer>, <integer>, <integer>)
Examples	ONGAIN (0, 0, 1, 0, 0)
Analysis and advice	The gain of each axis (servo control) of the robot is turned on. For details of the gain, see the descriptions of the GAIN command. To turn off the gain, use the OFFGAIN command. For each joint of axes 1 to 5, specify the gains by delimiting the values of the five axes with a comma. The values are specified with 1 or 0. When 1 is specified, the gain of the axis is turned on. When 0 is specified, the state of the axis remains unchanged. The on/off state of the gain is changed after the current motion is completed. This command can be executed only when file SCOL.LIB is present in the RAM drive of the controller. In this command, the system constants ON and OFF cannot be used.

Sample program

```
PROGRAM ONGAINSMPL
    MOVE A1
    OFFGAIN (1,1,0,1,1)
    MOVE A2
    ONGAIN (1,1,1,1,1)
    MOVE A3
END
```

After the robot is moved to A1, except for the axis 3, the gains are turned off. After the robot is moved to A2, the gains of all the axes are turned on.

OFFGAIN

Purpose	Turns off the gain of each axis (servo control).
Format	OFFGAIN (<integer>, <integer>, <integer> ,<integer>, <integer>)
Examples	OFFGAIN (1, 1, 0, 1, 1)
Analysis and advice	The gain of each axis (servo control) of the robot is turned off. For details of the gain, see the descriptions of the GAIN command. To turn on the gain, use the ONGAIN command. For each joint of axes 1 to 5, specify the gains by delimiting the values of the five axes with a comma. The values are specified with 1 or 0. When 1 is specified, the gain of the axis is turned off. When 0 is specified, the state of the axis remains unchanged. The on/off state of the gain is changed after the current motion is completed. This command can be executed only when file SCOL.LIB is present in the RAM drive of the controller. In this command, the system constants ON and OFF cannot be used.

Sample program

```
PROGRAM OFFGAINSMPL
    MOVE A1
    OFFGAIN (1,1,0,1,1)
    MOVE A2
    ONGAIN (1,1,1,1,1)
    MOVE A3
END
```

After the robot is moved to A1, except for the axis 3, the gains are turned off. After the robot is moved to A2, the gains of all the axes are turned on.

OPEN1, OPEN2, OPENI1, OPENI2

Purpose

These commands open the robot hand.

Format

OPEN1
OPEN2
OPENI1
OPENI2

Examples

OPEN1
OPENI2

Analysis and advice

These commands are used to open the hand. The numbers 1 and 2 refer to Hand 1 and Hand 2.

These commands open the hand by changing the state of the output signal which controls the robot hand.

The OPEN command directs the robot to open its hand after it completes the motion in progress.

The OPENI command directs the robot to open its hand immediately.

Note that these commands will not work if the file SCOL.LIB is not in the controller RAM drive.

Also, keep in mind that there is a slight delay from when an OPEN command is output until the robot actually opens its hand.

Corresponding commands CLOSE1, CLOSE2, CLOSEI1 and CLOSEI2 are provided in order to close the hand.

These commands execute a program written in the system library (SCOL. LIB). The data of SCOL. LIB should be changed according to the robot hand specifications.

Sample program

```
PROGRAM OPENSAMPLE
    CLOSEI1
    MOVE A1
    OPEN1
    DELAY 0.5
    MOVE A2
END
```

This program opens the hand after the robot has finished moving to point A1. The robot waits 0.5 seconds until the hand is open completely after the OPEN1 command has been executed.

```
PROGRAM OPENISMPL
    ENABLE NOWAIT
    OPENI 1
    DELAY 2
    MOVE A1
    CLOSEI 1
    DELAY 2
    MOVE A2
END
```

Here, the robot will open its hand1 while moving to point A1.

OR

Purpose

OR is used to find the logical sum of two logical expressions.

Format

<logical expression> OR <logical expression>

Examples

```
IF DIN (1) OR K < = 3 THEN J = 0
WAIT DIN (5) OR TIMER==0
```

Analysis and advice

The OR statement calculates the logical sum of the expressions to the right and left. If even one of the two statements is true, OR will return a TRUE.

The OR statement must be used in an expression.

Sample program

```
PROGRAM ORSAMPLE
    FOR K=1 TO 50
    IF DIN (1) OR K==50 THEN J=1 ELSE J=0
    PRINT TP, J, CR
    NEXT K
END
```

PAI

Purpose

PAI (normally written "~" or "pi ") is a system constant having a value of 3.14159

Format

PAI

Examples

```
R = D*PAI/180
D = N*PAI*2
```

Analysis and advice

PAI is a system constant having a value beginning with 3.14159.... You can use it to represent "pi" when calculating the length of an arc, the area of a circle, etc.

You cannot substitute into system constants including PAI.

Sample program

```
PROGRAM MAIN
   PAISAMPLE (5, D)
   PRINT TP, D, CR
END
PROGRAM PAISAMPLE (R, D)
   D = R/PAI*180
   RETURN
END
```

This subprogram converts the value of the argument from radians to degrees, and sends the result back to the main program as argument D.

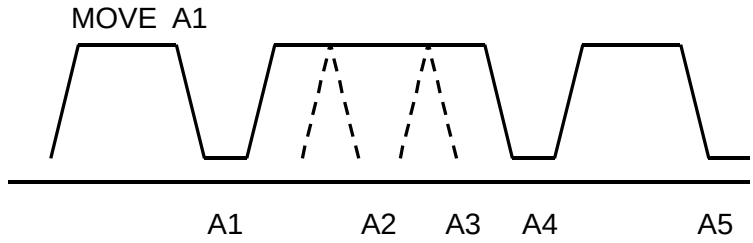
PASS

Purpose	PASS is a system switch used to specify short-cut movement. (For setting short-cut motion parameters, see the next page.)
Format	PASS
Examples	DISABLE PASS ENABLE PASS
Analysis and advice	PASS is a system switch used to invoke short-cut movement. Short-cut movement is an operating mode in which the robot is directed to begin its next move before completing the positioning of its previous move. The timing for switching over from the present movement to the next movement is specified with the system variable PASS command. Short-cut movement allows you to reduce the time it takes the robot to get from one place to another. For more information, refer to Section 5. The ENABLE and DISABLE commands are used to turn on and off system switches such as PASS. An ENABLE PASS statement will activate short-cut movement, and DISABLE PASS will cancel short-cut movement. The initial setting for the controller is DISABLE PASS.

Sample program

```
PROGRAM PASSSAMPLE
    MOVE A1
    PASS = 80
    ENABLE PASS
    MOVE A2
    MOVE A3
    DISABLE PASS
    MOVE A4
    MOVE A5
END
```

This program moves the robot with short-cut movement from A1 to A4, and for the movement from point A4 onwards, cancels the short-cut movement.



PASS

Purpose

PASS is a system variable used to set short-cut movement parameters. (For setting short-cut motion parameters, see the previous page.)

Format

PASS = <expression>

Examples

PASS = 80
PASS = PASS*0.8

Analysis and advice

PASS is a system variable used to specify parameters for short-cut movement.

Short-cut movement is an operating mode in which the robot is directed to begin its next move before completing the positioning of its previous move. The timing for switching over from the present movement to the next movement is specified with the system variable PASS command. The parameter for short-cut movement is expressed as a percentage of a motion completed by a robot relative to the entire motion. When the robot movement has exceeded that percentage, the motion being performed at that time and the following motion are superimposed.

The travel amount refer to a position that the robot is directed, at which the next movement is started even if the actual robot cannot move because of interference between the robot and the controller.

An integer value of 50 to 100 may be specified for PASS. Numbers less than 50 will be treated as 50%, and numbers greater than 100 will be treated as 100%.

The <expression> designation may contain a constant, variable, or calculation. However, you may not use vector-type data.

When referring to the PASS system variable, you can refer to the parameter of the current short-cut movement.

Short-cut movement allows you to reduce the time it takes the robot to get from one place to another. For more information, see Section 5.

The ENABLE and DISABLE commands are used to turn on and off system switches such as PASS. An ENABLE PASS statement will activate short-cut movement, and DISABLE PASS statement will cancel short-cut movement.

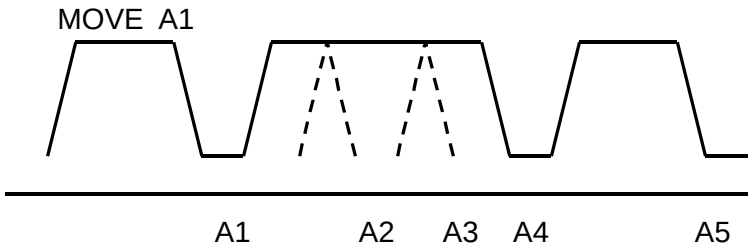
The initial setting for the controller is DISABLE PASS.

If the WAIT command and I/O command have been specified while a movement command is executed before the PASS movement stats the execution, the PASS movement may not be executed.

Sample program

```
PROGRAM PASSSAMPLE
    MOVE A1
    PASS = 80
    ENABLE PASS
    MOVE A2
    MOVE A3
    DISABLE PASS
    MOVE A4
    MOVE A5
END
```

This program moves the robot with short-cut movement from A1 to A4, and for the movement from point A4 onwards, cancels the short-cut movement.



PAUSE

Purpose

PAUSE is used to direct the controller to wait until the robot finishes a motion.

Format

ON <monitoring condition> [{BREAK | PAUSE}] DO <statement>

Examples

ON DIN (1) PAUSE DO SUB

Analysis and advice

When the monitoring conditions specified by the ON statement are established, the PAUSE directs the controller to wait until the robot finishes the motion in progress before executing the DO statement. For details, see the "ON" command.

Sample program

```
PROGRAM PAUSESMPL
    ENABLE NOWAIT
    REMARK *** MAIN PROGRAM ***
    ON DIN (24) PAUSE DO STOP
    MOVE A1
    MOVE A2
    MOVE A3
    WAIT MOTION >= 100
    IGNORE DIN (24)
END
```

Here, if something goes wrong with the system and Input Signal 24 turns ON, the robot will stop moving after completing the movement in progress at the time.

PAYLOAD

Purpose

PAYLOAD is a system variable used to set data for loads acting on the end of the robot hand.

Format

PAYLOAD = {<mass>, <center of gravity offset>}

Examples

PAYLOAD = {10, 10}
MOVE A1 WITH PAYLOAD = MOTOR

Analysis and advice

PAYLOAD is a system variable used to set data for loads acting on the end of the robot hand.

In order that the robot operate effectively under various loads, the SCOL language makes it possible to set load data which describes the mass and inertia acting on the end of the robot hand.

Loads acting on the robot hand are set with the system variable PAYLOAD. The controller uses these values to calculate control constants for robot acceleration and deceleration that are appropriate for the load.

Load data consists of values for the load mass and the load moment of inertia.

The <mass> specification designates the weight of the load applied to the tip of the robot hand in the order of kilograms. The <center of gravity offset> designates the distance between the center of gravity of the load applied to the tip of the robot hand and the center of the tool of the hand in the unit of millimeters.

Constants, variables and calculations may be used for the <mass> and <center of gravity offset> designations. Also, load-type data may be used for the {<mass>, <center of gravity offset>} specification.

Load-type data is set as shown below:

Load-type data format:
<variable> = {<mass>, <center of gravity offset>}

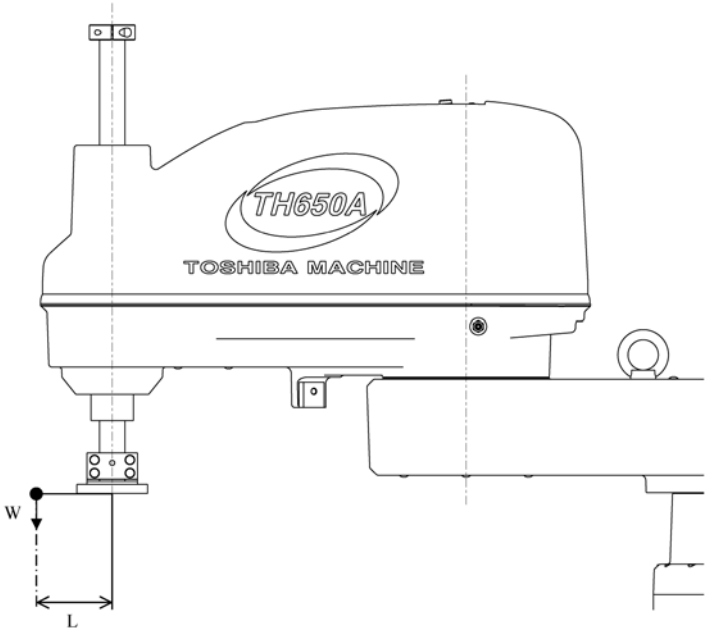
Example: MOTOR = {5, 10}
WORKA = HAND + MOTOR

The FREELOAD command is available to set load data to zero.

Sample program

```
PROGRAM PAYLOAD
    PAYLOAD = HAND
    MOVE A1
    CLOSE1
    DELAY 0.5
    MOVE A2 WITH PAYLOAD = HAND + MOTOR
    OPEN1
    DELAY 0.5
    MOVE A3
    FREELOAD
    MOVE A2
    MOVE A3
END
```

HAND is assigned as the load data before the robot moves to point A1. The robot grabs something at point A1, and the sum of HAND + MOTOR is assigned as the new load data.



Load data is entered in term of:
Mass W [unit: kg]
Center of gravity offset L [unit: mm].

POINT

Purpose

POINT creates positional type data.

Format

POINT (<expression>, <expression>, <expression>, <expression>, <expression>, <configuration>)

Examples

A = POINT (100, 100, 0, 0, 0, 0)
MOVE POINT (100, 100)

Analysis and advice

The POINT command creates positional type data.

From left to right, the <expression> designations correspond to the X, Y, Z, C and T elements. These elements are specified in units of millimeters or degrees.
The <configuration> element is to contain an integer from 0 to 2 that specifies robot configuration. The robot configuration is undefined (free) at 0, left handed at 1 and right handed at 2. In order to specify the system configuration, you may use these numbers or the system constants FREE, LEFTY and RIGHTY. As you would expect, the configuration is undefined at CONFIG = FREE, left handed at CONFIG = LEFTY, and right handed at CONFIG = RIGHTY. Anything less than 0 which is entered as the <configuration> will be treated as 0, and anything greater than 2 will be treated as 2.

Constant, variables or calculations may be used for the <expression> and <configuration> terms. However, you may not use vector-type data. Furthermore, any omitted <expression> or <configuration> terms will be taken as 0.
This command must be used in an expression.

Sample program

```
PROGRAM POINTSMPL
MOVE POINT (100, 100)
END
```

This command will move the robot to the position X = 500, Y = 500, Z = 0, A = 0, B = 0, C = 0, and T = 0.

PLCDATAR1~8 (Option of TS3000)

Purpose	These are the system variables for receiving data from the simple PLC built in the robot.
Format	PLCDATAR1
Examples	A = PLCDATAR1 B = PLCDATAR5
Analysis and advice	PLCDATAR1 ~ 8 are the read-only system variables. Reading of values set in the simple PLC is possible. (The simple PLC function is an option.) The value these system variables can receive are 0 ~ 65535. (Neither a negative value nor a decimal point can be used.)
Sample program	PROGRAM PLCDIN A=PLCDATAR1 B=PLCDATAR2 END

PLCDATAW1~8 (Option of TS3000)

Purpose

These are the system variables for writing data to the simple PLC built in the robot.

Format

PLCDATAW1 = <expression>

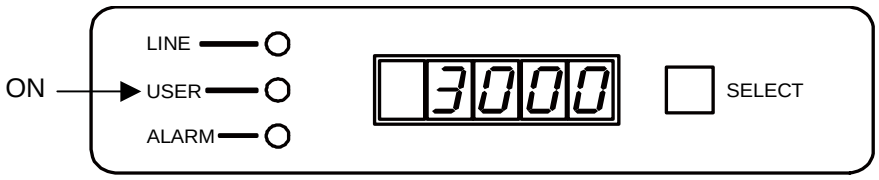
Examples

PLCDATAW1 = 1
PLCDATAW5 = A+B

Analysis and advice

PLCDATAW1 ~ 8 are the write-only system variables. Transfer of values to the simple PLC is possible. (The simple PLC function is an option.)
The value these system variables can transfer are 0 ~ 65535. (Neither a negative value nor a decimal point can be used.)

Unless the simple PLC function option is selected, only PLCDATAW1 is valid, and a written value is displayed on the 7-segment display (USER mode) on the controller front panel.



When the simple PLC function option is selected, values of PLCDATAW1 ~ 8 can be used in the sequence program at the user's discretion.

Sample program

```
PROGRAM PLCDIN
   A=10
   PLCDATAW1=1
   PLCDATAW2=A
END
```

PRINT

Purpose	The PRINT command outputs data to a specified communications channel.
Format	PRINT [{COM0 | COM1 | TP},] {<character string> | <expression>}[, {<character string> | <expression>}] ... [, CR]
Examples	PRINT "X = ", A1. X PRINT COM1, K, CR
Analysis and advice	The PRINT command is used to output data to a communication channel. Specify one (1) communication channel from COM0, COM1, and TP. COM0 and TP are channels used solely for the teach pendant. COM1 corresponds to controller COM1 communication channel. If you do not specify a communication channel in your PRINT statement, data will be output to the teach pendant communication channel. When a PRINT command is executed, the data will be output to the specified communication channel. Data contained in a <character string> will be output as it is. Data included in an <expression> will be output in solid blocks having a fixed length of 12 characters aligned on the right. Should the expression have a real value, output will consist of a real number having a maximum of four integer places and a maximum of three decimal places for a maximum total of eight spaces (counting the decimal point).

A one character space is provided in front of the number for a plus or minus sign, although the sign itself is omitted when it is +. Numbers will be pushed over to the right in the 12 character space, and any unused spaces will be left blank.

All data is in ASCII code. Should you write CR (Carriage Return) at the end of the PRINT command, output from a subsequent PRINT command will be displayed on the next line.

If you output data to the teach pendant, that data will be displayed on the teach pendant.

For information of data communication, refer to the Communication Manual.

After the moving arm has stopped, this command cannot be executed.

Sample program

```
PROGRAM PRINTSMPL
    PRINT COM0, "*** INPUT N1, N2, N3 ***"
    INPUT COM0, N1, N2, N3
    PRINT (N1 + N2 + N3)/3, CR
END
```

This program will read in three values (N1, N2 and N3) from the teach pendant, find the average, and display the average on the teach pendant.

PROGRAM

Purpose

The word PROGRAM is used to mark the beginning of a program.

Format

PROGRAM <program name> [(<variable name>, ...)]

Examples

PROGRAM SAMPLE
PROGRAM MAIN

Analysis and advice

PROGRAM SUB1 (N1, N2, N3)
PROGRAM is used to mark the beginning of a program.

The name of the program is designated by an identifier in the <program name> specification.

The program text itself is sandwiched between a PROGRAM statement and an END statement.

When designating a sub program, it is necessary to specify an argument in parentheses when required.
For details of sub programs and arguments, see "2.8 Programs."

Sample program

```
PROGRAM ENDSAMPLE
    MOVE A1
    MOVE A2
    MOVE A3
END
```

Everything from the PROGRAM statement to the END statement will be executed as a single program.

PULOUT

Purpose

The PULOUT command directs the controller to output an external signal as a pulse.

Format

PULOUT (<signal name> [, <signal name>)...)

Examples

PULOUT (1, 2, –3)
PULOUT (J, J+1, J+2)

Analysis and advice

The PULOUT command directs an output signal to be sent out as pulses having a width of 0.2 seconds.

<signal name> is to contain the number of the signal to be output. If the sign of <signal name> is positive, the signal will be modulated as OFF ~ ON ~ OFF. If the sign of <signal name> is negative, the signal will be modulated as ON ~ OFF ~ ON. Should the signal be ON and the sign of the <signal name> be positive, the signal will not be modulated. Likewise, should the signal be OFF and sign of the <signal name> be negative, the signal will not be modulated.

Up to ten <signal name> designations can be made with one PULOUT command.

You may use constants, variables or calculations for the <signal name>. However, you may not use vector-type data.

By using an ENABLE NOWAIT statement, it is possible to output pulse signals in parallel with (at the same time as) such operations as robot movement and processing of other, non-pulse output signals.

Should a DISABLE NOWAIT statement be in effect, processing of any commands which follow a PULOUT command will not begin until the pulse signal is completely output.

When the same signal is output consecutively after the execution of the PULOUT command while an ENABLE NOWAIT statement is in effect, the pulse output is not guaranteed.

Sample program

```
PROGRAM PULOUTSMPL
    DISABLE NOWAIT
    FOR K = 1 TO 16
    PULOUT (K)
    NEXT K
END
```

Output Signals 1 to 16 will turn ON one after the other at an interval of 0.2 seconds.

RCYCLE

Purpose	RCYCLE is a label used for cycle resetting.
Format	RCYCLE
Examples	RCYCLE
Analysis and advice	The RCYCLE label is used to start the execution of the main program from the first step only on the first cycle and from the step with the RCYCLE label on the second cycle and the subsequent cycles. You can program the command you wish to execute only once between the first step and the step with the RCYCLE label, for example, initialization of the counter increasing per cycle. By doing this, you can continue to pick up a workpiece from the pallet (depalletize) even if the robot has stopped during picking operation. When using the RCYCLE command, be sure to observe the following cautions. You can only use RCYCLE once in the main program. Be sure to program so that at least one (1) line describing “RCYCLE :” is executed. This function cannot be used for the multitask program. An error occurs at cycle reset. For the ON ~ DO command, this function cannot be used during execution of the DO statement. An error occurs at cycle reset. For more information on cycle resetting, refer to the Operator’s Manual.

Sample program

```
PROGRAM RCYCLESMPL
    COUNT = 0
    RCYCLE:
    MOVE A1
    MOVE A2
    COUNT = COUNT + 1
    IF COUNT < = 10 THEN GOTO RCYCLE
END
```

Here, even should execution be interrupted partway through, operation can be continued where left off since the value of counter COUNT is preserved.

READY

Purpose

The READY command returns the robot to its mechanical origin.

Format

READY

Examples

READY

Analysis and advice

The READY command moves each axis of the robot to its mechanical origin.

Note that this command will not work if you do not have the file SCOL.LIB in the controller RAM drive.

In the horizontal rotating type robot, the axes are moved in the order of axis 3, axis 4, axis 2, axis 1, and axis 5. Note that the machine zero point of axis 3 is near the lower motion limit.

Sample program

```
PROGRAM READYSMPL
    READY
END
```

This program will return the robot to its mechanical origin.

REAL

Purpose

The REAL command changes a numerical value into a real number.

Format

REAL (<expression>)

Examples

AK = REAL (–20)
N = REAL (K)
J1 = K – REAL (N – 28)

Analysis and advice

The REAL command converts the number or calculation result in the brackets ( ) to a real number.

This command is used when one wants to specify the data type of a variable as real (as opposed to integer).

You may use constants, variables or equations for <expression>. However, you may not use vector-type data.

The REAL command must be used in an expression.

Sample program

```
PROGRAM REALSAMPL
    K=REAL (0.12345)
    PRINT TP, K, CR
END
```

This program declares the data type of variable K as real.

REMAIN

Purpose

The REMAIN statement is used to refer to the amount of a motion remaining to be completed.

Format

REMAIN

Examples

K = REMAIN
ON REMAIN < = 50 DO DOUT (1)

Analysis and advice

The REMAIN statement can be used to see what percentage of a robot motion remains to be completed.

The "amount of motion remaining" is defined as the percentage of a motion not yet completed by the robot with respect to the total distance to be covered by that motion. Calculations for the amount of motion remaining are carried out for the axis that has the greatest distance to travel. REMAIN returns a real number.
By combining the REMAIN statement with an ON command, the robot can be made to send out signals while a motion is still in progress.

This statement must be used in an expression.

Note: The amount of motion referenced with the REMAIN command is the position commanded to the robot. Note that the current position of the robot has a delay to the commanded position while the robot is moving.
Be careful because == can't be used for the comparative operator

Because the path orbit formation wait to P2 has occurred from P1, the following example becomes an infinite loop. By replacing with the WAIT sentence, it is possible to avoid an infinite loop.

```
ENABLE PASS                                  ENABLE PASS
   PASS=50                                      PASS=50
   MOVE P1                       →              MOVE P1
LOOP1:                                       WAIT REMAIN < 5
   IF REMAIN > 5 THEN GOTO LOOP1                MOVE P2
   MOVE P2
```

Sample program

```
PROGRAM REMAINSMPL
    ENABLE NOWAIT
    ON REMAIN < = 50 DO DOUT (1)
    MOVE A1
    ON REMAIN < = 20 DO DOUT (2)
    MOVE A2
END
```

When the robot hand is 50% of the way to point A1, Signal 1 will be output. When the robot hand is 80% of the way to point A2 (or, in other words, when 20% of the motion remains to be completed), Signal 2 will be output.

REMAINT

Purpose

The REMAINT statement is used to refer to the amount of time remaining before a motion is to be completed.

Format

REMAINT

Examples

K = REMAINT
ON REMAINT < = 1 DO DOUT (1)

Analysis and advice

The REMAINT statement can be used to see how much time remains before a certain motion will be completed.

Remaining time is given as a real number in units of seconds. The remaining time will become 0 when the robot has completed final positioning for that movement.

By combining the REMAINT statement with an ON command, the robot can be made to send out signals while a motion is still in progress. This statement must be used in an expression.

Be careful because == can't be used for the comparative operator.

Because the path orbit formation wait to P2 has occurred from P1, the following example becomes an infinite loop. By replacing with the WAIT sentence, it is possible to avoid an infinite loop.

```
ENABLE PASS                                          ENABLE PASS
   PASS=50                                              PASS=50
   MOVE P1                              →               MOVE P1
LOOP1:                                               WAIT REMAINT < 1
   IF REMAINT > 1 THEN GOTO LOOP1                       MOVE P2
   MOVE P2
```

Sample program

```
PROGRAM REMAINTSMPL
    ENABLE NOWAIT
    ON REMAINT < = 1 DO DOUT (1)
    MOVE A1
    MOVE A2
END
```

Signal 1 will be output one second before the robot reaches point A1.

REMARK

Purpose

The REMARK statement is used to mark comments.

Format

REMARK [<comment>]

Examples

REMARK *** SCOL SAMPLE ***

Analysis and advice

Comments are used in the program to make it easier to read and understand.

REMARK statements themselves are interpreted as comments.

The comments are not executed.
The symbol (') has the same meaning as the REMARK statement. When writing a comment following other commands, this symbol (') is used. The characters following the symbol (') are all interpreted as comments.

Sample program

```
PROGRAM REMARKSMPL
    REMARK *** SAMPLE PROGRAM ***
    MOVE A1    'MOVES TO A1
END
```

“MOVES TO A1” is programmed as the comment.

RESET

Purpose

The RESET command is used to reset certain controller conditions such as the state of output signals.

Format

RESET <state> [, <state>]

Examples

RESET DOUT
RESET RESUME

Analysis and advice

The RESET command is used to reset the state of such things as the controller output signals. Only the following two statements may be used.

(1) DOUT
DOUT will turn OFF all user output signals.

(2) RESUME
RESUME will reset the robot movement suspended by a BREAK command in an ON construct. After resetting, the movement interrupted by the BREAK command cannot be resumed by the RESUME command.

Sample program

```
PROGRAM RESETSMPL
    RESET DOUT
END
```

This program will turn all user output signals OFF.

RESTORE

Purpose	This command updates the initial value of the global variable.
Format	RESTORE (“<variable>”)
Examples	RESTORE (“I”)
Analysis and advice	Change, etc. of the position data which has been taught is restored to the file. The variable which can be specified is only the global variable other than the array which does not have an initial value. If any other variable is to be restored, an error occurs at selection.
Sample program	GLOBAL A = 0 END PROGRAM STORETEST A = A + 1 RESTORE (“A”) PRINT "A=", A, CR END

RESUME

Purpose

The RESUME command restarts robot movement interrupted by a BREAK command.

Format

RESUME

Examples

RESUME

Analysis and advice

The RESUME command is used to restart robot motion suspended by a BREAK command in an ON construct.

Movement is restarted (resumed) from the location the robot movement was suspended. Therefore, should you restart in the circular interpolation mode, the path the robot takes may vary by quite a bit depending on the relation between the current position, the interpolation points and the destination.

When multiple BREAK commands have been executed (and are still in effect), only the movement interrupted by the last BREAK command may be resumed.

Should you execute a RESET RESUME statement, the suspended motion will be reset. The suspended motion will also be reset should a PAUSE command be executed by the ON command.

This command is effective only in the DO statement. If executed in a statement other than the DO statement, this command is ignored.

Sample program

```
PROGRAM RESUMESMPL
    ENABLE NOWAIT
    REMARK *** MAIN PROGRAM ***
    ON DIN (24) BREAK DO BREAKSUB
    MOVE A1
    MOVE A2
    MOVE A3
    WAIT MOTION > = 100
    IGNORE DIN (24)
END
```

Should something go wrong with the system and Input Signal 24 turn ON, the controller will stop robot movement immediately and shift program execution to subprogram BREAKSUB (shown below).

```
PROGRAM BREAKSUB
    REMARK *** SUBROUTINE ***
    WAIT DIN (–24)
    RESUME
END
```

The subprogram BREAKSUB will wait until Input Signal 24 turns OFF. When the error is resolved, the interrupted motion will be resumed.

RETURN

Purpose

The RETURN statement directs program execution to return to the main program from a subprogram.

Format

RETURN

Examples

RETURN

Analysis and advice

The RETURN statement is used to return the program from the subprogram to the main program.

Even should you forget to include a RETURN statement in your subprogram, the controller will return the control to the main program through the execution of the END statement.

You will get an error should you put a RETURN statement in your main program.

Sample program

```
PROGRAM MAIN
    RETURNSAMPLE (5, K)
    PRINT TP, K, CR
END
PROGRAM RETURNSMPL (N, K)
    K = N * N
    RETURN
END
```

This program will take argument N, multiply it by itself, and send the result back to the main program as argument K.

RIGHTY

Purpose

RIGHTY is a system constant used to change over the configuration of the robot to a right handed system.

Format

RIGHTY

Examples

```
CONFIG = RIGHTY
MOVE A1 WITH CONFIG = RIGHTY
```

Analysis and advice

RIGHTY is used in conjunction with CONFIG in order to set the robot configuration to a right handed system.

As a system constant, RIGHTY has the value of 2. If you wanted to, you could use it in your program as a constant having the value 2. However, this is not a good idea since it makes your program unnecessarily complicated.

You cannot substitute into system constants including RIGHTY.

For Cartesian coordinate robots, designation of robot configuration is ignored.

For information on robot configuration, see the CONFIG command.

Sample program

```
PROGRAM RIGHTYSMPL
    CONFIG = RIGHTY
    MOVE A1
    MOVE A2
END
```

This program will set the robot configuration to a right handed system before moving the robot on its way.

SAVEEND

Purpose	Specify the global variable preserved in the backup memory when the robot stops.
Format	SAVEEND:
Examples	SAVEEND:

Analysis and advice

When the robot stops, the specified variable is saved in the backup memory.

Please define variables that should be saved between “GLOBAL” and "SAVEEND:".

The saved data is restored after the controller is turned on at the next time.

The variables can be saved up 10kByte.

The specified global variable is saved by the following stop conditions.

- Cycle STOP
- Feed-Hold
- BREAK
- Emergency STOP
- Servo OFF

Moreover, the saved data is cleared according to the following operation timing.

- Program Select
- Program Reset
- When the main power source is turned off at movement(in the “running” status).

< Notice >

- **It takes about 100msec/1kbyte to save data.**

- Turn off the power supply after a second after the robot stops, if the saved data is large.

- The system variables are also saved. (However, TIMER variables and TID variables other than the main program are excluded.)

- About variable type that can be described in a global block, there is a limitation explained by manual "Chapter of the language" and "2.8.5 global variable definition". The variable that cannot be described in a global block by this limitation cannot be saved by this function.

Sample program

```
GLOBAL
    I=0          “I” is saved when “STOP”.
SAVEEND:
    J=0          “J” is not saved when “”STOP”.
END

PROGRAM MAIN
    FOR K=1 TO 10
        I=I+1
        J=J+1
    NEXT
    PRINT “I=",I,“J=",J,CR
END
```

Processing flow of data restoration when power OFF - ON

1. Start program "MAIN" in the cycle operation mode.
2. "I=10,J=10" is displayed on the teach pendant.
3. Program stops at the “END” command.
   ⇒"I=10" is saved into the backup memory.
4. Turn off → on the controller power.
5. Start program "MAIN".
6. "I=20,J=10" is displayed on the teach pendant.

SEGMENT

Purpose

SEGMENT is a system constant which is used to refer to the system operating mode.

Format

SEGMENT

Examples

IF MODE = = SEGMENT THEN RETURN

Analysis and advice

SEGMENT is used along with the MODE command to refer to the system operation mode. When MODE = = SEGMENT, the system is operating in the segment operation mode.

As a system constant, SEGMENT has a value of 2. If you wanted to, you could use it in your program as a constant having the value 2. However, don't do it since it will make your program hard to understand.

You cannot substitute into system constants.

The monitor command MODE MOTION can be used to specify segment operation.

For information on operating modes, see the MODE command.

Sample program

```
PROGRAM MAIN
    SEGMENTSAMPLE
END
PROGRAM SEGMENTSPL
    IF MODE = = SEGMENT THEN RETURN
    MOVE A1
    MOVE A2
    MOVE A3
END
```

If the mode is set to segment operation, program execution will return to the main program without execution.

SETGAIN

Purpose

The SETGAIN command is used to specify whether the gain (for servo control) is to be ON or OFF for each axis.

Format

SETGAIN (<integer>, <integer>, <integer>, <integer>, <integer>)

Examples

SETGAIN (0, 0, 1, 0, 0)

Analysis and advice

The SETGAIN command is used to specify whether the gain (servo control) of each axis is to be ON or OFF.

The SETGAIN command will take effect upon the completion of the robot motion being performed at the time.
For more information on gains, see the "GAIN" command.
The SETGAIN command will not work if you do not have the file SCOL.LIB in the controller RAM drive. With this command, a system constant cannot be turned on/off.

Two commands ONGAIN and OFFGAIN are provided in the library file for turning on and off the gain. Use these two commands when turning on and off the gain.

Sample program

```
PROGRAM SETGAINSMPL
    MOVE A1
    SETGAIN (0, 0, 1, 0, 0)
    MOVE A2
    SETGAIN (1, 1, 1, 1, 1)
    MOVE A3
END
```

This program turns off all gains except that the Axis 3 when the robot reaches point A1. Program will then turn on all gains when the reaches point A2.

SGN

Purpose

The SGN function returns the sign of a number value.

Format

SGN (<expression>)

Examples

S = SGN (–20.345)
N = ABS (K) * SGN (L)

Analysis and advice

The SGN function will return the sign of the <expression> in the brackets.

SGN will return 1 if the <expression> is positive, –1 if negative, and 0 if 0.

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

The SGN command must be used in an expression.

Sample program

```
PROGRAM MAIN
    SGNSAMPLE (5, 3, K)
    PRINT TP, K, CR
END
PROGRAM SGNSAMPLE (K1, K2, K)
    K = SGN (K1 – K2)
    RETURN
END
```

This subprogram will take in two arguments K1 and K2 and, as argument K, will return a 1 if K1 is bigger than K2, a 0 if the same, and a –1 if smaller to the main program.

SIN

Purpose

This function returns the sine of an entered value.

Format

SIN (<expression>)

Examples

K = SIN (60)
J1 = 1 – SIN (180 – D)

Analysis and advice

This function returns the sine of the value in the brackets ( ). Calculations are handled in units of degrees.

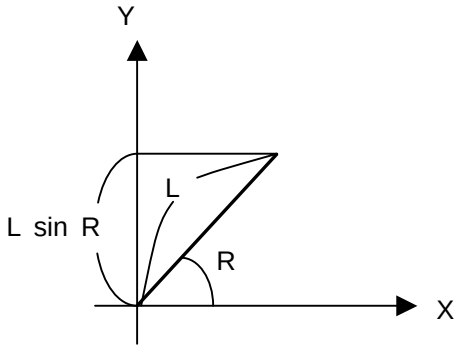
You may enter a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data.

This command must be used in an equation.

Sample program

```
PROGRAM MAIN
    SINSAMPLE (2.0, 60.0, Y)
    PRINT TP, Y, CR
END
PROGRAM SINSAMPLE (L, R, Y)
    LOOP:
        IF R > 180 THEN R = R – 360
        IF R < –180 THEN R = R + 360
        IF R > 180 OR R < –180 THEN GOTO LOOP
        Y = L * SIN (R)
        RETURN
END
```

Given (as arguments) a line segment with a length L and forming an angle R with the X-axis, this program finds the length of the Y-component of the line segment L and sends it back to the main program as argument Y.



SMOOTH

Purpose

This function designates a smooth motion.

(For the parameter setting of smooth motion, see the descriptions below.)

Format

SMOOTH

Examples

DISABLE SMOOTH
ENABLE SMOOTH

Analysis and advice

The smooth motion can be specified.
The motion command specified as SMOOTH allows the robot to move to the target position without decelerating. With the successive motion command, the robot starts moving without accelerating, irrespective of presence or absence of SMOOTH.

To turn on and off the system switch, use the ENABLE and DISABLE commands.

With the ENABLE SMOOTH command, the smooth motion starts.
With the DISABLE SMOOTH command, the smooth motion is cancelled.

The initial setting is DISABLE SMOOTH.

The smooth function is effective only for the interpolation commands (MOVES, MOVEC).

During the ENABLE SMOOTH mode, COARSE is automatically selected for the positioning accuracy.

If the speed at smooth connection has not reached the specified speed for that motion command, the robot accelerates (or decelerates) to the specified speed at maximum acceleration.

If the smooth motion command is specified while the C-axis travel distance of the smooth motion command is not enough, compared with the X, Y and Z travel distances, an error occurs.

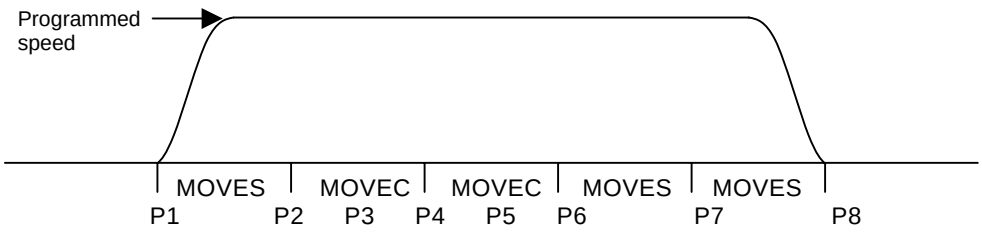
When the movement command including the travel of the T axis is specified as SMOOTH, an error occurs.

If the DISABLE SMOOTH motion command does not have a sufficient travel distance for deceleration and stop, speed control at deceleration and stop cannot be guaranteed.
The PASS function cannot be used together with the SMOOTH function. Also, the PASS motion cannot be connected with the SMOOTH motion. (See the sample programs SMPL04 and SMPL05.)

Sample program

```
PROGRAM SMPL01
  MOVE   P01
  ENABLE SMOOTH
  MOVES P02
  MOVEC P03 P04
  MOVEC P05 P06
  MOVES P07
  DISABLE SMOOTH
  MOVES P08
END
```

By connecting points P02, P04, P06 and P07 by smooth motion, the robot decelerates and stops at point P08.



(Bad example 1)

```
PROGRAM SMPL02
  PASS=50
  ENABLE SMOOTH
  MOVES P01
  MOVES P02
  MOVES P03
  ENABLE PASS        ← An alarm is generated.
                       2–039 “PASS command prohibit”
  MOVES P04
  DISABLE SMOOTH
  MOVES P05
  MOVES P06
  MOVES P07
  DISABLE PASS
  MOVES P08
END
```

If ENABLE PASS is specified in the ENABLE SMOOTH mode, an alarm occurs and the robot stops moving.

(Bad example 2)

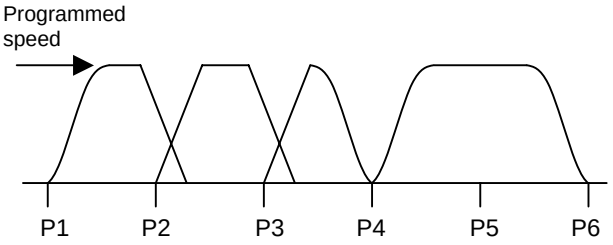
```
PROGRAM SMPL03
  PASS=50
  ENABLE PASS
  MOVES P01
  MOVES P02
  MOVES P03
  ENABLE SMOOTH      ← An alarm is generated.
                       2–040 “SMOOTH command
                       prohibit”
  MOVES P04
  DISABLE PASS
  MOVES P05
  MOVES P06
  MOVES P07
  DISABLE SMOOTH
```

```
   MOVES P08
END
```

If ENABLE SMOOTH is specified in the ENABLE PASS mode, an alarm occurs and the robot stops moving.

(Changeover from PASS to SMOOTH)

```
PROGRAM SMPL04
 PASS=50
 MOVE P01
 ENABLE PASS
 MOVES P02
 MOVES P03
 DISABLE PASS
 ENABLE SMOOTH
 MOVES P04
 MOVES P05
 DISABLE SMOOTH
 MOVES P06
END
```

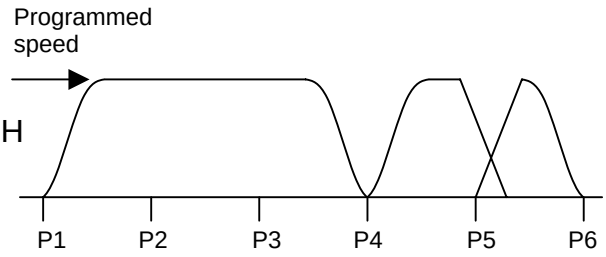


As the robot slows down and stops at the time of DISABLE PASS, SMOOTH designation for point P04 becomes invalid.

That is, points P02 and P03 are connected by short-cut, point P04 by deceleration and stop, and point P05 by smooth motion. At point P06, the robot slows down and stops. When this happens, alarm 1–018 of "Smooth connect invalid" is generated at point P04.

(Changeover from SMOOTH to PASS)

```
PROGRAM SMPL05
 PASS=50
 MOVE P01
 ENABLE SMOOTH
 MOVES P02
 MOVES P03
 DISABLE SMOOTH
 ENABLE PASS
 MOVES P04
 MOVES P05
 DISABLE PASS
 MOVES P06
END
```



As the robot slows down and stops at the time of DISABLE SMOOTH, PASS designation for point P04 becomes invalid.

That is, points P02 and P03 are connected by smooth motion, point P4 by deceleration and stop, and point P05 by short-cut. At point P06, the robot slows down and stops. When this happens, alarm 1–017 of "Pass connect invalid" is generated at point P4.

Caution: The machine service life may be affected by some operating condition of this function. Before the use, consult with us beforehand.

SPEED

Purpose

SPEED is a system variable used to specify the movement speed of the robot.

Format

SPEED = <expression>

Examples

SPEED = 50
MOVE A1 WITH SPEED = SPEED * 0.8

Analysis and advice

SPEED is a system variable which is used to specify the movement speed of the robot. It is expressed in terms of percent of the maximum speed (allowed by the controller).

SPEED is specified with a positive integer number. When a numeric value of 0 or less is specified, the specification is treated as 1. When a value of 100 or more is specified, the movement speed is suppressed to the maximum speed designated in the system.

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

By referring to this system variable, you can find the speed setting in effect at the time.

The initial setting for SPEED is 100%.

Sample program

```
PROGRAM SPEEDSMPL
    SPEED = 50
    MOVE A1
    MOVE A2
    MOVE A3 WITH SPEED = 100
    MOVE A4
END
```

Here, the robot will move to all points at 50% of full speed with the exception of point A3, to which the robot will move at full speed.

SQRT

Purpose

The SQRT function will return the square root of a given number.

Format

SQRT (<expression>)
J1 = SQRT (L1 ^ 2 + L2 ^ 2)

Examples

K = SQRT (60)
J1 = 1 – SIN (180 – D)

Analysis and advice

The SQRT function will return the square root of the value in the ( ) brackets.

You may use constants, variables or equations for the <expression> term. However, you may not use vector-type data.

When the value of <expression> is negative, an error occurs and take care.

This function must be used in an expression.

Sample program

```
PROGRAM MAIN
    SQRTSAMPLE (3, 5, L)
    PRINT TP, L, CR
END
PROGRAM SQRTSAMPLE (X, Y, L)
    L = SQRT (X ^ 2 + Y ^ 2)
    RETURN
END
```

This subprogram takes in arguments X and Y (the lengths of the two perpendicular sides of a right triangle), finds the length of the hypotenuse of that triangle, and returns that length as argument L to the main program.

STEP

Purpose

STEP is used in combination with a FOR command to specify how a loop is to repeat itself.

Format

FOR <variable> = <expression 1> TO <expression 2> [STEP <expression 3>]

Examples

FOR K = 1 TO 4 STEP 2
FOR N = K1 TO K1 + K2 STEP K3

Analysis and advice

The STEP statement is used in FOR ~ TO constructs to direct a part of the program to repeat itself a specified number of times.

The part of the program to be repeated is contained in the block starting with the FOR command and ending with the NEXT command. The block will keep on repeating itself until the condition specified by the FOR statement is satisfied.

When a FOR statement is executed, the value of <expression 1> is substituted into the <variable>. When the NEXT statement is executed, the value of <expression 3> specified by the STEP statement is added on to the <variable>. Should the value of the <variable> become greater than the value of <expression 2> at this time, the execution of the program will shift to the statement following the NEXT command. If <variable> is not greater than <expression 2>, the program execution will branch (go back) to the statement following the FOR statement.

The value of <expression 3> at the first loop is kept effective until the last loop. Therefore, even should this value be changed while executing the loop, the number of times the loop is repeated will not change.

If the value of <expression 3> is 1, you may omit the STEP statement (and everything after it) from the ON – DO construct.

A constant, variable or calculation may be used for <expression 3>. However, you may not use vector- type data.

For more information on program "looping," see the FOR command.

Sample program

```
PROGRAM STEPSAMPLE
    FOR K = 1 TO 100 STEP 2
        MOVE A1 WITH SPEED  K
        MOVE A2
    NEXT K
END
```

This program will move the robot fifty times back and forth between points A1 and A2. On each trip, the robot will speed up by 2%.

STOP

Purpose

The STOP command is used to stop execution of the program.

Format

STOP

Examples

STOP

Analysis and advice

The program will stop executing when a STOP command is encountered no matter what the system operating mode is at the time.

There is no way to restart a program thus stopped. Instead, you have to restart the program all over again.

When the program is restarted, the robot movement is restored from the subsequent step.

Sample program

```
PROGRAM STOPSAMPLE
    ENABLE NOWAIT
    ON DIN (10) DO STOP
    MOVE A1
    MOVE A2
    MOVE A3
END
```

This program will stop executing itself should Input signal 10 turn ON while the robot is moving.

SWITCH

Purpose

This command compulsively changes over to other task in the multitask operation.

Format

SWITCH

Examples

SWITCH

Analysis and advice

If the single task is effective or the system variable “SWITCH” is set to “DISABLE” and STEP command is effective, this command is invalid.

Sample program

```
GLOBAL
  MAXTASK=2
END
PROGRAM MAIN
  ID = 0
  ID = TASK("SUB1")
  LOOP:
  IF DIN(1) THEN SWITCH
  PRINT " TASK1 ",CR
  GOTO LOOP
END
PROGRAM SUB1
  ENABLE NOWAIT
  LOOP1:
  IF DIN(1) THEN SWITCH
  PRINT " TASK2",CR
  GOTO LOOP1
END
```

As the two (2) tasks use the same I/O, if DIN(1) is OFF, either task will occupy the I/O. When DIN(1) is set ON, the task is changed over compulsively to prevent one-sided occupation of I/O.

SWITCH

Purpose

This command prohibits or allows the task change-over.

Format

SWITCH

Examples

ENABLE SWITCH
DISABLE SWITCH

Analysis and advice

While this system variable is changed to DISABLE, even if the SWITCH command is executed or the task change-over conditions predetermined in the system are satisfied, the task is not changed over. The initial value of this system variable is ENABLE.

Sample program

```
GLOBAL
  MAXTASK=2
END
PROGRAM MAIN
  ID = 0
  ID = TASK("SUB1")
  LOOP:
  IF DIN(1) THEN DISABLE SWITCH
  ELSE ENABLE SWITCH
  MOVEA 1,90
  MOVEA 1,–90
  GOTO LOOP
END
PROGRAM SUB1
  ENABLE NOWAIT
  DOUT (–1,–2)
  TIMER=1
  WAIT TIMER==0
```

At the start of the main task loop, task changeover ENABLE or DISABLE is selected by the input of DIN (1).
If the task changeover is disabled, the subtask will not run and DOUT remains unchanged.

```
  DOUT (1)
  TIMER=1
  WAIT TIMER==0
  DOUT (2)
  TIMER=1
  WAIT TIMER==0
END
```

TAN

Purpose

This function returns the tangent of the number entered.

Format

TAN (<expression>)

Examples

K = TAN (60)
J1 = 1 – TAN (180 – D)

Analysis and advice

This function returns the tangent of the value in the brackets ( ).

Calculations are handled in units of degrees.

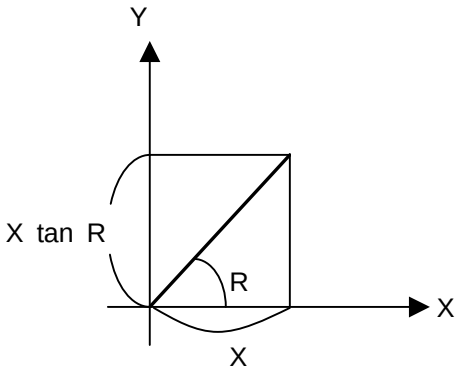
You may enter a constant, variable or calculation for the <expression> term. However, you may not enter vector-type data.

This function must be used in an equation.

Sample program

```
PROGRAM MAIN
    TANSAMPLE (2, 60, Y)
    PRINT TP, Y, CR
END
PROGRAM TANSAMPLE (X, R, Y)
    LOOP:
        IF R > 180 THEN R = R – 360
        IF R < – 180 THEN R = R + 360
        IF R > 180 OR R < – 180 THEN GOTO LOOP
        Y = X * TAN (R)
        RETURN
END
```

This program takes in the length of the X-component of the segment and the angle R forms with the X-axis (as argument R). The program then finds the length of the Y-component of the segment, and returns the result as the argument Y to the main program.



TASK

Purpose

This command executes the multitask.

Format

TASK (“<program name>”)

Examples

ID = TASK (“SUB”)

Analysis and advice

A program parenthesized starts as the task.
The return value is the number characteristic of the started subtask (task ID). The task ID is used as an argument at the stop of task.

When using the I/O command in the subtask, declare ENABLE NOWAIT in the subtask.

Sample program

```
GLOBAL
    ID=0
    MAXTASK=2
END
PROGRAM MAIN
    IF ID ==0 THEN ID= TASK (“SUB”)
    LOOP:
    MOVEA 1,90
    MOVEA 1,–90
END
PROGRAM SUB
    ENABLE NOWAIT
    WAIT DIN (1)
    DOUT (1)
    WAIT DIN (–1)
    DOUT (–1)
END
```

The subprogram starts as the task.
Asynchronous with the motion in the main task, the signal is output in reply to the signal input.

THEN

Purpose

The THEN statement is used in conjunction with an IF statement for judging conditions.

Format

IF <logical expression> THEN <statement> [ELSE <statement>]

Examples

IF DIN (1) THEN K = K + 1 ELSE K = 0

Analysis and advice

If the conditions of the <logical expression> following IF are satisfied, the <statement> following THEN will be executed. If the conditions are not satisfied, the statement following ELSE will be executed.

An ELSE statement is not mandatory in an IF ~ THEN construction. If the IF condition is not satisfied and there is no ELSE statement, the command following the IF statement is executed.

The <statement> following the THEN or ELSE statement may not contain PROGRAM, END, IF, FOR, NEXT or WAIT.

For more information on condition judgments, see the IF command.

Sample program

```
PROGRAM THENSAMPLE
    IF DIN (1) THEN K = 1 ELSE K = 0
    MOVE A1
    MOVE A2
    MOVE A3
    PRINT "K=", K, CR
END
```

Should Input Signal 1 be ON, K will equal 1.
Should Input Signal 1 be OFF, K will equal 0.

TID

Purpose

This command refers to the task ID (number) of own task.

Format

TID

Examples

MAINID = TID

Analysis and advice

This system variable is characteristic of each task started by the TASK command.

Writing of this variable is not possible.

Sample program

```
GLOBAL
    MAXTASK=2
END
PROGRAM MAIN
    TASK(“SUB”)
    LOOP:
    SWITCH
    PRINT “MAINID=”,TID,CR
    GOTO LOOP
END
PROGRAM SUB
    ENABLE NOWAIT
    SWITCH
    PRINT “SUBID=”,TID,CR
END
```

TID is expressed in the main task and subtask. Both values differ.

TIMER

Purpose

The TIMER system variable is a timer that can be changed by the system.

Format

TIMER

Examples

```
TIMER = 20
WAIT TIMER = = 0
```

Analysis and advice

TIMER is a timer that can be used inside of SCOL programs. The TIMER system variable can be set in units of seconds. When a value of 0 or less is specified, the timer will not operate correctly.

The value of the TIMER system variable counts down at the same time as it has been set. When it reaches 0, counting down cannot be executed any longer.

By referring to TIMER in your program, you can see how much time is remaining at that time in your program.

Sample program

```
PROGRAM TIMERSMPL
    TIMER = 5
    WAIT TIMER = = 0
    PRINT TIMER, CR
END
```

The program waits until the set value becomes 0 after it has been set to 5 seconds.

```
PROGRAM TIMERSMPL2
    FOR J = 1 TO 1000
        PRINT "*", CR
        TIMER = 1
        WAIT TIMER = = 0
    NEXT J
END
```

This program will display one * on the teach pendant every second for 1000 seconds.

TO

Purpose

TO is used in combination with a FOR command to specify that a portion of the program is to repeat itself *a* certain number of times.

Format

FOR <variable> = <expression 1> TO <expression 2> [STEP<expression 3>]

Examples

FOR K = 1 TO 4
FOR N = K1 TO K1 + K2 STEP K3

Analysis and advice

The TO statement is used in FOR TO constructs to direct a part of the program to repeat itself a specified number of times.

The part of the program to be repeated is contained in the block starting with the FOR command and ending with the NEXT command. The block will keep on repeating itself until the condition specified by the FOR statement is satisfied.

When a FOR statement is executed, the value of <expression 1> is substituted into the <variable>. When the NEXT statement is executed, the value of <expression 3> specified by the STEP statement is added on to the <variable>. Should the value of the <variable> become greater than the value of <expression 2> at this time, the execution of the program will shift to the statement following the NEXT command. If <variable> is not greater than <expression 2>, the program execution will branch to the statement following the FOR statement.

The values of <expression 1>, <expression 2> and <expression 3> used in the FOR construct are those in effect when the FOR statement was first executed. Therefore, even should these values be changed while executing the loop, the number of times the loop is repeated will not change.

Constants, variables and calculations may be used for <expression 1> and <expression 2>. However, you may not use vector-type data.

For more information on program "looping," refer to the FOR command.

Sample program

```
PROGRAM  TOSAMPLE
    FOR  K  =  1  TO  100
        MOVE  A1
        MOVE  A2
    NEXT  K
END
```

This program will move the robot 100 times back and forth between points A1 and A2.

TOOL

Purpose

TOOL is a system variable used to specify the tool coordinate system.

Format

TOOL

Examples

TOOL = TRANS (0, 0, 0, 0)
TOOL1 = TOOL
MOVE A1 WITH TOOL = TOOL + TRANS (,, 100)

Analysis and advice

TOOL is a system variable used to specify the tool coordinate system.
The system variable TOOL can be handled as normal coordinate-type data.
By referring to TOOL, you can find the values (location) of the current tool coordinate system.

You can directly designate values for tool offset with one of the following two methods:

TOOL = TRANS (X, Y, Z, C)
TOOL = {X, Y, Z, C}
(In order to make it clear just what kind of data type you are using, always try to use the TRANS command.)

X, Y, Z, C: X, Y, Z and C are coordinate values in real number (unit: mm or deg).

The TOOL coordinate system is created by "sliding" a distance of X, Y and Z along the respective axes of the mechanical interface coordinate system and than twisting the new Z axis by an amount C.

TOOL must be used in an expression.

Be aware that if you change tool offset within a program, there may be some misalignment between the positions as taught and the robot movement.

Sample program

```
PROGRAM TOOLSAMPLE
    MOVE A1
    MOVE A2
    TOOL = TOOL + TRANS (,, 500)
    MOVE A1
    MOVE A2
    TOOL = TRANS ( )
END
```

This program moves 500 mm along the Z axis of the tool coordinate system, and after that the robot moves to a point above 500 mm from the taught position.

TORQUE

Purpose

The TORQUE command is used to specify the limit value of torque for each axis.

Format

TORQUE = {<expression>, <expression>, <expression>, <expression>, <expression>}

Examples

TORQUE = {300, 300, 100, 300, 300}
MOVE A1 WITH TORQUE = {T, T, T, T, T}

Analysis and advice

The TORQUE command is used to specify the limit value of torque for each axis.

TORQUE is a system variable having five data elements corresponding to the five axes. The TORQUE command specifies the limit values of torque for five axes in the { } bracket following the TORQUE statement. Such data specifications are divided by commas with the first specification corresponding to Axis 1, the second to Axis 2, and so on.

The torque control value is expressed as a percentage of the torque rating of the corresponding motor. When there are no restrictions on torque, a motor may output up to 100% of its torque rating. Should torque control values be abbreviated, the controller will assume all non-specified torques to be 0.

When a value less than 100 is specified, a detection level of step-out and motor lock error will be decreased.

You may use constants, variables or calculations for the <expression> terms. However, you may not use vector-type data.

When the limit value of torque not more than 0 is specified the value is taken as 0.

Note that if the torque setting is too low, the robot will not be able to move properly and you will get an error.

Sample program

```
PROGRAM TORQUESMPL
    MOVE A1
    TORQUE = {100, 100, 100, 100, 100}
    MOVE A2
    OPEN
    DELAY 0.5
    MOVE A1
    TORQUE = {200, 200, 200, 200, 200}
    READY
END
```

This program limits the torque of the Z axis (Axis 3) to 100% of the motor torque rating as the robot approaches point A2.

TP

Purpose

The TP command is used with the PRINT and INPUT commands to specify the teach pendant as the communications channel.

Format

PRINT [{COM0 | COM1 | TP},]
{<character string> | <expression>} [,{<character string> | <expression>}] ... [,CR]

INPUT [{COM0 | COM1 | TP},]
<variable> [, <variable>] …..

Examples

```
PRINT TP, "*** INPUT N ***", CR
PRINT TP, N, N*10, CR
INPUT TP, K
```

Analysis and advice

The TP command is used with PRINT and INPUT commands to specify the teach pendant as the communications channel.

When TP is used to specify a communications channel, the controller will input and output data with the teach pendant on the communications channel (COM0) allocated to the teach pendant.

If you do not specify a communication channel for a PRINT or INPUT command, the controller will output (or input) your data to (or from) the teach pendant.

See the PRINT and INPUT commands for more information on communication processing.

Sample program

```
PROGRAM COMSAMPLE
    PRINT TP, "*** INPUT N ***", CR
    INPUT TP, N
    PRINT TP, N, CR
END
```

This program will input a number from the teach pendant and output it right back to the teach pendant.

TRANS

Purpose

The TRANS command is used to create coordinate-type data.

Format

TRANS (<expression>, <expression>, <expression>, <expression>)

Examples

A = TRANS (100, 100, 0, 0)
WORK = WORK + TRANS (100, 100)

Analysis and advice

The TRANS command is used to create coordinate-type variables.

The <expression) elements contain, from left to right, coordinate data values for X, Y, Z and C. Elements X, Y and Z are in units of millimeters and element C is in units of degrees.

You may use constants, variables or calculations for the <expression> terms. However, you may not use vector-type data. Any omitted <expression> terms will be taken as 0.

This command must be used in an expression.

Sample program

```
PROGRAM TRANSSMPL
    MOVE A1
    MOVE A2
    WORK=TRANS (,, 300)
    MOVE A1 WITH WORK=WORK1
    MOVE A2 WITH WORK=WORK1
END
```

This sets the values of the work coordinate system to Z=300, X=0, Y=0,A=0, B=0, C=0.

WAIT

Purpose

This function waits until the condition is established.

Format

WAIT <logical expression>

Examples

WAIT DIN (1)
WAIT TIMER == 0
WAIT MOTION >= 100

Analysis and advice

This function delays program execution until the condition of <logical expression> is established.

The condition is monitored, irrespective of the ongoing robot motion.

Sample program

```
PROGRAM WAITSAMPLE
    WAIT DIN (1)          Waits until input signal 1 is ON.
    MOVE A1
    MOVE A2
    MOVE A3
END
```

WITH

Purpose

The WITH statement is used to add a conditional statement to a movement command.

Format

WITH <statement> [, <statement>]...

Examples

MOVE A1 WITH SPEED = 50
MOVE A1 WITH TOOL = TOOL1, PASS = 80

Analysis and advice

The WITH statement is used to specify movement conditions corresponding to individual movement commands.

The movement conditions, such as speed, acceleration, and deceleration, are determined according to the set values of the system variables required for robot movement. In order to change movement conditions for a single motion, the corresponding movement condition is specified using the WITH clause. The movement condition specified by the WITH clause becomes valid only in the movement command where the WITH clause has been specified. The values of the system variables remain unchanged after and before the execution of the movement command.

The following movement conditions may be specified in the <statement> term. Should you wish to specify more than one such condition, be sure to keep the terms separate with commas.

Movement condition	System variable name
Robot configuration	CONFIG
Positioning accuracy	ACCUR
Acceleration (during acceleration)	ACCEL
Deceleration (during deceleration)	DECEL
Speed	SPEED
Short-cut movement parameter	PASS
Max. torque for each axis	TORQUE
Servo gain for each axis	GAIN
Tool coordinate system	TOOL
Base coordinate system	BASE
Work coordinate system	WORK
Robot load	PAYLOAD

Sample program

```
PROGRAM WITHSAMPLE
SPEED = 50
MOVE A1
MOVE A2 WITH SPEED = 100
MOVE A3
MOVE A4
END
```

This program will move to all points at half (50%) speed with the exception of point A2, to which the robot will move at full (100%) speed.

WORK

Purpose

WORK is a system variable used to specify the work coordinate system.

Format

WORK

Examples

```
WORK    TRANS (0, 0, 0, 0)
WORK1   WORK
MOVE A1 WITH WORK = WORK + TRANS (,, 100)
```

Analysis and advice

WORK is a system variable used to specify the work coordinate system. It can be handled as normal coordinate-type data. By referring to WORK, you can find the values (location) of the present work coordinate system.

You can directly designate values for work coordinates with one of the following two methods:

WORK = TRANS (X, Y, Z, C)
WORK = {X, Y, Z, C}
(In order to make it clear just what kind of data type you are using, always try to use the TRANS command.)

X, Y, Z, C: X, Y, Z and C are coordinate values in real number (unit: mm or deg).

The WORK coordinate system is created by "sliding" a distance of X, Y and Z along the respective axes of the WORLD coordinate system and then twisting the new Z axis by an amount C.

WORK must be used in an expression.

When positional data is entered with the teach pendant, the work coordinate system in effect at that time is also entered. Afterwards, when a movement command is executed using that positional data, the work coordinate system will automatically switch over to that in effect when the positional data was recorded. When you wish to move the robot in terms of two (or more) work coordinate systems (e.g., the work coordinate system in effect during teaching and a different coordinate system), you may either:

(1) Use the WITH statement to specify the work coordinate system in effect for that movement;
or
(2) Change the work coordinate data itself.

Be aware that if you change work coordinate systems within a program, there may be some misalignment between the positions as taught and the robot movement.

Sample program

```
PROGRAM WORKSAMPLE
    MOVE A1
    MOVE A2
    WORK1 = WORK + TRANS (,, 300)
    MOVE A1 WITH WORK = WORK1
    MOVE A2 WITH WORK = WORK1
END
```

The WORK command moves 300 mm along the Z axis of the work coordinate system, and after that the robot moves to a point above 300 mm from the taught position.

Section 4

Program Examples

In this section, we explain various programming examples using the SCOL language. When applying these programs to an actual robot, be sure to modify the programs accordingly to match individual robot operating conditions such as the work environment and range of movement.

(1) Program to move robot back to its mechanical origins

This program moves the robot back to its mechanical origins (zero-points for each axis). Using absolute single shaft movement, the robot will zero itself starting with the Z-axis (Axis 3) and then working in from the tip to the base.

```
PROGRAM ORIGIN
    MOVEA 3, 0
    MOVEA 5, 0
    MOVEA 4, 0
    MOVEA 2, 0
    MOVEA 1, 0
END
```

(2) Warm-up program

This program is used to warm up the robot before beginning work. The robot will start out slowly and gradually speed up.

```
PROGRAM WARMINGUP
  FOR K = 1 TO 100
    SPEED = K
    MOVEA 3, 100
    MOVEA 3, 0
    MOVEA 5, 50
    MOVEA 5, 0
    MOVEA 4, 50
    MOVEA 4, 0
    MOVEA 2, 50
    MOVEA 2, 0
```

```
    MOVEA 1, 50
    MOVEA 1, 0
  NEXT K
END
```

(3) Robot motion

A program for causing the robot to be moved from position A1 to position A2. The robot movement speed is set to 20 % of the maximum speed.

(a) When the PTP motion (MOVE command) is used:

```
PROGRAM MOVEA1A2
  SPEED=20
  MOVE A1
  MOVE A2
END
```

(b) When the linear interpolated motion (MOVES command) is used:

```
PROGRAM MOVESA1A2
  SPEED=20
  MOVES A1
  MOVES A2
END
```

(4) I/O signals

A program where turning on input signals 1 to 4 causes output signals 1 to 4 to be turned on, while turning off input signals 1 to 4 causes output signals 1 to 4 to be turned off.

```
PROGRAM SAMPLE
  IF DIN(1) THEN DOUT(1) ELSE DOUT(–1)
  IF DIN(2) THEN DOUT(2) ELSE DOUT(–2)
  IF DIN(3) THEN DOUT(3) ELSE DOUT(–3)
  IF DIN(4) THEN DOUT(4) ELSE DOUT(–4)
  END
```

(5) Interlock

A program for stopping the motion of the robot while input signal 1 is turned off in positions A1 to A4. The movement speed of the robot is set to 20% of the maximum speed.

(a) When WAIT command is used:

```
PROGRAM SAMPLE
   SPEED=20
   WAIT DIN(1)
   MOVE A1
   WAIT DIN(1)
   MOVE A2
   WAIT DIN(1)
   MOVE A3
   WAIT DIN(1)
   MOVE A4
END
```

(b) When the robot motion is stopped with the ON command and waits until input signal 1 is turned on:

```
PROGRAM SAMPLE
   SPEED=20
   ENABLE NOWAIT
   ON DIN(–1) BREAK DO SUB
   WAIT DIN(1)
   MOVE A1
   MOVE A2
   MOVE A3
   MOVE A4
END
PROGRAM SUB
   WAIT DIN(1)
   RESUME
   ON DIN(–1) BREAK DO SUB
END
```

(6) Pick and place

A program for picking up a workpiece at position A1 and placing it at position A2. The hand is opened/closed with output signal 201. The movement speed of the robot is set to 20 % of the maximum speed.

(a) When positions just above positions A1 and A2 are taught as A3 and A4, respectively:

```
PROGRAM SAMPLE
    SPEED=20
    DOUT(–201)
    MOVE A3
    MOVE A1
    DOUT(201)
    DELAY 0.5
    MOVE A3
    MOVE A4
    MOVE A2
    DOUT(–201)
    DELAY 0.5
    MOVE A4
END
```

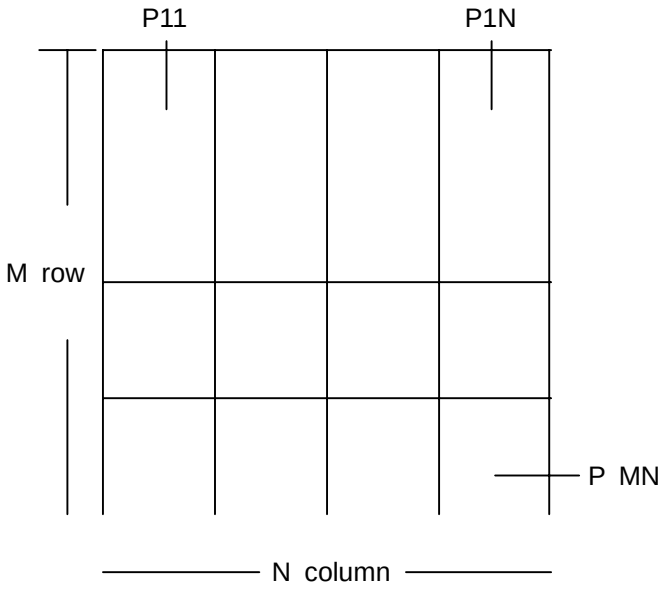
(b) When the robot is moved to a position over the taught position in the format of "MOVE A1+POINT(0,0,20)".

```
PROGRAM SAMPLE
    SPEED=20
    DOUT(–201)
    MOVE A1+POINT(0,0,20)
    MOVE A1
    DOUT(201)
    DELAY 0.5
    MOVE A1+POINT(0,0,20)
    MOVE A2+POINT(0,0,20)
    MOVE A2
    DOUT(–201)
    DELAY 0.5
```

```
    MOVE A2+POINT(0,0,20)
END
```

(c) When a short-cut motion is executed:

```
PROGRAM SAMPLE
    SPEED=20
    DOUT(–201)
    ENABLE PASS
    PASS=80
    MOVE A1+POINT(0,0,50)
    DISABLE PASS
    MOVE A1
    DOUT(201)
    DELAY 0.5
    ENABLE PASS
    MOVE A1+POINT(0,0,50)
    MOVE A2+POINT(0,0,50)
    DISABLE PASS
    MOVE A2
    DOUT(–201)
    DELAY 0.5
    ENABLE PASS
    MOVE A2+POINT(0,0,50)
END
```



(7) Palletize

Consider a program for placing parts on a pallet as shown in the right hand figure.

Pallet size:

M rows x N columns

Teaching positions:

P11: Position of column 1 and row 1 of pallet
P1N: Position of row 1 and column N of pallet
PMN: Position of row M and column N of pallet
PP: Part unloading position
PO: Standby position

Designation of I/O signals:

DI1: Pallet standby (This signal is turned on when the pallet is in the operating position)

DI2: Part standby (This signal is turned on when a part can be picked up.)

DO1: Hand close (This signal is turned on when the hand is closed.)

DO2: One pallet operation completion (This signal is turned on when all parts are placed on the pallet. When the pallet is removed and then the pallet standby signal is turned off.)

```
PROGRAM PALLET
   MOVE P0
   RESET DOUT
   M=10                              Specifies the number of lines and rows.
   N=15
   RX=(P1N.X–P11.X) /(N–1)           Computes the shift amount per element.
   RY=(P1N.Y–P11.Y)/(N–1)
   LX=(PMN.X–P1N.X)/(M–1)
   LY=(PMN.Y–P1N.Y)/(M–1)
   PASS=80
   ENABLE PASS
   ENABLE NOWAIT
   L=0
   LOOPL:                            Repeats the operation for each line.
      R=0
   LOOPR:                            Repeats the operation for each row.
      MOVE PP+POINT(0,0,50)
      WAIT DIN(2)
      MOVE PP
      WAIT MOTION>=l00
      DOUT(1)                        Unloads the part.
      DELAY 0.5
      MOVE PP+POINT(0,0,50)
      MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY, 100)
      WAIT DIN(1)
      MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY)
      WAIT MOTION>=100
      DOUT(–1)                       Places the part.
      DELAY 0.5
      MOVE P11+POINT(R*RX+L*LX,R*RY+L*LY, 100)
   R = R + 1
      IF R<=N–1 THEN GOTO LOOPR
      L = L + 1
      IF L<=M–1 THEN GOTO LOOPL
      WAIT MOTION>=l00
      DOUT(2)
```

```
        WAIT DIN(–2)
        MOVE P0
END
```

(8) Creating a program for monitoring an insertion error

A program for detecting a part insertion error and for an error handling is as follows.

A1: Position to insert the part

```
ACCUR=COARSE                        Sets the positioning accuracy to coarse.  [1]
MOVE A1+POINT(0, 0, 50)             Moves the hand to a position 50 mm just over
                                    the part insertion point.
SETGAIN(0, 0, 1, 0, 0)              Turns off the servo controls except for axis Z.
                                    [2]
TORQUE={300, 300, 50, 300, 300}     Limits the torque of axis Z to 50%.  [3]
MOVE A1                             Inserts all the parts.

SETGAIN (0, 0, 0, 0, 0)  ⎫
MOVE HERE                ⎭          Turns off the servo control for all the axis. [4]

SETGAIN (0, 0, 1, 0, 0)  ⎫
MOVE HERE                ⎭          Turns on the servo control for axis Z.  [5]

HS=HERE.Z–A1.Z           ⎫
IF HS<5 THEN GOTO OK     ⎭          When the difference of the height between
                                    the current position and the inserted position
                                    is less than 5 mm, it is determined that the
                                    inserted position is normal.  [6]

SETGAIN(1, 1, 1, 1, 1)   ⎫
DELAY 0.1                ⎪
TORQUE={300, 300, 300, 300, 300}⎪   Turns on the servo controls for all the axis,
DELAY 0.1                ⎪          returns the torque to 300% and moves the
MOVE A1+POINT(0, 0, 50)  ⎭          hand to a position 50 mm just over the
                                    part insertion point.
```

Process in the case that the parts have been abnormally inserted.

OK:

```
SETGAIN(1, 1, 1, 1, 1)
DELAY 0.1
TORQUE={300, 300, 300, 300, 300}
DELAY 0.1
MOVE A1+POINT(0, 0, 50)
```

Turns on the servo controls for all the axes, returns the torque to 300% and moves the hand to a position 50 mm just over the part insertion point.

Process in the case that the parts have been normally inserted.

Comments for program description:

[1] Unless the positioning accuracy is set to coarse, the robot system may detect an error and thereby cause the automatic operation to be stopped.

[2] By turning off the servo controls except for axis Z, the robot can be freely moved on the X–Y plane. Thus, when workpiece or part has been chamfered, it can be inserted along the chamfered surface.

[3] With the TORQUE command, the torque which occurs in the motor is limited. When an insertion error occurs, the force applied to the work and hand is suppressed. (Normally, the torque is set to 300 % of the rating value.)
By suppressing the torque to less than 100 %, the error detection level of the controller will be decreased. Thus, when monitoring the insertion error, it is necessary to set the torque to less than 100%. When the torque setting value is excessively decreased, since the torque necessary for moving the robot cannot be obtained, an error takes place.

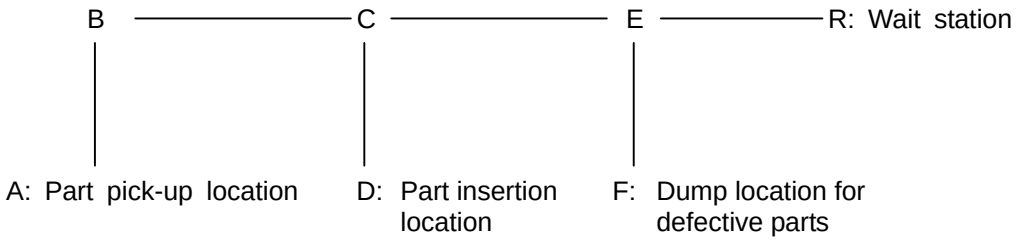
[4] The position being read with the HERE command is the position commanded to the robot. To read the present position of the robot with the HERE command, it is necessary to turn off the servo control and then turn it on again. Thus, with the GAIN command, the servo controls for all the axes are turned off.

[5] The servo control which was turned off in □ is turned on.

[6] The difference of height between the current position and the insertion position should be specified to a proper value.

(9) Program example of short-cut (PASS) movement

The following programming example uses short-cut movement to carry out "pick and place" operation.



The robot will take a part from the part pick-up location (A), move through points B and C, and try to insert the part at the part insertion location (D). If the part is defective (i.e., cannot be inserted), the robot will move through points C and E, and place the part at the dump location for defective parts (F).

Short-cut movement is used throughout the program.

The following signals are used in this program:

Input signals:

DI1: Completion of pick-up preparation
Turns ON when preparation of the part to be picked up is completed.

DI2: Completion of insertion preparation
Turns ON when preparation of the part insert position is completed.

DI3: Defective part
Turns ON when the grasped part is determined to be defective.

DI4: Completion of defective part dumping preparation
Turns ON when preparation of the part dump position is completed.

Output signals:

DO1: Completion of part pick-up
Turns ON when part pick-up is completed and preparation of the next part may be begun.

DO2: Completion of part insertion
Turns ON when part insertion is completed and preparation for the insertion of the next part may be begun.

DO3: Completion of defective part dumping
Turns ON when dumping of a defective part is completed and preparation for dumping the next defective part may be begun.

The example program is presented below:

```
PROGRAM PICKPLACE

*  PICK AND PLACE SAMPLE PROGRAM *

 INPUT:                          Initial settings
    OPEN1
    DOUT (1, 2, 3)               Part pick-up, part insertion
    ENABLE NOWAIT                Completion of disposal for defective part
    SPEED = 80
    PASS = 80
    ENABLE PASS
    MOVE B                       Move to position above part pick-up location

 PICKUP:                         Section for picking up part
    DOUT (–1)                    Begin picking up part
    WAIT DIN (1)                 Wait until pick-up preparations are completed
    MOVE A                       Lower down to A
    WAIT MOTION > = 100
    CLOSE1                       Close hand
    DELAY 0. 3
    MOVE B                       Move up to B

 PLACE:                          Section for placing part
    MOVE C                       Move to position above part insertion location
    DOUT (1, –2)                 Part pick-up complete – begin part insertion
    WAIT DIN (2)                 Wait until insertion preparations are completed
    MOVE D                       Lower down to D
    WAIT MOTION > = 100
    IF DIN (3) THEN GOTO ERR     Judgement of part as defective
    OPEN1                        Open hand
    DELAY 0. 3
```

```
        MOVE C                          Move up to C
        MOVE B                          Move up to position above part pick-up location
        DOUT (2)                        Completion of part insertion
        IF MODE = = CYCLE THEN GOTO CYCLEEND
                                        Mode check
        GOTO PICKUP
    ERR:
        MOVE C                          Section for handling defective part
                                        Move to position above part insertion location
        MOVE E                          Move to position above defective part dump
                                        location
        DOUT (–3)                       Begin dumping of defective part
        WAIT DIN (4)                    Wait until dumping preparations are completed
        MOVE F                          Lower down to F
        WAIT MOTION > = 100
        OPEN1                           Open hand
        DELAY 0. 3
        MOVE E                          Move up to E
        MOVE B                          Move to position above part pick-up location
        DOUT (3)                        Completion of dumping of defective part
        IF MODE = = CYCLE THEN GOTO CYCLEEND
                                        Mode check
        GOTO PICKUP
    CYCLEEND:                           Processing for end of cycle
        MOVE R                          Move to wait station
END
```

Section 5

Programming Hints and Warnings

This section explains timing considerations, things not to do, and things to watch out for when writing up a program.

5.1 Program Execution Timing

Robot programs are executed one line at a time starting from the top. Normally, when a robot carries out a movement command, the next command is not executed until final positioning for that movement is completed. However, when handling I/O signals and other operations not directly related to robot movement, this causes time to be wasted (waiting for the movement to be completed).

In order to efficiently utilize time and speed up the robot as much as possible, the SCOL language allows you to input/output signals and to process communications while the robot is moving.

5.1.1 Arm Movement and Signal I/O Timing

When inputting or outputting a signal, the robot program instructions written in the robot language specify whether or not to wait for the arm to complete the motion.

The system switch NOWAIT is used to tell the controller whether to wait for the robot to stop moving (finish positioning itself) before inputting or outputting signals.

ENABLE NOWAIT	This tells the controller not to wait for the arm to finish positioning itself before inputting or outputting signals.
DISABLE NOWAIT	This tells the controller to wait for the arm to finish positioning itself before inputting or outputting signals.

The initial (default) setting is DISABLE NOWAIT.

An example program is shown below. Using this program, we will describe I/O signal timing relative to command processing and arm movement.

Example: Let's consider the case of executing the following commands.

```
PROGRAM SAMPLE
  MOVE P1
  DOUT (1)
  MOVE P2
  DOUT (2)
  MOVE P3
  DOUT (3)
  MOVE P4
  DOUT (4)
  MOVE P5
  DOUT (5)
END
```

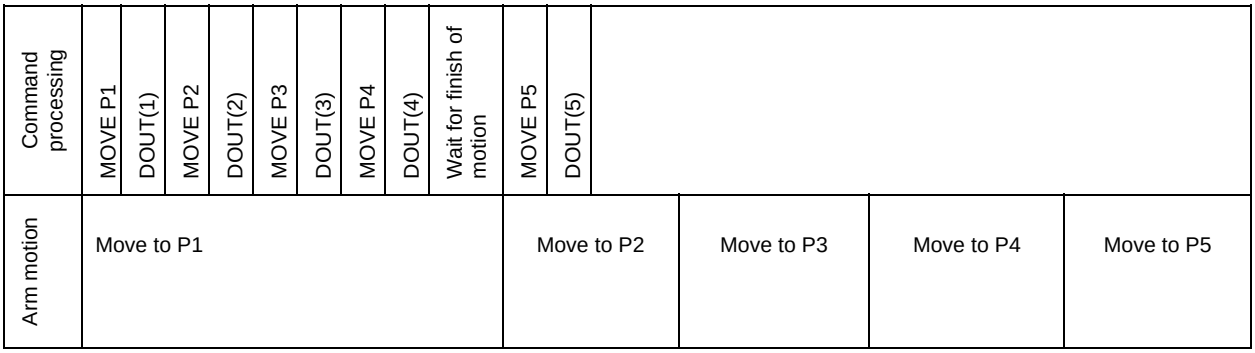
(1) Timing when DISABLE NOWAIT is in effect

Command processing	MOVE P1	Wait for finish of motion	DOUT(1)	MOVE P2	Wait for finish of motion	DOUT(2)	MOVE P3	Wait for finish of motion	DOUT(3)	MOVE P4	Wait for finish of motion	DOUT(4)	MOVE P5	Wait for finish of motion
Arm motion	Move to P1		Move to P2			Move to P3			Move to P4			Move to P5		

As shown above, signals are output after the arm has stopped moving.

Note: When ACCURE=COARSE is specified, since the subsequent command is executed before the positioning operation is completed, a signal may be output before the robot motion is completely stopped.

(2) Timing when ENABLE NOWAIT is in effect



The arm motion by the previous command completes, and the arm motion by the next command starts. Even if the arm is moving, however, processing of signal output described next to that motion command is executed, which is called "pre-reading of motion command." This robot controller pre-reads up to four (4) motion commands. (While the arm is moving to point P1, the controller pre-reads motion commands up to point P4 and waits for completion of the motion just before the arm executes the motion command to point P5.)

5.1.2 Synchronization of Arm Movement and Program Execution

In the preceding paragraph, we explain that the ongoing arm motion is executed in parallel with signal input/output. To explain it in more detail, the arm motion and SCOL commands other than the arm motion command can be executed at the same time.

In the normal SCOL program, therefore, processing of the program is executed apparently prior to the arm motion, because a special procedure (i.e., "WAIT MOTION >= 100" command) is required to synchronize the arm motion with program execution. This command will not complete until the arm has been located. Describe "WAIT MOTION >= 100" before the command to be executed in synchronization.

An example program is shown below. Using this program, we will describe I/O signal timing relative to command processing and arm movement.

Example: Let's consider the case of executing the following commands.

```
PROGRAM SAMPLE
  MOVE P1
  K = 1
  DOUT (1)
  K = K + 1
  WAIT MOTION >= 100
  K = K – 1
  DOUT (2)
END
```

(1) Timing when DISABLE NOWAIT is in effect

Command processing	MOVE P1	K=1	Wait for finish of motion	DOUT(1)	K=K+1	WAIT MOTION>=100	K=K–1	DOUT(2)
Arm motion	Move to P1							

The DOUT command waits until the arm movement finishes.

(2) Timing when ENABLE NOWAIT is in effect

Command processing	MOVE P1	K=1	DOUT(1)	K=K+1	WAIT MOTION>=100	K=K–1	DOUT(2)			
Arm motion	Move to P1									

The command of “WAIT MOTION >=100” waits until the arm movement finishes.

Note: In addition to signal input and output, processing of commands other than those given below goes on, irrespective of the robot movement.

DIN BCDIN INPUT PLCINPUT
DOUT BCDOUT PULOUT PRINT PLCPRINT

5.1.3 DELAY Command and WAIT Command

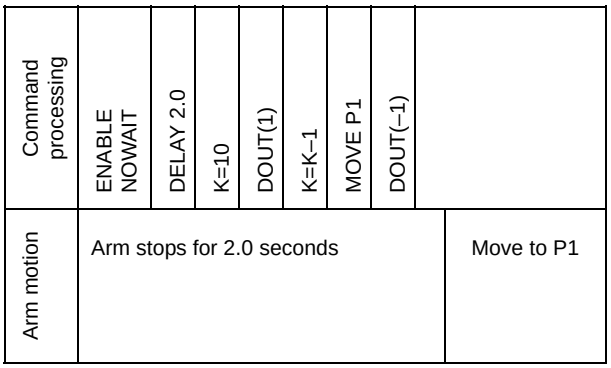
To stop the arm motion for the specified time during program execution, there are two ways. One method utilizes the DELAY command and the other method utilizes the WAIT command in conjunction with the TIMER command. When writing a program, you should keep in mind that there is a difference in execution timing between the two.

(1) DELAY command

The DELAY command is a kind of motion control commands and stops the arm motion for the predetermined time. However, as the SCOL language program is processed in synchronization with ongoing arm motion, the program is executed even during execution of the DELAY command.

Example: Let's consider the case of executing the following commands.

```
PROGRAM SAMPLE
  ENABLE NOWAIT
  DELAY 2.0
  K = 10
  DOUT (1)
  K = K – 1
  MOVE P1
  DOUT (–1)
END
```



Since the program continues to be executed even after the DELAY command causes the robot arm to pause, Signal 1 will be output while the arm is stopped.

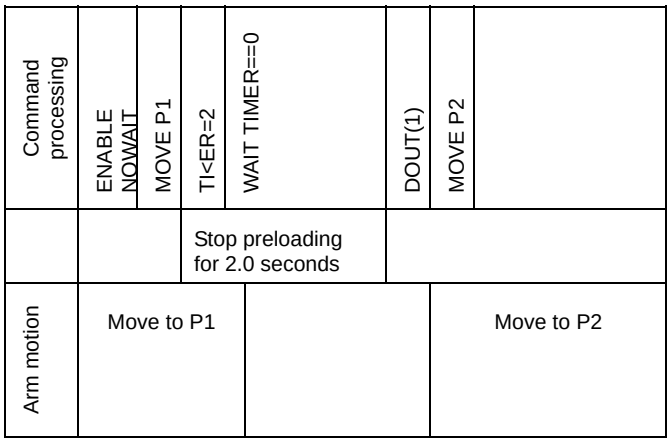
Should you wish to delay program execution as well, you should insert a “WAIT MOTION >= 100” command between the DELAY command and DOUT command.

(2) WAIT command

The WAIT command can be used with the TIMER command to delay program execution for a specified period of time. However, as with the DELAY command described above, the program is processed in parallel with the robot movement and one must be careful of the execution timing.

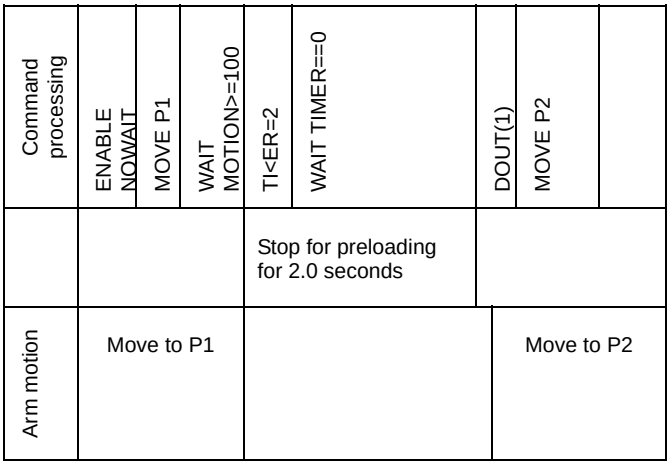
Example: Let's consider the case of executing the following commands.

```
PROGRAM SAMPLE
    ENABLE NOWAIT
    MOVE P1
    TIMER = 2
    WAIT TIMER = = 0
    DOUT (1)
    MOVE P2
END
```



The delay in program execution specified by the WAIT command comes to an end while the robot is still moving. When it is necessary for the robot to wait also, insert a WAIT MOTION statement as shown below:

```
MOVE P1
WAIT MOTION >= 100
TIMER = 2
WAIT TIMER = = 0
```



5.2 Things Not to Do When Programming

This paragraph presents restrictions and prohibitions in effect when writing programs. Refer to the command descriptions in Section 3 for information on individual commands.

5.2.1 Variables

(1) Referring to undefined variables

The data type of a variable is first defined when something is substituted into it. Therefore, do not refer to variables which appear for the first time in your program. If you do refer to a variable which has not been used (substituted into) previously, the variable data type and values become undefined. You will be very sorry when you try to debug your program.

Example:

```
PROGRAM SAMPLE
    IF K < > 0 THEN GOTO RESTART
    K = 1
    A1 = A
RESTART:
    FOR N = 1 TO 3
        MOVE A1
        A1 = A1 + POINT (100, 100)
    NEXT N
END
```

Here, when the program is first executed, variable K is referenced in the IF statement. However, variable K has never been used (substituted into) before, so it becomes undefined. After that, 1 is substituted in K, and so the variable K takes the value 1.

5.3 Things to Watch Out for When Writing a Program

This paragraph describes things to watch out for and presents an outline of some SCOL commands which may come in useful when writing a program.

5.3.1 Types of Commands

A functional classification of the SCOL language was presented in Section 1. Here, we describe the SCOL language in terms of internal processing.

SCOL commands can be broken down into basic commands, functions and system variables. Basic commands make up the core of the SCOL language and are executed in conjunction with parameters following the commands. Functions are provided as a convenience to make SCOL easier to use. System variables are used to directly refer to (and change) such items as speed, coordinate systems, etc. They can be handled like any other variable.

These three types of commands are described in more detail below.

(1) Basic commands

Basic commands provided by the SCOL language are shown below. Unlike the functions listed below in Para. (2), you may not use any of these inside of a (mathematical) expression.

MOVE	MOVES	MOVEC	MOVEA
MOVEI	MOVEJ	DELAY	BREAK
PAUSE	RESUME	ON ~ DO	IGNORE
GOTO	RETURN	WAIT	STOP
IF ~ THEN ~	ELSE	FOR ~ NEXT	
PRINT	INPUT	ENABLE	DISABLE
PROGRAM	END	RESET	REMARK
DIM ~ AS	TASK	KILL	SWITCH
GLOBAL	MAXTASK		
RESTORE	MOVESYNC	SAVEEND	RCYCLE

(2) Functions

As opposed to basic commands, functions have the feature that they can pass arguments back and forth just like subroutine programs (subprograms). Up to ten arguments can be specified.

Functions can be broken down into built-in functions and system library functions (which are so called because they are kept in a file in the system library). In the SCOL language, the system library is called SCOL.LIB, and unless SCOL.LIB is in the system RAM drive, you will not be able to use any of the system library functions.

The following built-in functions are provided by SCOL.

MOTION	MOTIONT	REMAIN	REMAINT
HERE	DEST	POINT	TRANS
DIN	DOUT	PULOUT	BCDIN
BCDOUT	SIN	COS	TAN
ASIN	ACOS	ATAN	ATAN2
SQRT	ABS	SGN	INT
REAL	LN	LOG10	EXP
MODE			

Built-in functions may be used in (mathematical) expressions with the exception of the following signal I/O commands: DOUT, PULOUT and BCDOUT.

The following system library functions are provided by SCOL (and are contained in the system library).

OPEN1	CLOSE1	OPENI1	CLOSEI1
OPEN2	CLOSE2	OPENI2	CLOSEI2
READY	ONGAIN	OFFGAIN	FREELOAD
SETGAIN			

If you make a subprogram and give it the same name as a function contained in the system library, this new subprogram should be executed as a priority.

(3) System variables

System variables are used to change system conditions and can be handled just like any other variable.

The following system variables are provided by SCOL.

CONFIG	ACCUR	ACCEL	DECEL
SPEED	PASS	TORQUE	GAIN
TOOL	BASE	WORK	TIMER
ERROR	PAYLOAD	TID	

5.3.2 Robot Coordinate Systems

This paragraph describes robot coordinate systems handled by the SCOL language.

(1) Robot coordinate systems

The robot has five types of coordinate systems, i.e. the world coordinate system, the base coordinate system, the work coordinate system, the mechanical interface coordinate system and the tool coordinate system. A brief description of these systems is presented below:

(a) World coordinate system (absolute coordinate system)

The world coordinate system is a single system used to describe the orientation of the work site surrounding the robot. If this coordinate system can be defined independently of the robot.

If this coordinate system is set at the home position of the robot, the world coordinate system and the base coordinate system are the same.

(b) Base coordinate system (mechanical coordinate system)

The base coordinate system is the system used by the robot itself. The location of the base coordinate determine is determined by the robot design, with the origin of the system always being the mechanical origins of the robot.

(c) Work coordinate system

The work coordinate system is defined in terms of the workpiece to be handled by the robot.

(d) Mechanical interface coordinate system
The mechanical interface coordinate system is defined in terms of an end effector attached to the robot. This coordinate system will shift in accordance with robot movement.

(e) Tool coordinate system
The tool coordinate system is defined in terms of the end of a tool attached to the robot. This coordinate system will shift in accordance with robot motion.

These coordinate systems play a role when guiding the robot, when teaching it positions, and when operating it. The world, work, and tool coordinate systems can be specified when guiding the robot. The robot moves along the specified coordinate system. However, there is no particular need to worry about which coordinate system to select when teaching or operating the robot. Under normal usage, the robot will move to the points you taught it to move to.

Caution: Coordinate system and additional axes
Axis 5 (T-axis) of the SCARA robot, and axis 4 (C-axis) and axis 5 (T-axis) of the Cartesian coordinate robot are the additional axes which will not be affected by the coordinate system. Values are the same in any coordinate system selected.

Fig. 5.1 shows the robot coordinate system.

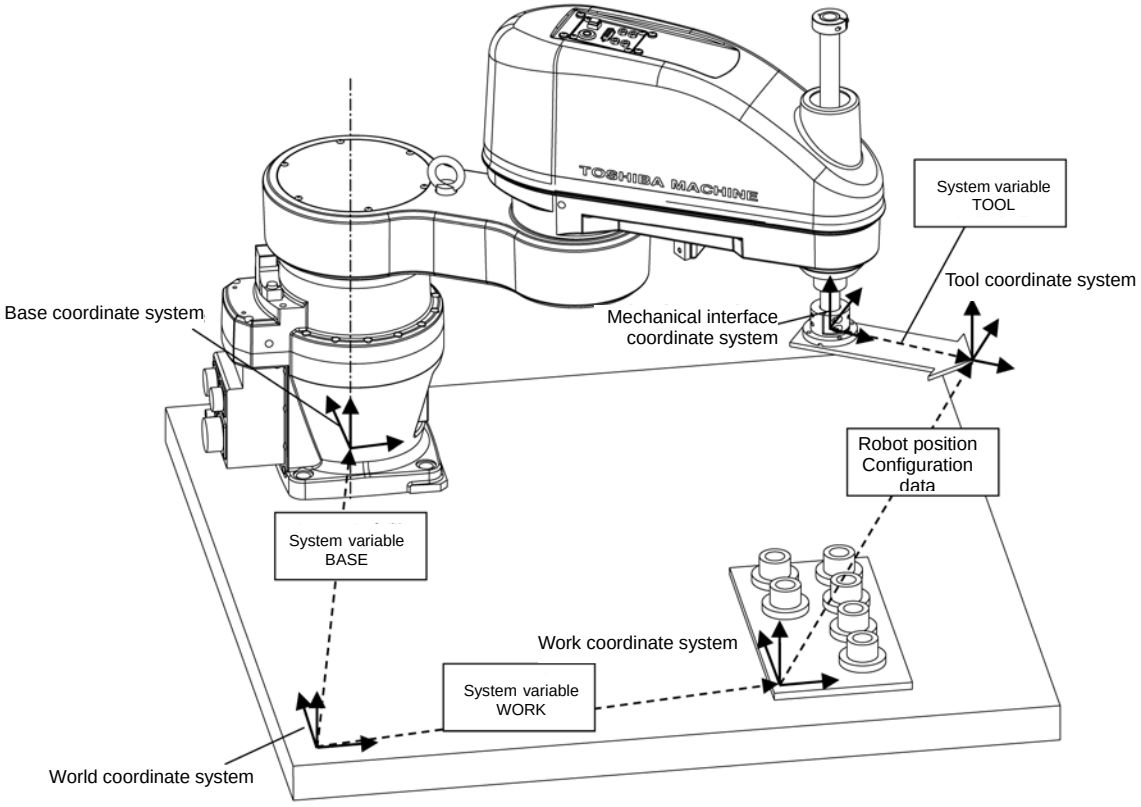


Fig. 5.1 Robot coordinate system

(2) Coordinate systems and system variables

The base, work and tool coordinate systems may each contain multiple coordinate systems, each of which can be selected according to robot tasks. Each coordinate system can be also specified in the SCOL language program. These coordinate systems may also be specified (defined) inside of a program by using the system variables BASE, WORK and TOOL. These system variables may be used just like any other coordinate-type system variables.

The meaning of each of these system variables is described below. For more information on teaching coordinate systems to the robot or selecting such a coordinate system, refer to the Operator's Manual.

(a) System variable BASE

The system variable BASE defines the origin of the base coordinate system in terms of the world coordinate system. It is used when it is necessary to base robot movement on the coordinate system of the work-site (world coordinate system) or when it is necessary to guide the robot in terms of world coordinates (as opposed to base coordinates).

With such a system, the robot can be reinstalled in a separate location and put back into action after simply redefining the location of the base coordinate system.

Should the value of the BASE system variable be set to all 0, the world coordinate system and the base coordinate system will become the same. (Note that the BASE system variable is of a coordinate data-type and contains four elements, all of which must be set to 0 in order to set the BASE system variable itself to 0.) You should set BASE to 0 if there is no particular need to specify a world coordinate system. Also, be careful not to accidentally change the base coordinate system after it has been set.

(b) System variable WORK

The system variable WORK defines the origin of the work coordinate system in terms of the world coordinate system. Should WORK have a value of 0, the world and work coordinate systems will be the same. (Note that the WORK system variable is of a coordinate data-type and contains four elements.)

When handling multiple workpieces, multiple work coordinate systems may be specified to help keep track of where each workpiece is.

Also, the WORK system variable is useful when it is necessary to guide the robot in relation to the workpiece (rather than, for example, the robot base). When there is no particular need to specify a work coordinate system, set the WORK system variable to all 0.

(c) System variable TOOL

The system variable TOOL defines the origin of the tool coordinate system in terms of the mechanical interface coordinate system. Should TOOL have a value of 0, the tool coordinate system and the mechanical interface coordinate system will be the same. (Note that the TOOL system variable is of a coordinate data-type and contains four elements.)

When handling multiple tools, multiple tool coordinate systems may be specified to help keep track of where each tool is. Also, the WORK system variable is useful when it is necessary to guide the robot in relation to the tool (rather than, for example, the robot base).

You should be careful when changing any of these system variables in your program since the coordinate system in which the robot moves will also change.

(3) Teaching data and coordinate systems

When you teach a position to the robot, the robot will also record the position of the tool tip relative to a work coordinate system. In addition, the robot will also remember the work coordinate system in effect at the time.

When a program using this data is executed, the robot will move to the position defined by the positional data for that point. However, if the work coordinate system itself was changed (and is therefore not the same as the work coordinate system in effect when taught), the same positional data will define a different point (since the frame of reference is different). Therefore, the robot will not move to the point as taught, but to another point.

(4) Changing coordinate system data in the program

A simple explanation of how to change coordinate system data with the program is presented below. You should not change such data unless you have a good reason for doing so.

(a) Changing the base coordinate system

There is no need to change the base coordinate system from the program.

(b) Changing the work coordinate system

When you teach the robot a position, the robot will also remember the work coordinate system in effect at the time. When a movement command tells the robot to move to that point, the current work coordinate system (in effect at the time the command was encountered in the program) will automatically change over to the previous work coordinate system (in effect when the point was taught to the robot).

Example:
You have three different work coordinate systems, i.e. WORK1, WORK2 and WORK3. In each coordinate system, you taught the robot one point, i.e. you have point A1 defined in terms of work coordinate system WORK1, point A2 defined in terms of work coordinate system WORK2, and point A3 defined in terms of work coordinate system WORK3. When you execute the following program, the work coordinate system will change as follows.

PROGRAM SAMPLE
MOVE A1: The work system in effect during this command will be WORK1.
MOVE A2: The work system in effect during this command will be WORK2.
MOVE A3: The work system in effect during this command will be WORK3.
END

When handling multiple workpieces, you can carry out the same operation over and over on different workpieces by changing the work coordinate system. There are three ways to do this.

1) Changing the work coordinate system itself
The same operation can be carried out on multiple workpieces by changing the values of the work coordinate system itself from the program.

Example:
You have three points (A1, A2 and A3) defined in terms of work coordinate system WORK1. You wish to repeat the operation, but this time in terms of work coordinate system WORK2.

```
PROGRAM SAMPLE
    DUMMYWORK = WORK1
    WORK1 = WORK2
    MOVE A1
    MOVE A2
    MOVE A3
     WORK1 = DUMMYWORK
END
```

Here, the movements to points A1, A2 and A3 were performed in terms of WORK1 just as before. The only difference is that the value of WORK1 itself was changed beforehand to that of WORK2. Therefore, in effect, the robot moved in the frame of reference of WORK2.

The variable DUMMYWORK is used to hold the value of the original WORK1. Otherwise, the value of the original WORK1 would be lost forever when you put the value of WORK2 into WORK1. When the robot is finished moving, the original value of WORK1 will be restored.

2) Changing the work coordinate system with a WITH statement
You can change between different work coordinate systems by using WITH statements.

Example:
You have three points (A1, A2 and A3) defined in terms of work coordinate system WORK1. You wish to repeat the operation, but this time in terms of work coordinate system WORK2.

```
PROGRAM SAMPLE
    MOVE A2
    WORK2=WORK+TRANS(, , 20)
    MOVE A1 WITH WORK = WORK2
    MOVE A2 WITH WORK = WORK2
    MOVE A3 WITH WORK = WORK2
END
```

Using WITH statements in this way, it is possible to specify work coordinate systems different from that used during teaching.

3) General method
As a general method, we recommend using a combination of the above two methods. Specifically, your program should use temporary variables to define the work coordinate system. Every time the workpiece is changed, a corresponding temporary variable should be use to change over the work coordinate system.

Example:
You have three points (A1, A2 and A3) defined in terms of work coordinate system WORK1. You wish to repeat the operation, but this time in terms of work coordinate system WORK2.

```
PROGRAM SAMPLE
    WORK2=WORK+TRANS(, , 20)
    DUMMYWORK = WORK2
    MOVE A1 WITH WORK = DUMMYWORK
    MOVE A2 WITH WORK = DUMMYWORK
    MOVE A3 WITH WORK = DUMMYWORK
END
```

(c) Changing the Tool Coordinate System

By changing the tool coordinate system, you can handle operations in which tools are changed several times as work progresses.

The robot always uses the current tool coordinate system in order to move the tip of the tool to the position defined by the work coordinate system. Therefore, if you are not careful when specifying the tool coordinate system, the robot may move somewhere unexpected.

Example:
You are using two tools with the tool offsets being TOOL1 and TOOL2.

```
PROGRAM SAMPLE
    TOOL1=TRANS(, , 10)
    TOOL2=TRANS(, , 30)
    TOOL = TOOL1
    MOVE A1
    TOOL = TOOL2
    MOVE A1
END
```

This program takes two different tools, and positions the tips of these tools at the same position (point A1). (Note that the program above does not have a tool change routine as would be required for actual operation.)

5.3.3 Short-Cut Movement

The SCOL language considers one movement to start when the robot begins moving and to stop when the robot finishes positioning itself. Normally, one movement command corresponds to one movement.

However, one may also direct the robot to move continuously under multiple movement commands without stopping to position itself before heading for the next destination or passing the nearest point toward the next destination. This is called short-cut movement.

Short-cut movement reduces operating time since the robot can both cut corners and not have to spend time positioning itself.

The short-cut movement cannot be executed between MOVES or MOVEC command and other movement commands. (However, the short-cut movement is available between the MOVES command and the MOVEC command.)

(1) Specifying Short-Cut Movement

Short-cut movement is invoked or discontinued with the system switch PASS.

ENABLE PASS – Invokes short-cut motion.
DISABLE PASS – Discontinues short-cut motion.

With short-cut motion, the robot continuously changes the speeds of the axes while being careful not to exceed any maximum speeds. When the amount of the movement per one movement command has exceeded the specified percentage, the robot will begin to execute the next movement command. This percentage is specified with the system variable PASS (which is not the same thing as the system switch PASS). This percentage which is formally called the short-cut movement parameter may be specified as an integer value between 0 and 100. Note that anything smaller than 50 will be treated as 50%.

Example:

```
PROGRAM SAMPLE
    MOVE A1          Move to point A1.
    PASS = 80        Set the short-cut movement parameter to 80%.
    ENABLE PASS      Invoke short-cut movement.
    MOVE A2          When 80% of the movement to point A2 is
                     completed, begin moving to point A3.
    MOVE A3          When 80% of the movement to point A3 is
```

```
                        completed, begin moving to point A4.
    DISABLE PASS        Discontinue short-cut movement.
    MOVE A4             Move to point A4.
END
```

The short-cut movement parameter may be changed while short-cut motion is in effect.

Example:

```
PROGRAM SAMPLE
    MOVE A1             Move to point A1.
    PASS = 80           Set the short-cut movement parameter to 80%.
    ENABLE PASS         Invoke short-cut movement.
    MOVE A2             When 80% of the movement to point A2 is
                        completed, begin moving to point A3.
    MOVE A3 WITH PASS = 60
                        When 60% of the movement to point A3 is
                        completed, begin moving to point A4.
    MOVE A4             When 80% of the movement to point A4 is completer,
                        begin moving to point A5.
    PASS = 90           Set the short-cut movement parameter to 90%.
    MOVE A5             When 90% of the movement to point A5 is
                        completed, begin moving to point A6.
    DISABLE PASS        Discontinue short-cut movement.
    MOVE A6             Move to point A6.
END
```

(2) Commands which interrupt short-cut movement

The following commands will interrupt short-cut motion should short-cut motion be in effect at the time.

WAIT command
INPUT command
PRINT command
STOP command
BREAK command
PAUSE command

Furthermore, should DISABLE NOWAIT be in effect, the following commands will interrupt short-cut motion.

DOUT command
RESET DOUT command
PULOUT command
DIN command
BCDIN command
BCDOUT command

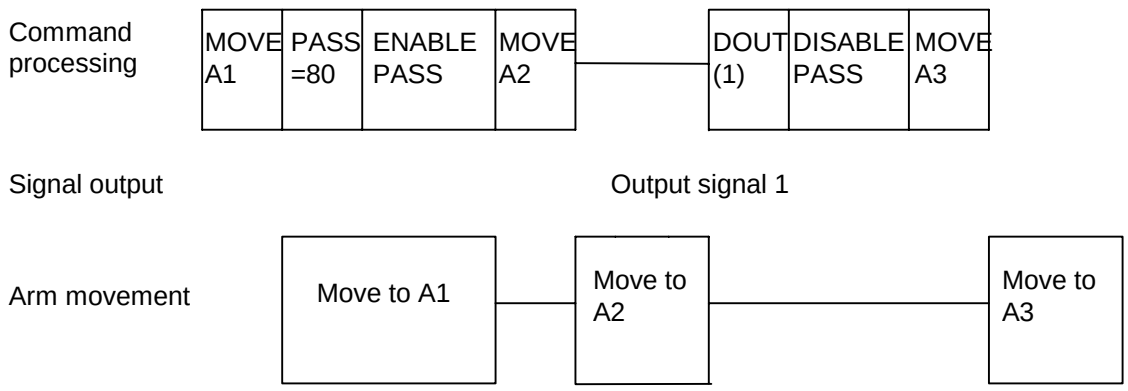
Moreover, when there are many commands between movement commands or when the amount of motion of an individual motion is small, the short-cut motion may be stopped. When the short-cut movement is specified between the MOVES or MOVEC command and another movement command, the short-cut movement will be stopped.

(3) Output signal timing under short-cut movement

Signal output timing under short-cut movement relative to robot arm motion is described below with the following example.

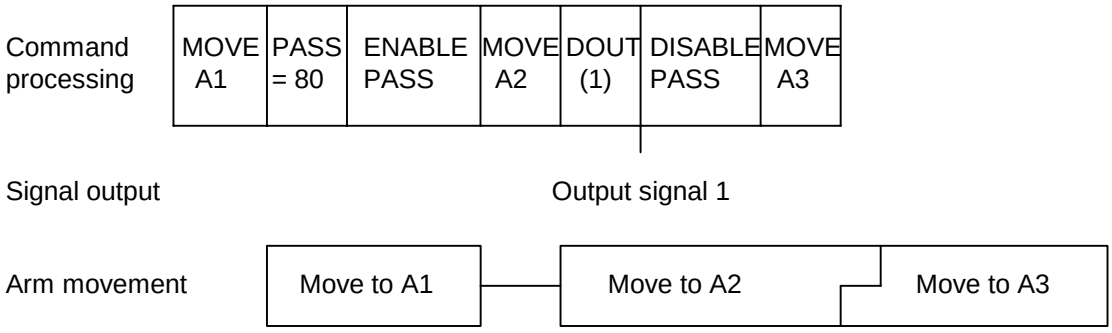
```
PROGRAM SAMPLE
    MOVE A1
    PASS = 80
    ENABLE PASS
    MOVE A2
    DOUT (1)
    DISABLE PASS
    MOVE A3
END
```

(a) Timing under DISABLE NOWAIT



Here, short-cut movement will be interrupted so that the output signal may be processed.

(b) Timing under ENABLE NOWAIT



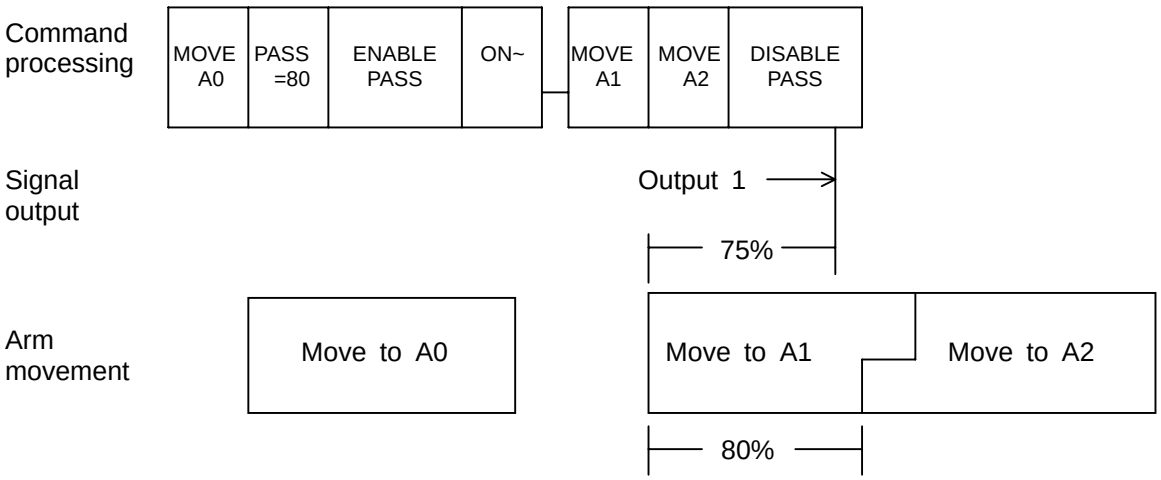
Here, the output signal is processed while the arm is in motion.

(4) Referring to the robot operating condition under short-cut movement
When you refer to the amount of robot movement with the MOTION, MOTIONT, REMAIN or REMAINT command while under short-cut movement, a value for one movement command will be returned as the result.

Example:

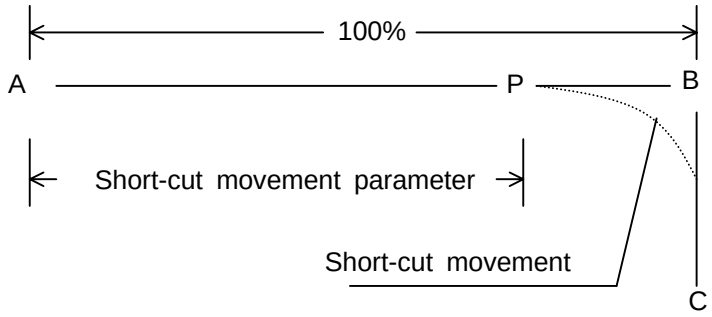
```
PROGRAM SAMPLE
    MOVE A0
    PASS = 80
    ENABLE PASS
    ON MOTION > = 75 DO DOUT (1)
    MOVE A1
    MOVE A2
    DISABLE PASS
END
```

Here, Signal 1 will be output when the robot has moved 75% of the specified value from A0 to A1.



(5) Control parameters for short-cut movement

The short-cut movement parameter has the following meaning.



In the above diagram, the robot begins at point A, goes through the vicinity of point B, and moves to point C. In short-cut movement, the robot will move from point A towards point B until it reaches point P. When the robot reaches point P, the robot will start moving towards point C. The position of point P is defined as a percentage of the total length between points A and B. In other words, the value of P is the short-cut movement parameter. Sometimes this parameter is also referred to as the pass rate.

In the above example, the short-cut movement parameter (pass rate) is given as follows:

Pass rate = ((Distance from A to P) / (Distance from A to B)) * 100 (%)

The short-cut movement parameter must be in the range of 50 to 100%. Anything less than 50% will be taken as 50%.
At times, short-cut movement may not work exactly as specified for the reasons below.

(a) Restrictions on maximum acceleration
According to the structure of the robot and parts in use, a maximum speed is determined. Furthermore, robot acceleration and deceleration varies depending on the reduction of the movement speed of the robot and the use of the ACCEL and DECEL command written in the robot language. During short-cut movement, the speed of each axis is calculated so that these accelerations are not exceeded.

(b) Restrictions from the following movement
The short-cut movement superimposes the current movement onto the subsequent movement in such a way that the current movement finishes until the robot moves 50% of the subsequent movement.

For the reasons described above, the timing with which short-cut movement begins cannot be speeded up by more than a certain amount. This is true even should the short-cut movement parameter (pass rate) be made smaller.

(6) The notice in the case where the direction of the shortcut becomes the direction of the identical vector
When the fellow of the operation to cut short becomes the one-way, it sometimes becomes faster than the speed that the falling of the speed ingredient increased and that the speed specified.

5.3.4 Robot Configuration

With SCRA robots, viewed from the rotation center of the robot to the configuration of which the arm extends straight, when the second arm bends to the left (i.e. the elbow sticks out to the right), it is called the right-handed system, and when the second arm bends to the right (i.e. the elbow sticks out to the left), it is called the left-handed system. Therefore the robot can take two postures: right-handed system configuration and left-handed system configuration to one position (X, Y) specified in the work coordinate system.

(1) Configuration during teaching and configuration during operation
When you teach a position to the robot, the robot will also remember (as positional data) its configuration at the time. With normal operation, the robot will move with the configuration it had when taught.

(a) Specifying the configuration for movement
The configuration the robot is to take while moving is specified by the CONFIG command. The CONFIG command may either be used independently or with a WITH statement.

When the CONFIG command is used with a WITH statement, the CONFIG command only has effect for one single movement command.
When a CONFIG command is used by itself, all subsequent movement commands are executed with that configuration.
When you want the robot to move with the same configuration with which it was taught, you must set the robot configuration to undefined, CONFIG=0 or CONFIG=FREE. The initial setting of the robot is FREE.

CONFIG = 1 or CONFIG = LEFTY will set the robot configuration to left, and CONFIG = 2 or CONFIG = RIGHTY will set the robot configuration to right.

When you set a robot configuration, the robot moves with the specified configuration during the execution of the subsequent movement command.

Should you operate the robot with a configuration different from that with which is was taught, there may be a discrepancy between where the robot was taught to move and where it does move. Therefore, always try to operate the robot with the configuration the robot was taught.

(b) Commands for which the configuration cannot be changed
Any configuration specifications become invalid when any of the following movement commands are executed.

1) Interpolated movement commands
With command for the linear interpolation and arc interpolation, since the configuration of the robot cannot be changed, an error will occur. There are two interpolated movement commands, i.e. MOVES and MOVEC.

2) Single axis control commands
With commands which direct one single axis of the robot to move, the robot configuration may not be specified. There are two single axis control commands, i.e., MOVEA and MOVEI.

(2) Moving to positions created in the program
When you directly specify coordinate values to which the robot is to move (for example, MOVE POINT (100, 100)), the robot will move with the configuration in effect at the time. This is true even should the configuration be undefined.

Should you create positional data in your program, the configuration of the robot will be the configuration of the positional data in that variable before you substituted in your new values. Should that positional data variable have no specified configuration (i.e., should that variable be substituted into for the first time), the robot configuration will be undefined (free).

5.3.5 Data Blocks

This paragraph describes how positional data is kept in the controller files. Note that there is no particular need for the programmer to worry about this if he or she is programming with the teach pendant. Rather, this paragraph is for use when creating a SCOL language program or positional data with any computer other than the robot controller.

All positional data files are in ASCII code.

(1) Data blocks

Robot positional data is stored in the controller in units called data blocks. The file in the controller contains a plurality of programs and one data block. One file always has one data block. The programs in the same file share positional data in the data block. You cannot refer to the data block in different file. The data block stores coordinate data and load data in addition to positional data.

A data block is declared in a file in the following manner:

```
DATA
(data declarations)
. . .
END
```

A data block is declared from DATA to END. Individual datum is declared from DATA to END one after the other. The data block is declared at the end of the file. The data block cannot be declared in a program.

(2) Data declarations

Positional data, coordinate data and load data are normally taught to the robot with the controller data editor. In such a case, the data is automatically entered into the data block of the appropriate file.

Data defined (created) in the program (and not by the data editor) is not entered into the data block. Rather, this data is temporarily stored in the controller (and not in the file) while the program is running.

Data in a data block is declared with the following format.

<data type> <identifier> = [<element>,]...

The <data type> designation indicates the type of data you are declaring. Here, you should write POINT for positional-type data, TRANS for coordinate-type data, or PAYLOAD for load-type data.

The <identifier> designation indicates the name of the data. The <element> designation indicates the numerical value of each element in real numbers. Any omitted <element> designations will be taken as 0.

(a) Declaring positional data

Positional data is declared with the following format.

POINT <identifier> = X, Y, Z, C, T/<configuration>

X, Y, Z, C and T are coordinate values expressed in real numbers. Units are in millimeters or degrees.

<configuration> is an integer from 0 to 2 which designates the robot configuration.

NONE: Undefined (free)
LEFTY: Left handed
RIGHTY: Right handed

The system constants FREE (undefined), LEFTY (left handed) and RIGHTY (right handed) may also be used to designate the robot configuration.

Examples:

```
POINT A = 100, 200, 30, 45, 0
POINT A1 = 444.44, 333.33, , ,/RIGHTY
POINT ZERO =
```

(b) Declaring coordinate data

Coordinate data is declared with the following format.

TRANS <identifier> = X, Y, Z, C

X, Y, Z and C are coordinate values expressed in real numbers. Units are in millimeters or degrees.

Examples:

```
TRANS WORK1 = 10, 20, 30, 45
TRANS TOOL2 = , , –20,
TRANS ZEROW =
```

(c) Declaring load data

Load data is declared with the following format.

PAYLOAD <identifier> = <mass>, <center of gravity offset>

<mass> is the mass acting on the tip of the robot hand expressed as a real number. Units are in kilograms.

<center of gravity offset> is the offset for center of gravity acting on the tip of the robot hand expressed as a real number. Units are in millimeters.

Examples:

```
PAYLOAD HAND1 = 4.8, 0.48
PAYLOAD HAND2 = 2, 0.004
PAYLOAD HAND0 =
```

(3) Specifying work coordinate systems

Positional data is specified in the SCOL language as the position of the tool tip defined in terms of the work coordinate system. Therefore, positional data also contains information as to which work system was used to define that data. To explicitly specify a work coordinate system, use a WORK statement to give the work coordinate system a name. You may then use the work coordinate system in your program. The work coordinates declared in this manner remain in effect until superceded by another WORK statement.

Example:

```
DATA
    POINT A00 = 650, 0, 0, 0, 0
    POINT A01 = 400, 400, 0, 0 / RIGHTY
    TRANS WORK1 = 100, – 100, 0, 0
    WORK WORK1
    POINT A10 = 0, 0, 0, 0, 0, 0
    POINT A11 = 200, 0, 0, 0, 0, 0
    TRANS WORK2 = –100, –100, 0, 0, 0
    WORK WORK2
    POINT A20 = 246.8, 69.1, 23.5, 18.3, / RIGHTY
    POINT A21 = 0, 0, –30, 0, 0 / LEFTY
END
```

In the above example, positional data A10 and All are defined in terms of WORK1, and positional data A20 and A21 are defined in terms of WORK2. Also, since no particular work coordinate system was in effect when positional data A00 and A01 were specified; the work coordinate system for these points is taken as {0, 0, 0, 0}.

5.3.6 Global Data Block

The variable defined by the SCOL language contains the global data and temporary data.
The global data which can be referred to from all parts of the program is described in this paragraph.

The variable defined in the data block is dealt with as the global data. The data which can be defined in the data block is limited to the position type, coordinate type and load type.

Data of global integer type, real number type and array type can be used by the declaration of global data.

(1) Global data block

The global data is declared in the global data block and is dealt with as a part of the program in the different manner as the data block. The global data block is edited by the program editor. The file of the controller contains a plural number of programs and one global data block. One file has one global data block.

The global data can be shared in the programs of the same file. The global data of different file cannot be referred to. The integer data, real number data and array data can be stored in the global data block.

The global data block is declared in the file, using the following format.

```
GLOBAL
(Declaration of data)
. . .
END
```

The global data block is declared in the GLOBAL ~ END statements.
Respective data are declared in the GLOBAL ~ END statements one by one.
The global data block is declared at the head of file.
The global data block cannot be declared in the program.

(2) Declaration of data

The global data block cannot be edited by the data editor.
It is edited by the program editor in the same manner as the program.
Data in the global data block are declared in the format of substitution statement for the variable in the same manner as the program.

(a) Declaration of integer data
The integer data is declared as shown below.

<Identifier> = <integer constant>
Example: N1 = 1

Note: If the real number is substituted for the integer type global variable in the program, the decimal places are omitted. Care should be taken.

(b) Declaration of real number
The real number is declared as shown below.

<Identifier> = <real number constant>
Example: R1 = 1.0

(c) Declaration of array data
The array data is declared as shown below.

DIM <identifier> (I, j . . .) AS <variable type>

Example: Integer type three dimensional array which has $2 \times 3 \times 4$ elements
DIM IDAT (2, 3, 4) AS INT

Real number type two dimensional array which has 4×3 elements
DIM RDAT (10, 50) AS REAL

Position type one dimensional array which has five elements
DIM PDAT (5) AS POINT

Note: For the DIM command, only the type and number of elements of array type global data are specified and the initial value is unsettled. Like normal global data, initial values of integer type and real number type data should be specified in global data blocks, and those of position type, coordinate type and load type data in data blocks.

5.3.7 Robot Movement Speed

(1) Speed in each mode

The range of robot speed utilized in each mode is shown below.

Mode		Speed range
Automatic operation mode	PTP	0 ~ 100 %
	Linear and circular interpolation	0 ~ 100 %
Test operation mode	PTP	0 ~ 25%
	Linear and circular interpolation	0 ~ 25 %

Note: The maximum speed listed in the specifications is taken as 100%. With linear and circular interpolation, 1 m/s is taken as 100%.

(2) Speed for automatic operation

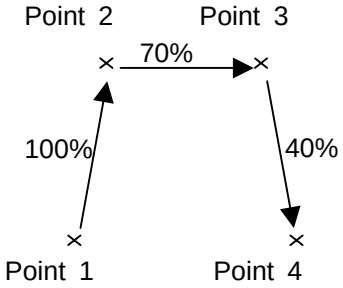
Speeds for automatic and test operation have the following format.

Speed = {Speed setting in program (1 to 100%) × Override speed (1 to 100%); Limit speed}

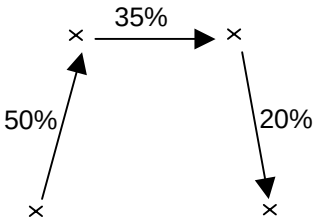
Here,

Speed setting in program: Setting specified by the SPEED command in the robot program.
Override speed: Changes all speeds by the same fraction.
Limit speed: Reduces any speeds over the limit down to the limit value.

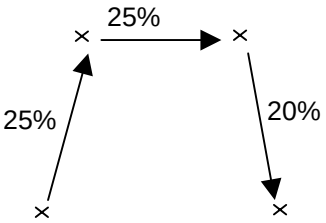
Program setting → Override 50% → Limit 25%



Speed for each operation can be set.



Speed for all movements is 50% of the specified speed value.



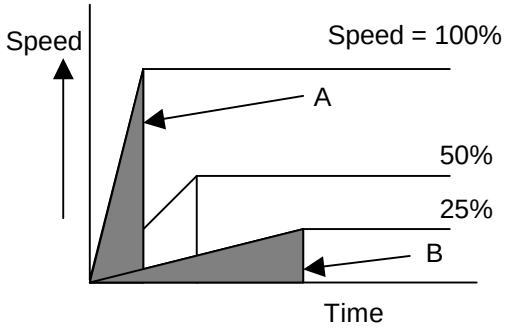
Since the movement speed from Point 1 to Point 3 exceeds 25%, it is reduced to 25%.

5.3.8 Robot Acceleration

Robot acceleration (and deceleration) will vary depending on the following factors.

[1] Operating mode: Linear interpolation (MOVES), point-to-point (MOVE)
[2] Speed: SPEED command, override
[3] Acceleration commands: ACCEL, DECEL

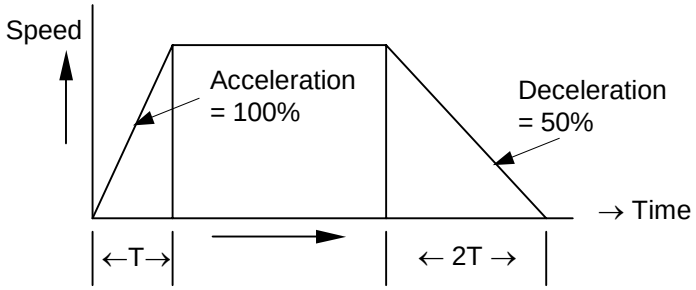
(1) The maximum acceleration in each operating mode is calculated from the robot strength and motor torque and set. Also, the motors will speed up and slow down in such a way that the robot moves smoothly.



Change in speed according to override

As shown in the figure above, the acceleration time is inversely proportional to the speed override setting. The distance of travel during acceleration (areas of triangle A and B in the figure) is constant independent of the speed override setting. Therefore, the path of the robot will stay very nearly the same no matter what the override setting.

(2) Acceleration and deceleration are specified separately in the SCOL language. This comes in useful when handling delicate parts in that, for example, you can slow the robot down gradually before stopping in order to keep the hand and workpiece from vibrating.



Appendix A List of Commands

Movement control commands

MOVE<position>[WITH clause]	Simultaneous movement
MOVES<position>[WITH clause]	Linear interpolated movement
MOVEC<position>,<position>[WITH clause]	Circular interpolated movement
MOVEA<axis>,<absolute position>[WITH clause]	Absolute single axis movement
MOVEI<axis>,<relative position>[WITH clause]	Relative single axis movement
MOVEJ<position>,<arch definition>	Relative single axis movement
READY	Moves to machine coordinate origin
DELAY<time>	Pauses for specified time
OPEN1,OPEN2	Opens hand immediately
OPENI1,OPENI2	Opens hand
CLOSE1,CLOSE2	Closes hand immediately
CLOSEI1,CLOSEI2	Closes hand
RESUME	Resumes interrupted movement

Program control commands

PROGRAM	<program name>	Marks beginning of program
ON<monitoring condition>	[{BREAK | PAUSE}]	Monitors conditions

DO<statement>

IF<logical statement> THEN<statement> [ELSE<statement>]	Judges conditions
WAIT<logical statement>	Waits for an operation
IGNORE<monitoring condition>	Cancels monitoring
GOTO<label>	Branches unconditionally
GOTO (<expression>) <label> [, <label>]...	Branches in accordance with the value of an expression
RCYCLE	Label for cycle setting
RETURN	Returns to main program

FOR<variable> = <expression> TO <expression> [STEP <expression>]	Repeats an operation
NEXT[<variable>]	
STOP	Stops the program
REMARK[<comment>]	Remarks and comments
END	Marks end of program
TASK (“program name”)	Starts task program
KILL (<expression>)	Ends task program
SWITCH	Changes over task program
LOADLIB<file name>	Dynamic link library build-in

I/O control commands

DIN(<signal name>[,<signal name>]...)	Reads in an input signal
DOUT(<signal name>[,<signal name>]...)	Outputs a signal
PULOUT(<signal name>[,<signal name>]...)	Outputs a pulse signal
RESET<condition>[,<condition>]	Resets the controller
BCDIN(<signal name>,<signal length>)	Inputs a BCD signal
BCDOUT(<signal name>,<signal length>,<expression>)	Outputs a BCD signal
HEXIN(<signal name>,<signal length>)	Inputs signals in hexadecimal notation.
HEXOUT(<signal name>,<signal length>,<expression>)	Outputs signals in hexadecimal notation.
PRINT[{COM0 | COM1 | TP},]	
{<character string> <expression>}	
[,{<character string><expression>}]...[, CR]	Outputs communication data
INPUT[{COM0 | COM1 | TP},]	
<variable>,[<variable>]...	Inputs communication data

Movement condition commands

CONFIG=<expression>	Specifies configuration
ACCUR=<expression>	Specifies positioning accuracy

ACCEL=<expression>	Specifies acceleration (during acceleration)
DECEL=<expression>	Specifies deceleration (during deceleration)
SPEED=<expression>	Specifies speed
PASS=<expression>	Short-cut movement parameter
TORQUE={<expression>, <expression>, <expression>, <expression>, <expression>, <expression>}	Torque on each shaft
GAIN={<expression>, <expression>, <expression>, <expression>, <expression>, <expression>}	Gain on each shaft
SETGAIN=(<integer>, <integer>, <integer>, <integer>, <integer>)	Gain on each axis in synchronous motion
ENABLE<switch>[,<switch>]...	System switch on
DISABLE<switch>[,<switch>]...	System switch off
PAYLOAD={<mass>,<center of gravity offset>}	Sets load data
FREELOAD	Cancels load data

Palletize command

INITPLT (<pallet number>,<i>,<j>,<k>)	Initializes a pallet
MOVEPLT (<pallet number>, <element number>, <X>, <Y>,<Z>,<C>)	Moves pallet to specified position

Calculator commands

SIN(<expression>)	Sine
COS(<expression>)	Cosine
TAN(<expression>)	Tangent
ASIN(<expression>)	Arcsine
ACOS(<expression>)	Arccosine
ATAN(<expression>)	Arctangent
ATAN2(<expression>,<expression>)	Arctangent
SQRT(<expression>)	Square root
ABS(<expression>)	Absolute value

SGN(<expression>)	Extracts sign
INT(<expression>)	Changes number to an integer
REAL(<expression>)	Changes number to a real number
LN(<expression>)	Natural logarithm
<expression> MOD <expression>	Remainder
LOG10 (<expression>)	Common logarithm
EXP (<expression>)	Exponent to power e
<logical expression> AND <logical expression>	Logical product
<logical expression> OR <logical expression>	Logical sum
NOT <logical expression>	Negation
HERE	Present position
DEST	Destination position
POINT (<expression>, <expression>, <expression>, <expression>, <expression>, <expression>, <configuration>)	Creates positional type data
TRANS (<expression>, <expression>, <expression>, <expression>)	Creates coordinate type data

Movement condition commands

MOTION	Amount of movement which has been executed
MOTIONT	Time expended for a motion
REMAIN	Amount of movement remaining to be executed
REMAINT	Time remaining for a motion
TIMER	Timer
MODE	System operating mode
TOOL	Tool coordinate system
BASE	Base coordinate system
WORK	Work coordinate system

Appendix B List of Reserved Words

ABS	ACCEL	ACCUR	ACOS
AND	AS	ASIN	ATAN
ATAN2	BASE	BCDIN	BCDOUT
BREAK	CLOSE1	CLOSE2	CLOSEI1
CLOSEI2	COARSE	COM0	COM1
CONFIG	CONT	COS	CR
CYCLE	DATA	DECEL	DELAY
DEST	DIM	DIN	DISABLE
DO	DOUT	ELSE	ENABLE
END	EXP	FINE	FOR
FREE	FREELOAD	GAIN	BLOBAL
GOTO	HERE	HEXIN	HEXOUT
IF	IGNORE	INITPLT	INPUT
INT	KILL	LATCH	LATCHTRG1~8
LATCHSIG1~8	LATCHPSN1~8	LEFTY	LOADLIB
LN	LOG10	MAXTASK	MOD
MODE	MOTION	MOTIONT	MOVE
MOVEA	MOVEC	MOVEI	MOVEJ
MOVEPLT	MOVES	MOVESYNC	NEXT
NOT	NOWAIT	OFF	OFFGAIN
ON	ONGAIN	OPEN1	OPEN2
OPENI1	OPENI2	OR	PAI
PASS	PAUSE	PAYLOAD	PLCDATAR1~8
PLCDATAW1~8	POINT	PRINT	PROGRAM
PULOUT	RCYCLE	READY	REAL
REMAIN	REMAINT	REMARK	RESET
RESTORE	RESUME	RETURN	RIGHTY
SAVEEND	SEGMENT	SETGAIN	SGN
SIN	SMOOTH	SPEED	SQRT
STEP	STOP	SWITCH	TAN
TASK	THEN	TID	TIMER
TO	TP	TOOL	TORQUE
TRANS	WAIT	WITH	WORK

Appendix C Contents of Library File (SCOL.LIB)

The contents of the library file included as standard on the system are shown below. Details may vary slightly by the customer.

After the library file has been changed, be sure to execute the SELECT command again. Otherwise, the change thus made will not be reflected on the currently selected program.

```
    ‘(C) COPYRIGHT 2008 by TOSHIBA MACHINE CO., LTD.
    ‘ALL RIGHTS RESERVED.
    ‘TS3000 SCOL SUBPROGRAM LIBRARY

PROGRAM READY
    MOVEA 3, 0
    MOVEA 4, 0
    MOVEA 2, 0
    MOVEA 1, 0
    MOVEA 5, 0
END

PROGRAM OPEN1                       ’ OPEN HAND-1
    WAIT MOTION > = 100
    DOUT (203, –204)
END

PROGRAM CLOSE1                      ’ CLOSE HAND-1
    WAIT MOTION > = 100
    DOUT (–203, 204)
END

PROGRAM OPENI1                      ’ OPEN HAND-1 IMMEDIATE
    DOUT (203, –204)
END

PROGRAM CLOSEI1                     ’ CLOSE HAND-1 IMMEDIATE
    DOUT (–203, 204)
END
```

```
PROGRAM OPEN2                                   ' OPEN HAND-2
    WAIT MOTION > = 100
    DOUT (201, –202)
END

PROGRAM CLOSE2                                  ' CLOSE HAND-2
    WAIT MOTION > = 100
    DOUT (–201, 202)
END

PROGRAM OPENI2                                  ' OPEN HAND-2 IMMEDIATE
    DOUT (201, –202)
END

PROGRAM CLOSEI2                                 ' CLOSE HAND-2 IMMEDIATE
    DOUT (–201, 202)
END

PROGRAM FREELOAD                                ' FREE PAYLOAD
    PAYLOAD = (0, 0)
END

PROGRAM SETGAIN (J_0, J_1, J_2, J_3, J_4)       ' SET GAIN VARIABLE
    WAIT MOTION>=100
    GAIN = (J_0, J_1, J_2, J_3, J_4)
    MOVE HERE WITH PASS=100
    MOVE HERE
END

PROGRAM ONGAIN (J_1, J_2, J_3, J_4, J_5)
    J_6 = 0
    J_7 = 0
    J_8 = 0
    J_9 = 0
    J_0 = 0
    IF J_1 == 0 THEN J_6 = GAIN.1 ELSE J_6 = 1
```

```
    IF J_2 == 0 THEN J_7 = GAIN.2 ELSE J_7 = 1
    IF J_3 == 0 THEN J_8 = GAIN.3 ELSE J_8 = 1
    IF J_4 == 0 THEN J_9 = GAIN.4 ELSE J_9 = 1
    IF J_5 == 0 THEN J_0 = GAIN.5 ELSE J_0 = 1
    WAIT MOTION >= 100
    GAIN = (J_6, J_7, J_8, J_9, J_0)
    MOVE HERE WITH PASS = 100
    MOVE HERE
    RETURN
END

PROGRAM OFFGAIN (J_1,J_2,J_3,J_4,J_5)
   J_6 = 0
   J_7 = 0
   J_8 = 0
   J_9 = 0
   J_0 = 0
   IF J_1 == 0 THEN J_6 = GAIN.1 ELSE J_6 = 0
   IF J_2 == 0 THEN J_7 = GAIN.2 ELSE J_7 = 0
   IF J_3 == 0 THEN J_8 = GAIN.3 ELSE J_8 = 0
   IF J_4 == 0 THEN J_9 = GAIN.4 ELSE J_9 = 0
   IF J_5 == 0 THEN J_0 = GAIN.5 ELSE J_0 = 0
   WAIT MOTION > = 100
   GAIN = {J_6, J_7, J_8, J_9, J_0}
   MOVE HERE WITH PASS = 100
   MOVE HERE
   RETURN
END
```

Appendix D Domains and Ranges of Calculator Functions

Function	Domain of arguments X, Y	Range of result Z
SIN (X)	(*)	$-1 \leq Z \leq 1$
COS (X)	(*)	$-1 \leq Z \leq 1$
TAN (X)	(*)	(*)
ASIN (X)	$-1 \leq X \leq 1$	$-90° \leq Z \leq 90°$
ACOS (X)	$-1 \leq X \leq 1$	$0 \leq Z \leq 180°$
ATAN (X)	(*)	$-90° < Z < 90°$
ATAN2 (X, Y)	$Y \neq 0$	$-180° < Z < 180°$
SQRT (X)	$X \geq 0$	$Z \geq 0$
ABS (X)	(*)	$Z \geq 0$
SGN (X)	(*)	$Z = -1, 0, 1$
INT (X)	(*)	(*)
REAL (X)	(*)	(*)
LN (X)	$X > 0$	(*)
X MOD Y	$Y \neq 0$	(*)
LOG10 (X)	$X > 0$	(*)
EXP (X)	(*)	$Z > 0$
Comments	(*) refers to any number within the range that can be handled by the controller.	

Appendix E How to Read Symbols

The meanings of keys and symbols used in the robot are as follows (alphanumeric characters are omitted).

[F1] ~ [F5]	:	Function keys F1 to F5
[Esc]	:	Escape key
[INS]	:	Insert key
[DEL]	:	Delete key
[BS]	:	Backspace key
[{]	:	Left middle size brace
[}]	:	Right middle size brace
[[]	:	Left large size brace
[]]	:	Right large size brace
[Error]	:	Error indication key
[Utility]	:	Utility key
[!]	:	Exclamation mark
[;]	:	Semicolon
[:]	:	Colon
[’]	:	Apostrophe
[%]	:	Percent
[^]	:	Accent circumflex
[&]	:	Ampersand
[”]	:	Quotation marks (double quotation marks)
[(]	:	Left parentheses
[)]	:	Right parentheses
[Alt]	:	Alt key (alternative key)
[+]	:	Plus
[–]	:	Minus
[/]	:	Slash
[*]	:	Asterisk
[ ]	:	Space
[<]	:	Less than
[>]	:	Greater than

[,]	:	Comma
[.]	:	Period
[?]	:	Question mark
[=]	:	Equal
[EXE]	:	Execution key
[↑]	:	(Up) cursor key
[↓]	:	(Down) cursor key
[←]	:	(Left) cursor key
[→]	:	(Right) cursor key

Appendix F List of Compile Errors

Compile error messages displayed on the teach pendant are tabled below.

Error No.	Descriptions
200	The system is not ready for execution.
201	The working memory cannot be maintained.
202	The command is illegal.
205	Constant of the numerical value is illegal.
206	Constant of the character string is illegal.
207	A character that cannot be used has been found.
208	An error has been found in the expression of substitution.
209	An error has been found in the expression of program format.
210	An error has been found in the program format.
211	The GLOBAL variable is used for the GOTO label.
212	A vector variable cannot be initialized in the GLOBAL area.
213	The PROGRAM statement is not at the head of the line.
214	The position of the RETURN command is illegal.
215	The PROGRAM statement disagrees with the END statement.
216	The PROGRAM command has been declared in the DATA area.
217	The PROGRAM command has been declared in the GLOBAL area.
218	The GLOBAL statement is not at the head of the line.
219	The GLOBAL statement disagrees with the END statement.
220	The GLOBAL command has been declared in the PROGRAM area.
221	The GLOBAL command has been declared in the DATA area.
222	The number of condition monitor (ON~DO~) areas exceeds 50.
223	The [GOTO] label is defined repeatedly.
224	The DATA command has been declared in the PROGRAM area.
225	The DATA command has been declared in the GLOBAL area.
226	This command cannot be declared in other than the GLOBAL area.
227	The number of dimensions of array is illegal.
228	The IF statement, THEN statement and ELSE statement disagree with each other.
229	The FOR statement disagrees with the NEXT statement.
230	The number of nesting of FOR~NEXT statements exceeds 127.
231	Multi-definition of the reserved word has been commanded.

Error No.	Descriptions
232	The monitoring condition is illegal.
233	The expression is illegal.
234	The operator is illegal.
235	The RCYCLE label can be used only in the MAIN function.
236	No vector variable can be used.
237	This command cannot be used for the vector variable.
238	Too many elements have been specified.
239	I/O instruction cannot be used for argument of function.
240	The parentheses have not been specified legally.
242	The RCYCLE label cannot be used only for the MAIN function.
243	No command of jump to the FOR loop area is allowed.
244	The label is not at the head of the line.
245	The specified label is absent.
246	No PROGRAM data is available.
247	No inequality can be used for the THEN statement or ELSE statement.
248	An error has been found in the reserved word.
249	The real argument and temporary argument of the function are not identical.
250	More than ten arguments of function cannot be specified.
252	The GLOBAL variable cannot be used as the function name.
253	The reserved word cannot be used as the function name.
254	The functional declaration is illegal.
255	The name of function is already declared.
256	The specified function is absent.
257	The name of GLOBAL variable is not defined yet.
258	The GOTO label is used as the variable name.
259	This command has not been declared in the PROGRAM area.
260	The specified reserved word cannot be used for the GLOBAL or DATA area.
261	Neither GLOBAL nor DATA variable can be substituted or redeclared.
262	The specified variable or constant cannot be used.
263	Neither logical operator nor inequality can be used.

Error No.	Descriptions
264	The type variable used is not common.
265	Under declaration. No END statement is present.
266	This command cannot be used in other than the head of the GLOBAL block.
267	The number of backup variables is too many.
268	The total number of array variables that can be declared is 11,000 max.
269	The number of POINT teach points that can be specified is 1,500 max.
270	The number of teach points other than POINT, that can be specified is 500 max.
271	The number of signals is too many.
272	Double definition of PASS and SMOOTH is not allowed.

* For details of Error No. 201. see the descriptions on the subsequent pages.

Detailed information on Compile Error No. 201 is given below.

No.	Error contents	Max. value	Detailed descriptions
1	Token code conversion buffer	75500	
2	GLOBAL variable	500	The maximum number of GLOBAL variables defined is 500.
3	AUTO variable	600	The maximum number of AUTO variables defined is 600.
4	Function	50	The maximum number of functions is 50.
5	GOTO label	1000	The maximum number of GOTO labels is 1,000.
6	Teaching of WORK coordinates	50	The maximum number of WORK coordinates is 50.
7	Variable name saving buffer	10000	
8	No. of program lines	6000	The total number of GLOBAL, program and data blocks is 6,000 max. (including blank lines).
9	Undefined variable information table	100	Unused.
10	Array variable	100	The maximum number of array variables defined is 100.
11	Declaring integer type GLOBAL variable	100	The maximum number of integer type GLOBAL variables defined is 100.
12	Declaring real number type GLOBAL variable	100	The maximum number of real number type GLOBAL variables defined is 100.
13	Declaring load type GLOBAL variable	100	The maximum number of load type GLOBAL variables defined is 100.
14	Declaring coordinate type GLOBAL variable	100	The maximum number of coordinate type GLOBAL variables defined is 100.
15	Declaring position type GLOBAL variable	1500	The maximum number of position type GLOBAL variables defined is 1,500.

No.	Error contents	Max. value	Detailed descriptions
16	Declaring integer type AUTO variable	500	The maximum number of integer type AUTO variables defined is 500.
17	Declaring real number type AUTO variable	500	The maximum number of real number type AUTO variables defined is 500.
18	Declaring load type AUTO variable	100	The maximum number of load type AUTO variables defined is 100.
19	Declaring coordinate type AUTO variable	100	The maximum number of coordinate type AUTO variables defined is 100.
20	Declaring position type AUTO variable	1000	The maximum number of position type AUTO variables defined is 1,000.
21	Declaring undefined AUTO variable	100	
22	Information on destination to which function is sent	300	The maximum number of function calls is 300. (One (1) function is called six (6) times on the average.)
23	Information on source from which function is called.	500	To be limited by No.22 above.
24	Information on argument of function	100	The maximum number of function arguments is 100. (One (1) function can have two (2) arguments on the average.)
25	Information on source from which GOTO command is called	1000	The maximum number of GOTO commands is 1,000.
26	Integer type AUTO constant	2000	The maximum number of integer type constants used in the program block is 2,000.
27	Real number type AUTO constant	2000	The maximum number of real number type constants used in the program block is 2,000.
28	Load type AUTO constant	200	The maximum number of load type constants used in the program block is 100 (= 200 ÷ 2).

No.	Error contents	Max. value	Detailed descriptions
29	Coordinate type AUTO constant	100	The maximum number of coordinate type constants used in the program block is 25 (= 100 ÷ 4).
30	Position type AUTO constant	2000	The maximum number of position type constants used in the program block is 333 (= 2000 ÷ 6).
31	Information on AUTO character string	1000	The maximum number of position type constants used in the program is 1,000.
32	Integer type GLOBAL constant	100	The maximum number of integer type constants used in the GLOBAL and data blocks is 100.
33	Real number type GLOBAL constant	100	The maximum number of real number type constants used in the GLOBAL and data blocks is 100.
34	Load type GLOBAL constant	100	The maximum number of load type constants used in the GLOBAL and data blocks is 50 (= 100 ÷ 2).
35	Coordinate type GLOBAL constant	100	The maximum number of coordinate type constants used in the GLOBAL and data blocks is 25 (= 100 ÷ 4).
36	Position type GLOBAL constant	10000	The maximum number of position type constants used in the GLOBAL and data blocks is 1666 (= 10000 ÷ 6).
37	No. of integer type GLOBAL variables used	1000	The total number of integer type GLOBAL variables used is 1,000 max.
38	No. of real number type GLOBAL variables used	200	The total number of real number type GLOBAL variables used is 200 max.
39	No. of load type GLOBAL variables used	100	The total number of load type GLOBAL variables used is 100 max.

No.	Error contents	Max. value	Detailed descriptions
40	No. of coordinate type GLOBAL variables used	100	The total number of coordinate type GLOBAL variables used is 100 max.
41	No. of position type GLOBAL variables used	3000	The total number of position type GLOBAL variables used is 3,000 max.
42	No. of integer type AUTO variables used	1000	The total number of integer type AUTO variables used is 1,000 max.
43	No. of real number type AUTO variables used	1000	The total number of real number type AUTO variables used is 1,000 max.
44	No. of load type AUTO variables used	100	The total number of load type AUTO variables used is 100 max.
45	No. of coordinate type AUTO variables used	100	The total number of coordinate type AUTO variables used is 100 max.
46	No. of position type AUTO variables used	2000	The total number of position type AUTO variables used is 2,000 max.
47	No. of AUTO undefined variables used	100	Unused.
48	No. of GOTO source indexes	2000	To be limited by No.25.
49	No. of function source indexes	1000	To be limited by No.22.
50	No. of function argument indexes	200	To be limited by No.24.
51	No. of array variables used	200	The maximum number of array variables defined is 200.
52	No. of RESTORE commands	100	The maximum number of RESTORE commands used is 100.
53	No. of array variable numerals saved	17000	The total number of elements of initialized array variables is 17,000 max. (In the position type array of 2×3 dimensions, it is 36 ($= 2 \times 3 \times 6$).

No.	Error contents	Max. value	Detailed descriptions
100	No. of indexes	5000	The total number of variable, constant, function, label, etc. used is 5,000.
101	No. of numerical data	25000	The total number of variable, constant, function, label, etc. defined is 25,000.
102	No. of codes created	399800	
200	Interpreter execution information area		

* If the library file (SOCL.LIB) exists, the number of data used there is also added. For the restrictions, see the restrictions on SCOL program as stated below.

The restrictions imposed on the SCOL program are tabled below.

Item	Max. No. per file	Remarks
No. of program lines	5,500	
Function	49	
Argument of function	10	
No. of GOTO labels	999	However, the number of declared labels and GOTO labels that can be specified in one (1) function is 599 max. (The same GOTO label is counted as one (1) even if a plural number of identical GOTO labels exist.)
Integer type GLOBAL variable	99	
Real number type GLOBAL variable	99	
Load type GLOBAL variable	48	
Coordinate type GLOBAL variable	23	
Position type GLOBAL variable	499	
Integer type AUTO variable	499	
Real number type AUTO variable	499	
Load type AUTO variable	99	
Coordinate type AUTO variable	99	However, up to 24 constants can be set in the same file (i.e., program).
Position type AUTO variable	999	However, the number of variables that can be set in one (1) function is 599 max. Up to 333 constants can be set in the same file (i.e., program).
Information on AUTO variable and label	599 (per function)	This is the total number of AUTO variables and GOTO labels included in one (1) function, and is not the limit value specified in one (1) file.
Array variable	99	
Total number of array variable elements	11,000	

Item	Max. No. per file	Remarks
No. of teach points of array variable position type data (POINT)	1,500	
No. of teach points of data other than array variable position type data (POINT)	500	
No. of nesting of FOR~NEXT	127	
No. of condition monitors (ON~DO~) specified simultaneously	10	
No. of condition monitors (ON~DO~) declared	50	

* The SCOL.LIB file is also counted as one (1) file.

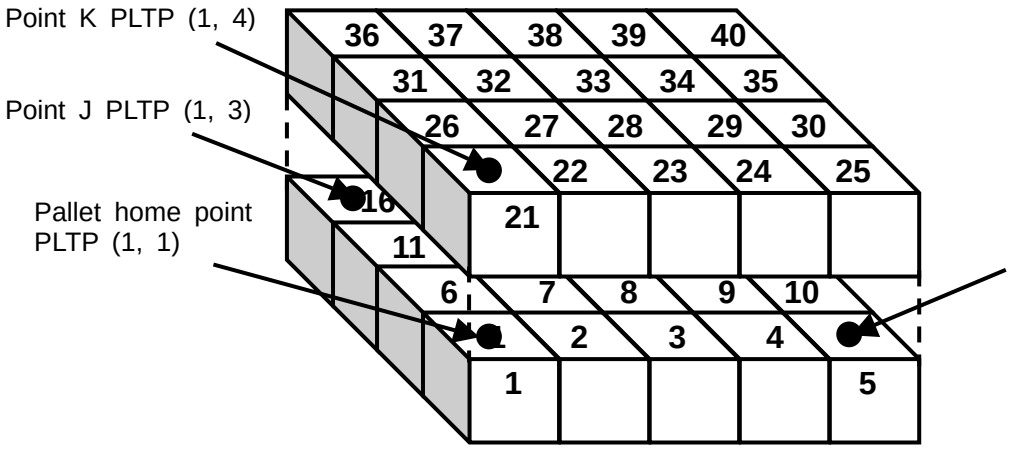
Appendix G Dynamic Link Library

Appendix G–1 Palletizing Library

Library Name	PALLET. LIB
Purpose	Library of palletizing commands. Up to three (3)-dimensional palletizing of (i × j × k) is possible by teaching pallet home point, point i, point j, and point k.
Command	INITPLT (<Pallet number>, <i>, <j>, <k>) The pallet specified by the pallet number is initialized as the three (3)-dimensional pallet of "i × j × k". i : Number of elements between pallet home point and point I j : Number of elements between pallet home point and point J k : Number of elements between pallet home point and point K MOVEPLT (<Pallet number>, <Element number>, X, Y, Z, C) The robot moves to the position which is specified by the pallet number and element number and includes X, Y, Z and C offsets. The X, Y, Z and C offset values cannot be omitted. (Unless offset is effected, specify zero (0).)

[Descriptions of terms]

Ex. Pallet No. 1, 3-dimensional pallet of "5 × 4 × 2":



Pallet number : The pallet number is assigned in turn, starting with number "1" for pallets used in appropriate program.

Teach point : The four (4) points above (i.e., home point, point I, point J and point K) are the teach points of this pallet.
The teach point name is predetermined as "PLTP (<Pallet number>, 1 ~ 4)."

Element number : This number is automatically assigned for pallet elements. For the pallet of "5 × 4 × 2" as exemplified above, numbers 1 ~ 40 are assigned for respective elements.
The palletizing command allows the robot to move a desired position by designating the pallet number and element number.

Library Build-in

To use the "PALLET LIB", the following commands [1] and [2] are required.

[1] In the GLOBAL area of the user program, library build-in should be declared.

LOADLIB PALLET.LIB
↑
Library build-in declaration

[2] In the GLOBAL area of the user program, global variable used in the library should be declared. The variable name is predetermined as "PLTP".

DIM PLTP (<Pallet number>, 7) AS POINT

The pallet number should be any value larger than "1" and its maximum value changes with the number of teach points and number of arrays specified in the program.

Number "7" is a constant and is used to keep the variable area used in the library.

This global variable is used to transfer the number of teach points and calculated values to and from the PALLET LIB.

```
GLOBAL
    LOADLIB PALLET.LIB          [1]  Library build-in declaration.
    DIM PLTP (2,7) AS POINT     [2]  Global variable declaration.
END
PROGRAM MAIN
    :
    (Omitted)
    :
END
```

Analysis and advice

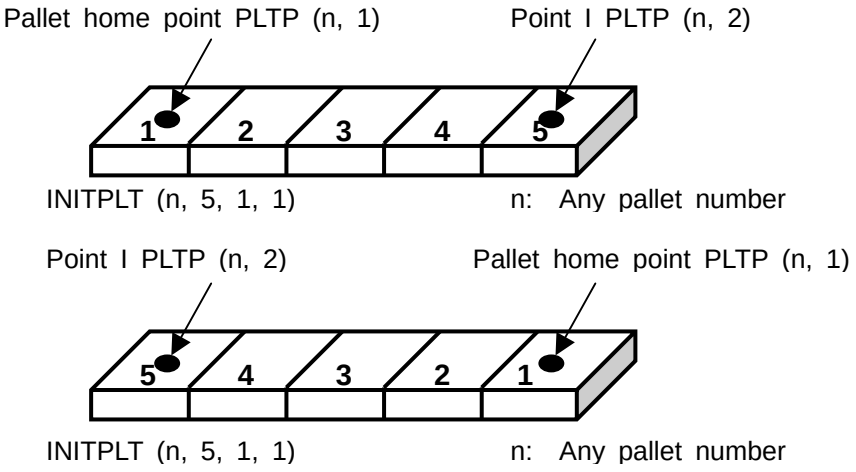
[1] Pallet
The pallet should be set horizontally in the X–Y plane.
(It should not be tilted.)

[2] Teaching and effective data
Teaching of four (4) points PLTP (n, 1) ~ PLTP (n, 4) is performed. (n: Pallet number 1 ~ n)
Pallet home point:
All coordinates of X, Y, Z, C and T can be used as the teach data. The move position is calculated by adding a shift value to this PLTP (n, 1) data.
Point I PLTP (n, 2), point J PLTP (n, 3):
Only X and Y coordinates can be specified as the teach data. They are used to figure out a shift value in the X and Y directions.
Point K PLTP (n, 4):
Only Z coordinate can be specified as the teach data. It is used to figure out a shift value in the Z direction.
For the one (1)-dimensional pallet, teaching of point J PLTP (n, 3) and point K PLTP (n, 4) can be omitted.
For the two (2)-dimensional pallet, teaching of point K PLTP (n, 4) can be omitted.
The teach point name cannot be changed.

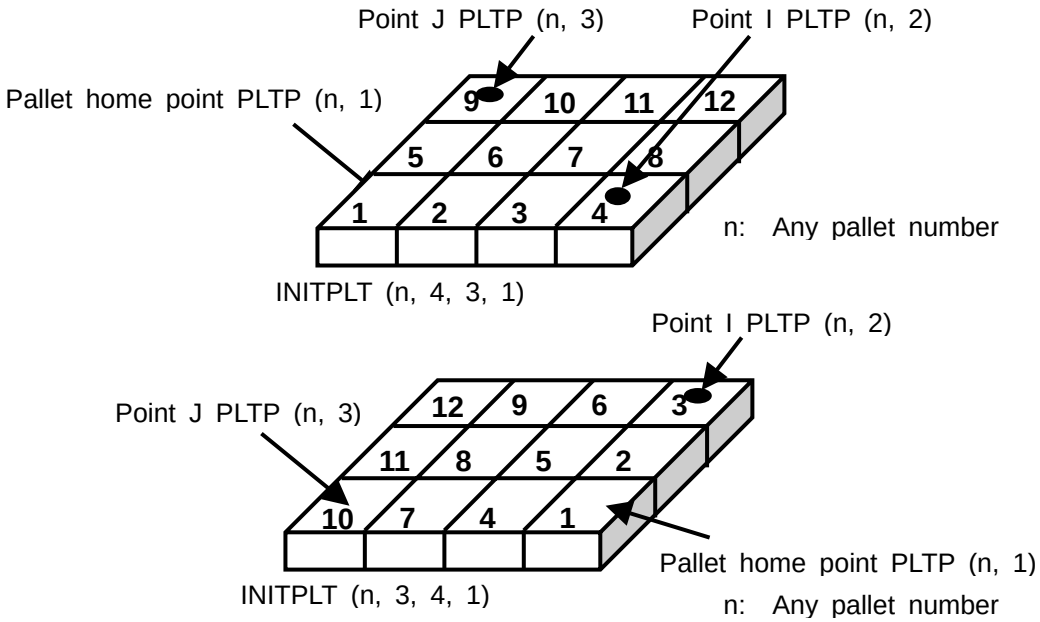
[3] When "PALLET.LIB" is read by the LOADLIB command, variable names used in "PALLET.LIB" (INITPLT****, MOVEPLT**** ; * any number) cannot be used in the user's program as the variable names or teach point names.

[4] Teaching method and element number of pallet
The element number is automatically assigned by the INITPLT command. Even if the pallet is the same, the element number differs with the teaching sequence.

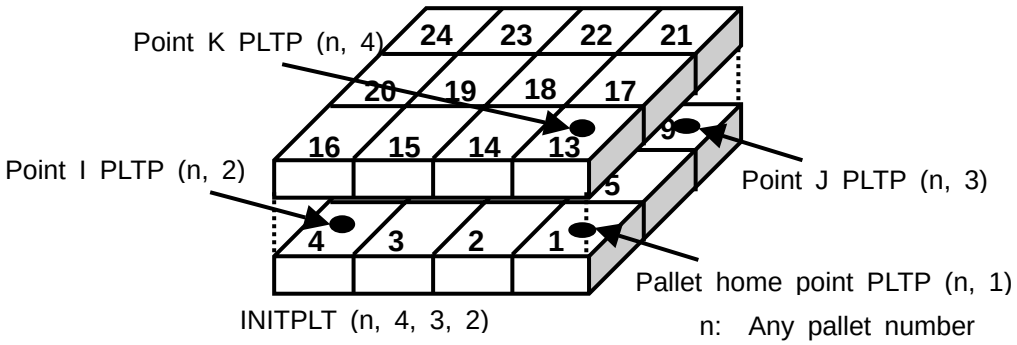
- One (1)-dimensional pallet



- Two (2)-dimensional pallet

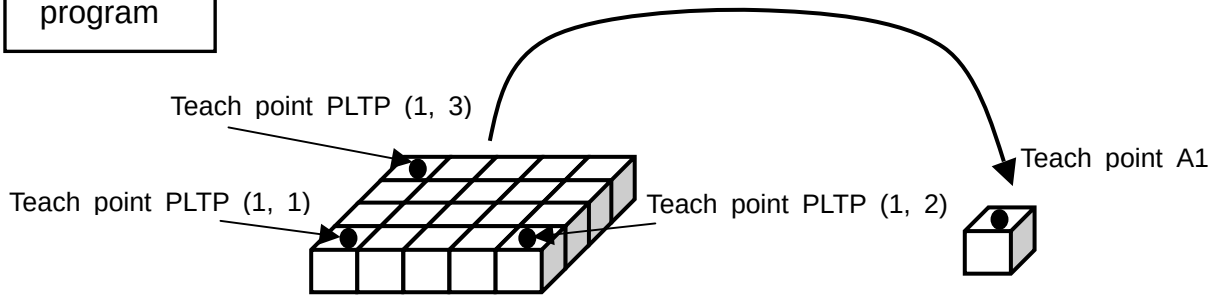


• Three (3)-dimensional pallet



Sample program

[1] When parts are supplied from the pallet to point A1:



```
GROBAL
    LOADLIB PALET.LIB
    DIM PLTP(1,7) AS POINT
END

PROGRAM PALLET
    INITPLT(1,5,4,1)
    OPEN1
    FOR I=1 TO 20 STEP 1
        MOVEPLT(1,I,0,0,50,0)
        MOVEPLT(1,I,0,0,0,0)
     CLOSE1
        MOVEPLT(1,I,0,0,50,0)
        MOVE A1+POINT(0,0,50)
        MOVE A1
        OPEN1
        MOVE A1+POINT(0,0,50)
    NEXT I
END

DATA
    POINT A1          = 650.000, –0.010, 187.140,    2.457,
0.000 / LEFTY
    POINT PLTP(1,1) = 203.346, 390.635, 94.252, 30.261,
0.000 / LEFTY
    POINT PLTP(1,2) = 357.548, 503.825, 94.252, 30.261,
0.000 / LEFTY
    POINT PLTP(1,3) = 337.299, 207.424, 94.252, 30.261,
0.000 / LEFTY
    POINT PLTP(1,4) = 337.299, 207.424, 94.252, 30.261,
0.000 / LEFTY
END
```

Contents of PALLET.LIB

The contents of PALLET.LIB standardly attached to the system are shown below.

```
'*****************************************************************************
'*      TS3000 Dynamic Link Library                                          *
'*                                                                           *
'*      file name : PALET.LIB                                                *
'*      function   : PALLETIZE                                               *
'*      command     : INITPLT(PALLET_NO,I,J,K)                               *
'*                  : MOVEOLT(PALLET_NO,POZITION_NO,X,Y,Z,C   *
'*                                                                           *
'*      Copyright(C) 2008 by TOSHIBA MACHINE CO.,LTD                         *
'*---------------------------------------------------------------------------*
'*   08/08/28New                                                             *
'*                                                                           *
'*****************************************************************************

PROGRAM INITPLT (INITPLTNO,INITPLTI,INITPLTJ,INITPLTK)
'
      INITPLT1P = PLTP(INITPLTNO,1)
      INITPLT2P = PLTP(INITPLTNO,2)
      INITPLT3P = PLTP(INITPLTNO,3)
      INITPLT4P = PLTP(INITPLTNO,4)

      PLTP(INITPLTNO,5) = POINT(INITPLTI,INITPLTJ,INITPLTK)
INITPLT010:
      IF INITPLTI < 1 THEN GOTO INITPLTERR
      IF INITPLTI < 2 THEN GOTO INITPLT015
      INITPLT5I = INITPLTI – 1
      INITPLTXX = (INITPLT2P.X – INITPLT1P.X) / INITPLT5I
      INITPLTXY = (INITPLT2P.Y – INITPLT1P.Y) / INITPLT5I
      GOTO INITPLT020
```

```
INITPLT015:
    INITPLTXX = 0
    INITPLTXY = 0
INITPLT020:
    IF INITPLTJ < 1 THEN GOTO INITPLTERR
    IF INITPLTJ < 2 THEN GOTO INITPLT025
    INITPLT5J = INITPLTJ ─ 1
    INITPLTYX = (INITPLT3P.X – INITPLT1P.X) / INITPLT5J
    INITPLTYY = (INITPLT3P.Y – INITPLT1P.Y) / INITPLT5J
    GOTO INITPLT030
INITPLT025:
    INITPLTYX = 0
    INITPLTYY = 0
INITPLT030:
    IF INITPLTK < 1 THEN GOTO INITPLTERR
    IF INITPLTK < 2 THEN GOTO INITPLT035
    INITPLT5K = INITPLTK – 1
    INITPLTZZ   = (INITPLT4P.Z – INITPLT1P.Z) / INITPLT5K
    GOTO INITPLT040
INITPLT035:
    INITPLTZZ = 0
INITPLT040:
    PLTP(INITPLTNO,6) =
POINT(INITPLTXX,INITPLTXY,INITPLTZZ)
    PLTP(INITPLTNO,7) =
POINT(INITPLTYX,INITPLTYY,INITPLTZZ)
    GOTO INITPLTEND
INITPLTERR:
    PRINT "ERR !!   ELEMENT IS TOO SMALL."
    STOP
INITPLTEND:

END
PROGRAM MOVEPLT
(MOVEPLTNO,MOVEPLTPSN,MOVEPLTX,MOVEPLTY,MOVEPLTZ,MO
VEPLTC)
```

```
        MOVEPLTI = 0
        MOVEPLTJ = 0
        MOVEPLTK = 0
        MOVEPLTPS1 = MOVEPLTPSN –1
        MOVEPLT1P = PLTP(MOVEPLTNO,1)
        MOVEPLT5P = PLTP(MOVEPLTNO,5)
        MOVEPLT6P = PLTP(MOVEPLTNO,6)
        MOVEPLT7P = PLTP(MOVEPLTNO,7)
        MOVEPLTA = MOVEPLT5P.X * MOVEPLT5P.Y
        MOVEPLTB = MOVEPLTPS1 MOD MOVEPLTA
        MOVEPLTMAX = MOVEPLTA *    MOVEPLT5P.Z
        IF 1 > MOVEPLTPSN THEN GOTO MOVEPLTER2
        IF MOVEPLTMAX < MOVEPLTPSN THEN GOTO MOVEPLTER3
        MOVEPLTI = MOVEPLTB MOD    MOVEPLT5P.X
        MOVEPLTJ = INT(MOVEPLTB / MOVEPLT5P.X)
        MOVEPLTK = INT(MOVEPLTPS1 / MOVEPLTA )
        MOVEPLTXXX = MOVEPLTI * MOVEPLT6P.X    + MOVEPLTJ *
MOVEPLT7P.X + MOVEPLTX
        MOVEPLTYYY = MOVEPLTI * MOVEPLT6P.Y    + MOVEPLTJ *
MOVEPLT7P.Y + MOVEPLTY
        MOVEPLTZZZ = MOVEPLTK * MOVEPLT6P.Z    + MOVEPLTZ
        MOVE MOVEPLT1P+
POINT(MOVEPLTXXX,MOVEPLTYYY,MOVEPLTZZZ,MOVEPLTC,0)
        GOTO MOVEPLTEND
MOVEPLTER2:
        PRINT "ERR !!    ELEMENT NO. IS TOO SMALL."
        STOP
MOVEPLTER3:
        PRINT "ERR !!    ELEMENT NO. IS TOO LARGE."
        STOP
MOVEPLTEND:
END
```

Appendix H SCOL Program Language Executing Stop of Pre-Reading

The commands executing stop of pre-reading are listed below.

- PRINT
- WAIT
- ON to DO
- IGNORE
- DOUT
- DIN
- XIN (Option of conveyor)
- BCDIN
- BCDOUT
- PULOUT
- MOVE
- MOVEA
- MOVEI
- MOVES
- MOVEC
- MOVEJ
- DELAY
- LATCH
- SYNC (Option of conveyor)
- UNSYNC (Option of conveyor)
- STOP
- RETURN
- END

APPROVED BY:_____

CHECKED BY: _____

PREPARED BY:_____