EPSON

EPSON ROBOT

User's manual

for SRC-300/320

Rev. 2

EM971R502F

CE

EPSON ROBOT *User's manual for SRC-300/320* EPSON

EPSON ROBOT

User's manual for SRC-300/320

Rev. 2

WARRANTY

The robots and their options are shipped to our customers only after being subjected to the strictest quality controls, tests and inspections to certify their compliance with our high performance standards.

Product's malfunction(s) resulting from normal handling operation will be repaired free of charge up to 12 months after delivery.

However, customers will be charged for repairs in the following cases:

1. Damage or malfunction caused by improper use which is not described in the manual, or careless use.
2. Malfunctions caused by customers' unauthorized disassembly.
3. Damage due to improper adjustments or unauthorized repair attempts.
4. Damage caused by natural disasters such as earthquake, flood, etc.

SERVICE CENTER

Contact the following service center for robot repairs, inspections or adjustments.
Please have the model name, M. CODE, software version and a description of the problem ready when you call.

MANUFACTURER

SEIKO EPSON CORPORATION

Okaya Plant No. 2
1-16-15, Daiei-cho
Okaya-shi, Nagano-ken, 394
Japan

TEL: 81-266-23-0020 (switchboard)
81-266-24-2004 (direct)
FAX: 81-266-24-2017

NOTICE

- No part of this manual may be copied or reproduced without authorization.
- The content of this manual is subject to change without notice.
- Seiko Epson asks that you please notify it if you should find any errors in this manual or if you have any comments regarding its content.

Safety Precautions

Before using these products, be sure to read the following safety precautions as well as the product manual and other relevant manuals.

The safety section of this manual explains minimum safety requirements that users must follow when building and using the robot system. Please read the manual carefully before commencing system design work and implement the appropriate safeguards.

After reading these materials, keep them in a place where they can be easily retrieved or reference if questions or problems arise.

⚠ WARNING

- The robot system manufacturer/supplier shall design and construct robot systems in accordance with the principles described in "Safety section" of this manual. Please read this manual first.

- This robot has been designed and manufactured strictly for use in a normal indoor environment. Do not use the robot in an environment that exceeds the conditions set forth in the manuals for the manipulator and controller.

- Do not use the robot in excess of the usage conditions and product specifications described in the manuals. Doing so will not only adversely affect the life of the product but may also present a serious safety problem.

- Only trained personnel should be allowed to design, install, operate, perform function test, and maintain this robot and the robot system.
  Trained personnel are those who have taken a robot training course held by the dealer or those who have carefully read the manuals and have equivalent knowledge or skill.

FOREWORD

MANUALS

1. **User's manual**

   A manual that gives a general description of robots. It describes such things as safety precaution, operating methods, teaching methods, programming methods, and file management.

2. **Manipulator manual**

   A manual for the manipulator itself. It describes such things as robot installation, motion range, safety, and hands.

3. **Robot controller manual**

   A manual for the robot controller. It describes such things as installation, switch settings, and connection with peripheral equipment.

4. **Reference manual**

   A manual that describes the commands for the SPEL III robot language.

5. **Maintenance manual**

   A manual that describes the maintenance procedure of the robot. It describes such things as check points, troubleshooting, how to repair and so on.

6. **Operating unit manual (option)**

   A manual for the operating unit that describes such things as operating methods.

7. **Programming support software manual (option)**

   A manual for the program development support software. It describes such things as operating environment and operating methods of SPEL Editor or SPEL for Windows.
   We provide two kinds of software, SPEL Editor (for MS-DOS) and SPEL for Windows (for Microsoft Windows). We also provide Vision Guide, the integrated robot vision system, as an option of SPEL for Windows.

8. **Teaching pendant manual (option)**

   A manual for the teaching pendant. It describes such things as how to operate the teaching pendant.

Command entry format

Enter commands according to the following format rules.

Simple character string	:Enter the string as it is shown. Example: MOTOR ON
[ ] (square brackets)	:Use these to indicate a type of data. Example: ON [output bit number]
| | (vertical lines)	:Use these to indicate options. Example: MOTOR |ON | |OFF|
{ } (brackets)	:Use these to indicate omittable settings (switches, etc.). The function differs depending on whether or not the setting is omitted. Example: DIR {/W}
{ }n (brackets + n)	:Use these to indicate that the setting in brackets will be repeated "n" times (enter a number as "n"). Simply entering an "n" (not a number) indicates that the setting can be repeated.
~ (tilde)	: Use this to indicate that the command format continues onto the next line or from the previous line.

For SPEL for Windows users

We don't recommend using SPEL for Windows with SPEL Editor. If using the SPEL for Windows with SPEL Editor, see "7. How to Use the Teaching Pendant and SPEL Editor" of SPEL for Windows manual.

TABLE OF CONTENTS

SAFETY SECTION

INTRODUCTORY SECTION

ELEMENTARY SECTION

APPLIED SECTION

SAFETY SECTION

CHAPTER 1. BASIC FUNCTIONS FOR SAFETY

The major safety functions of our robots are explained in this chapter. These explanations represent the minimum knowledge necessary for designing a system.

The explanations in this chapter are purposely kept simple and brief in order to give you a general understanding of safety. Please refer to the controller manuals and other relevant manuals for details and build a safe system based on a solid understanding of the characteristics of our robots.

1. 1 Low Power and High Power

The robot has two motor power modes: low power and high power. In low power mode the robot's speed of motion and torque are reduced. Conversely, in high power mode, the robot can be operated at the programmed speed and specified torque.

Reduced speed is designed to give operators time to avoid danger if the robot should unexpectedly malfunction for some reason. Reduced torque is designed to prevent operators from sustaining serious or fatal injuries if they should be struck by the robot. Each robot model has its own predetermined maximum values for reduced speed and reduced torque. Users cannot change these.

For safety reasons the initial state of robots is always the low power mode. The robot will not shift to high power mode unless the specified command is executed.

These motor power modes are committed to memory. Nevertheless, even in high power mode, the motor control is forced into a low power state that is identical to low power mode when the safeguard opens. (The motor power of the robot reverts to the high power state after the safeguard is closed.) The robot is also automatically reset to low power mode whenever the operation mode (TEACH/AUTO) is switched or the controller is reset.

	Low power mode	High power mode
Safeguard open	low power	low power
Safeguard closed	low power	high power

Multiple protect circuits and mutual monitoring circuits in the controller prevent the robot from running out of control and exceeding the specified reduced speed and reduced torque even if a single failure occurs in the low power mode.

1.2 Safeguard

REMOTE1 connector of controller has an input circuit that is connected to the safeguard's interlock switch.
This safeguard input operates as follows:

When AUTO is the operation mode

Safeguard closed: Automatic operation is possible

Safeguard open: Robot stops immediately and enters the low power mode. Robot cannot be restarted until the safeguard is closed and a START signal is input.

When TEACH is the operation mode

Safeguard closed: Robot can operate even at the high power for program verification

Safeguard open: Robot stops immediately but can only operate at the low power thereafter for teach operations.

Refer to (6) - e) in section 2. 3 of this manual and "4. REMOTE 1" of controller manual.

For SRC-320 only

The safeguard input circuit is dual-redundant. An open safeguard is always detected even if one of the circuits fails. An error is displayed in the event of a failure. Therefore, we strongly recommend dual-redundant wiring for the safeguard interlock switch.

In case of SRC-320, once an open safeguard is detected by the safeguard input circuit, the signal will be latched, and even after the door is closed, it will not be recognized as such until the latch release signal is input.
This has been provided as a means of preventing the kind of confusion that can easily arise during the construction of a robot system when operator and the robot system itself perceive the state of the safeguard differently; that is, when one believes the door is open, while the other perceives it as being closed. (For example, when faulty adjustment causes a difference in the state of the door and the state of the interlock switch; or when the door closes without the operator intending for it to be closed; etc.)

1.3 Emergency stop

The robot controller is equipped with emergency stop input terminals. When the normally-closed emergency stop switch is pressed, the power that is supplied to the motor is shut off and the robot is stopped by means of dynamic brake.

The trajectory of robot motion before stopping and the exact point at which it will stop after the emergency stop switch is pressed cannot be specified. In most cases the robot will not travel beyond the taking point that was specified prior to execution of the emergency stop; however, overrun may occur depending on the robot's load and speed of motion. Therefore, associated equipment needs to be installed an adequate distance away from the robot.

An emergency stop output terminal that is interlocked with the robot's emergency stop is provided. User can use it to control associated equipment.

1.4 Enable Switch (Dead Man Switch) for SRC-320 only

An enable switch is provided for persons working within the safeguarded space. In TEACH mode the robot only operates when the switch is being pressed and it stops when the switch is released. Robot operation cannot be reinitiated simply by pressing the enable switch.

The enable switch is located on the side of the teach pendant, TP-320. To operate the robot using the pendant, the enable switch must be pressed continuously, regardless of whether the safeguard is open or closed.

A personal computer is used to perform operations from outside the safeguarded space for program development. Program development support software for PC (SPEL Editor, SPEL for Windows) has a teaching function. Please connect a PC cable that is equipped with an enable switch (available as an option) when using this function. When a PC cable that is not equipped with an enable switch is connected, please never use the PC in the safeguarded space.

CHAPTER 2. RECOMMENDATION FOR SAFETY

2.1 General

This chapter describes the basic precautions that must be taken to ensure safe use of the robot and robot system.

Please carefully read this chapter and keep safety foremost in mind when using this robot equipment. Specific tolerance values, usage conditions and such matters necessary for ensuring safety are explained in the controller and manipulator manuals.

Matters covered here are based on ISO 10218 (Manipulating industrial robots - Safety), but safety measures have been made more concrete and some new items have been added.

Some safety standards related to robots and robot systems are listed below. Please implement full safety measures after referring not only to this chapter but to each standard. (Note: The following is only a partial list of necessary safety standards.)

EN775	European Standard; Manipulating industrial robots - Safety
ANSI/RIA R15.06	American National Standard; Industrial Robots and Robot Systems - Safety Requirements
IEC204-1 (EN60204-1)	Safety of machinery - Electrical equipment of machines Part 1. Specification for general requirements
EN292-1,-2	Safety of machinery - Basic concepts, general principles for design Part 1. Basic terminology, methodology Part 2. Technical principles and specifications
EN418	Emergency stop equipment, functional aspects - principles for design
prEN953	General requirements for design and construction of guards

2.2 General design requirements

The robot system manufacturer/supplier shall design and construct robot systems in accordance with the principles described in this clause and the next clause.

(1) Failure to safety

The robot system shall be designed, constructed, and implemented so that in case of foreseeable failure of any single component safety functions are not affected or when they are, the robot system is left in a safe condition.

(2) Electrical equipment

The application of the electrical equipment of the robot and robot system shall be in accordance with IEC204-1.

(3) Power supply

The power supply and grounding (protective earth) requirements shall be in accordance with the specifications described in the controller manual.

(4) Isolation of power sources

Each robot system shall have means to isolate each of its power sources. These means shall be located in such a way that no person will be exposed to hazards and they shall have a lockout/ tagout capability.

(5) Environmental requirements

The environmental requirements shall be in accordance with the specifications described in the manipulator and controller manuals.

2.3 Design and safeguarding of the robot system

(1) Physical arrangement of the robot system

a) When designing the layout of a robot system, please take adequate care to secure enough space between the robot and the various pieces of associated equipment so that they do not interfere with one another. When an emergency stop switch is pushed, the robot may travel a path that differs from the normal path of operation before stopping, thus a layout design that takes this into account is necessary. Designers also need to refer to the manipulator manuals and lay out the system so that there is enough room to perform maintenance and inspections.

b) When designing a robot system having a restricted range of motion, please restrict the range of motion in the manner described in the manuals for the manipulator. Please be certain to take steps to restrict the range of motion by means of both software and mechanical stops.

c) Please design the wires and lines (hoses) of the robot end effector, in such a way that work-pieces in its grasp are not thrown even when the robot system's power is suddenly shut down or system failure is occurred.

d) Please design the weight and moment of inertia of the robot end-effector within the tolerance limits. Using the robot beyond the tolerance limits puts an enormous load on the robot. This not only shortens product life but also may invite an unpredictable, dangerous situation due to the external force applied to the end-effector and work-piece.

e) Please design the loading and unloading of parts and materials to the robot system so that operator safety can be fully ensured. When it is necessary to load and unload parts without stopping the robot, a shuttle device needs to be installed or some other means needs to be designed so that operators do not have to enter the hazardous zone.

f) In cases in which a multiplicity of robots are used in one system, the layout must be designed so that their maximum operating areas will not interfere with each other.

(2) Shut down

Shutting down (removing power to) the robot system or any associated equipment shall not result in a hazardous situation.

Not only is it necessary to prevent the danger presented by the throwing of work-pieces as mentioned above in (1); it is also necessary to verify the safety of the robot's associated equipment. Does the equipment stop safely? Does the removal of motive power conversely result in a dangerous situation? These and other questions need to be answered.

(3) Emergency stop

Each robot system work station shall have a readily accessible emergency stop device. Any emergency stop push-button switch that is employed shall conform to relevant safety standards, e.g. EN418, IEC204-1, ISO10218.

(4) Control from remote locations

The robot controller is designed based on the idea of "a single point of control" in order to prevent danger by operating the robot from a remote location. Similarly, safety measures for the robot system as a whole are necessary to prevent hazards by allowing associated equipment to be started and stopped from a remote location.

(5) Safeguards

The robot system needs to be equipped with safeguards to ensure safety. Please perform a risk assessment to determine the additional space required beyond the restricted space to define the safeguarded space. (References: prEN953 6. Guard Design)

Safeguards shall

a) be constructed to withstand foreseeable operational and environmental forces;

b) prevent access to the safeguarded space except through openings equipped with interlocks or presence sensing devices;

c) be permanently fixed in position and only be removable with the aid of a tool; and

d) be free of sharp edges and projections and shall not themselves be a hazard.

(6) Designing, installing and adjusting interlocking guards

a) Please be sure to use a switch that conforms with safety standards (EN1088, EN60947-5-1, etc.) for the safeguard interlock. (The major requirements for the switch are that it be a two contact point type having a forced opening mechanism and that it have a protection level suitable for the surrounding environment.)

b) As a precaution against switch failure and other unforeseeable events, the interlock switch should be arranged so that the switch is forced to be pressed (the contact point is opened) when the gate is opened. Interlock switches that simply open the contact point by means of the spring action of the switch itself when the gate is opened are not suitable.

c) Where whole-body access to the safeguarded space can be gained through an interlocking door, a device which prevents inadvertent closing of the door should be provided.

d) Please connect the interlock switch to the robot controller’s safeguard inputs. We recommend dual-redundant wiring as a precaution against failure.

e) Please install a switch to cancel the latch of the "safeguard open" outside the safeguarded space and in close proximity to the door.

(7) Presence sensing device

The safeguard interlock described above can be thought of as a presence sensing device in that it indicates the possibility that a personnel is present in the safeguarded space. Before a different presence detecting device is to be installed, please perform a full risk assessment and take the utmost care to ensure its reliability.

Whenever presence sensing devices are used for safety purposes, they shall comply with the following.

a) A presence sensing device shall be installed and arranged so that persons cannot enter and reach into a hazardous area without activating the device or cannot reach the restricted space before the hazardous conditions have ceased.

b) Their operation shall not be adversely affected by any of the environmental conditions for which the system was intended.

c) Resumption of robot motion shall require the removal of the sensing field interruption. This shall not be the control to restart automatic operation.

(8) awareness means

Awareness barrier or awareness signal may be used in addition to but not as a substitute for the safeguards.

(9) Safe working procedures

It is recognized that for certain phases of the robot system life (e.g. commissioning, process changeover, cleaning, and maintenance) it may not be possible to design completely adequate safeguards to protect against every hazard or that certain safeguards may be suspended. Under these conditions, appropriate safe working procedures shall be used.

(10) Reset of safeguards

Restarting the system shall require a deliberate action from outside the safeguard space. Reestablishing the interlocked door shall not in itself restart automatic operation of the robot system. (Robot itself is designed as so.) The restarting device shall be located so that it cannot be reached from inside the safeguarded space and should be located in a manner to afford a view of the safeguarded space.

2.4 Use and care

(1) General

Please store pendants that are not in use out of reach to prevent an operator from mistakenly pressing the emergency stop switch on a pendant that is not connected to the controller during an emergency.

(2) Automatic (normal) operation

Automatic operation shall only be permissible when

a) the intended safeguards are in place and functioning.

b) no personnel are present within the safeguarded space, and

c) proper safe working procedures are followed.

Please equip the pendant key holder with an operating **mode selection key** and take steps so that only the personnel holding the pendant can change the mode. The **mode selection key** must not be left on the operating unit outside the safeguarded space when a personnel is inside the safeguarded space.

(3) Teaching

(Prior to teaching)

a) The teacher shall be trained on the type of robot used in the actual robot system and shall be familiar with the recommended teaching procedures including all of the safeguarding methods.

b) The pendant shall be tested to ensure proper operation. Any faults or failures shall be corrected prior to teaching.

c) Before entering the safeguarded space, the teacher shall ensure that all necessary safeguards are in place and functioning.

(During teaching)

a) The robot system shall be under the sole control of the teacher within the safeguarded space. (As the robot is designed with single point of control, so the robot system shall be designed.)

b) Movement of other equipment in the safeguarded space which can present a hazard shall either be prevented or under the sole control of the teacher. When under the control of the teacher, it shall require deliberate action on the part of the teacher separate from the action to initiate robot motion.

c) All robot system emergency stop devices shall remain functional.

(Returning to automatic operation)

The teacher shall return the suspended safeguards to their original effectiveness prior to initiating automatic operation of the robot system.

(4) Programming data

Programmed data shall be stored in a suitably protected environment when not in use.

(5) Program verification

When the program verification with High Power Mode is necessary, it shall be made with all persons outside the safeguarded space.
When it is necessary to perform program verification with personnel inside the safeguarded space, the same precaution and procedure as teaching shall apply.

(6) Trouble shooting

Trouble shooting shall be performed from outside the safeguarded space. When this is not practicable, the following requirements shall apply.

a) personnel responsible for trouble shooting are specifically authorized and trained for these activities;

b) personnel entering the safeguarded space shall use the teaching pendant equipped with an enabling device to allow motion of the robot;

c) safe working procedures are established to minimize the exposure of personnel to hazards within the safeguarded space.

(7) Maintenance

a) The maintenance manual is attached to the robot. Additionally the robot system shall have an inspection and maintenance program to ensure continued safe operation of the robot system. Maintenance of the robot or robot system shall be performed in accordance with the maintenance manuals.

b) Personnel who perform maintenance or repairs on robot or robot systems shall be trained in the procedures necessary to perform safely the required tasks.

c) When it is necessary to perform maintenance within the safeguarded space, the followings shall be taken.

(1) The robot system shall be shut off using a lockout/tagout procedure.

(2) Alternatively, intervention within safeguarded space while power is available to the robot shall require the followings.
Prior to entering the safeguarded space, a visual inspection of the robot system shall be made to determine if any conditions exist that are likely to cause malfunctions. If pendant controls are to be used, they shall be functionally tested prior to such use to ensure their proper operation. If any damage or malfunction is found, required corrections shall be completed and retesting shall be performed before personnel enter the safeguarded space.
Personnel performing maintenance or repair tasks within the safeguarded space shall have total control of the robot or robot system. All robot system emergency stop devices shall remain functional.
The suspended safeguards shall be returned to their original effectiveness prior to initiating automatic operation of the robot system.

2.5 Installation, commissioning and functional testing

(1) Installation

The robot or robot system shall be installed in accordance with the manufacturer's manual. ISO9946 shall be consulted for additional guidance during installation.

(2) Commissioning and functional testing

When the safeguarding methods are not in place prior to commissioning and functional testing, interim means of designating the restricted space shall be in place before proceeding.
During the commissioning and functional testing, personnel shall not be allowed in the safeguarded space until the safeguards are functional.
An initial start-up procedure shall include, but is not necessarily limited to, the following.

a) Before applying power, verify that

- the robot has been properly mechanically mounted and is stable,
- the electrical connections are correct and that the power (i.e. voltage, frequency, interference levels) is within specified limits,
- the compressed/vacuum air is properly connected and within specified limits,
- the peripheral equipment is properly connected,
- the interlocking switches are applied on the safeguard and they are correctly functional, and
- the physical environment is as specified in the manipulator and controller manuals.

b) After applying power, verify that

- the start, stop, and mode selection control devices function as intended,
- each axis moves and is restricted as intended,
- emergency stop circuits and devices are functional,
- it is possible to disconnect and isolate the external power sources,
- the teach and auto facilities function correctly,
- the safeguards and interlocks function as intended,
- other safeguarding is in place
- in reduced speed, the robot operates properly, and
- the robot has the capability to perform the intended task at the rated speed and load.

(3) Restart of the robot system after modification

A procedure for the restart of the robot system after hardware, software or task program modification, repair, or maintenance shall include but not necessarily be limited to the following:

a) check any changes or additions to the hardware prior to applying power;

b) functionally test the robot system for proper operation.

2.6 Documentation

The robot system documentation shall contain the documents of all the components included in the system with their identification (e.g. robot, associated equipment, safeguards).

It shall also as a minimum include the following:

a) a clear, comprehensive description of the robot system and its installation including mounting and connection to external power sources;

b) a description of foreseeable hazardous conditions and how to avoid them;

c) a description (including interconnecting diagrams) of the safeguards, interacting functions, and interlocking of guards with hazardous conditions particularly with interacting installations;

d) any further instructions for use specific to the system.

2.7 Training

The user shall ensure that personnel who program, operate, maintain, or repair robots or robot systems are adequately trained and demonstrate competence to perform their jobs safely. Training shall include, but is not limited to, the following:

a) a view of applicable standard safety procedures and the safety recommendations of the robot manufacturer and robot system designers;

b) a clear definition of assigned tasks;

c) identification and explanation of all control devices and their functions used in performing the assigned task;

d) identification of the hazards associated with the assigned task;

e) the designated method(s) of safeguarding including the safe working procedures from the identified hazards;

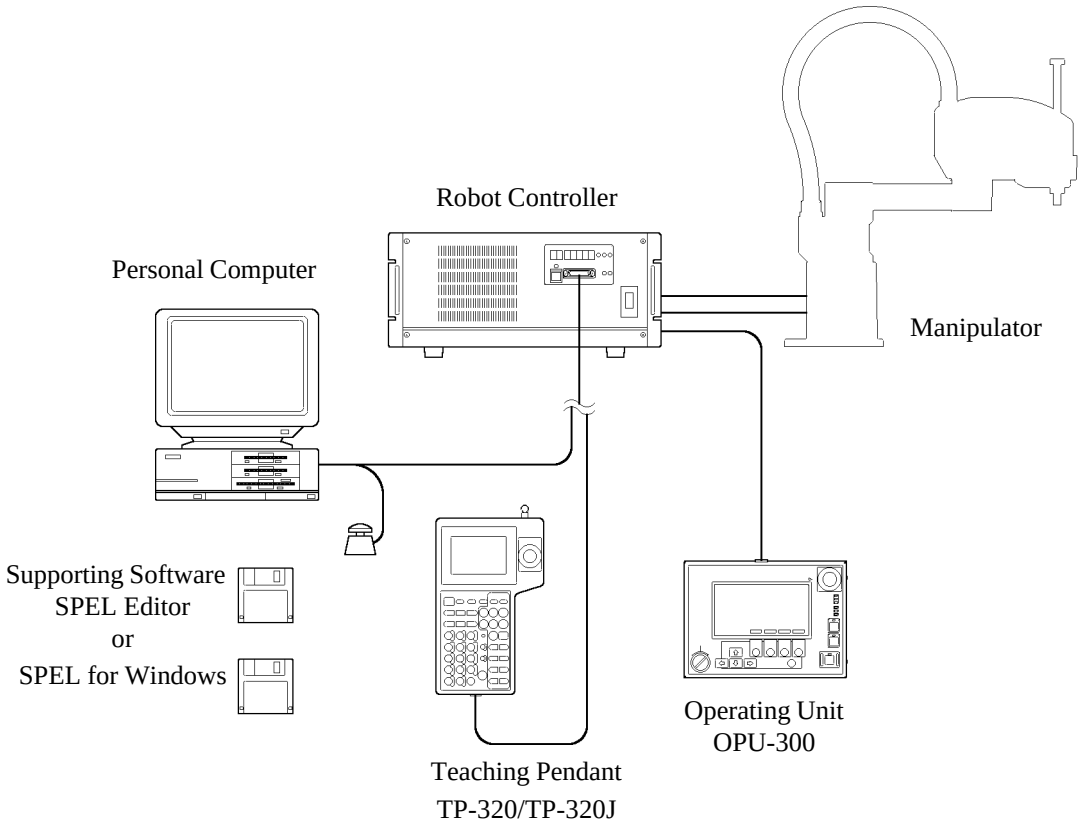
f) the method for testing or otherwise ensuring the proper functioning of the safeguards and interlocks.

INTRODUCTORY SECTION

CHAPTER 1. PREPARATION FOR OPERATION

1. 1 Robot Components

The basic components of our robot is shown in the following diagram.



NOTE Components other than the manipulator and robot controller are optional.

Options

SPEL Editor
The program development support software for MS-DOS.

SPEL for Windows
The program development support software for Microsoft Windows.

Operating unit (OPU-300)
A multifunction robot operating unit with a large, backlighted LCD display.

Teaching pendant (TP-320/TP-320J)
Small operating unit for teaching jobs.

1.2 Installation Precautions

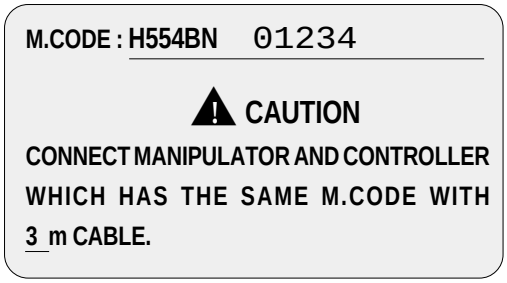
First, read the precautions in the manipulator and controller manuals. Below are some additional precautions to be noted.

Check M. CODE and cable length

Each controller is set and adjusted for a particular manipulator, and breakdowns or other problems may occur if a different type of controller is connected. Therefore, an "M. CODE" label is attached to each manipulator and controller to indicate correct matches between manipulators and controllers. On most models, this label is attached on the rear of the unit.

> **⚠ CAUTION** When connecting the manipulator and controller, be used that the two devices have the same M. CODE.

M. CODE label



Arm fastener

When shipped, the robot arm is held in place by a special arm fastener. Do not remove this arm fastener until the manipulator has been secured on its base table. Be sure to remove the arm fastener before turning on the power.
See the manipulator manual for further description of the arm fastener.

1.3 Programing Unit

The equipment, called the programming unit, is used to teach the points and to create programs in order to operate the robot. Generally, a personal computer is used with the programing software, SPEL Editor or SPEL for Windows.

Preparation for using programing unit

The preparation for utilization of the programing unit is described in "1.3 Preparations" of SPEL Editor manual or "Setup" of SPEL for Windows manual. Refer to each manual for details.

1.4 Mode

Mode on controller

The controller has two modes: TEACH and AUTO. When operating the robot, you need to select the mode corresponding to the operating equipment. Note that the method of operation differs with the mode selection. There are two ways to switch between these modes.

(a) Key switch on operating unit
Turn the operating unit's key switch to change the mode.

(b) Send mode signal to REMOTE2 connector (when operating unit is not used)
Send a mode signal to the pin on the REMOTE2 connector that corresponds to the desired mode. For further description, see the section entitled "If OPU-300 is not used" in the controller manual.

The functions of the two controller modes are described below.

TEACH mode

This mode is used for teaching, programming, and debugging robots when the controller is connected to a programming device (PC) or teaching pendant. No matter which device is connected, this mode operates by sending instructions to the robot via the controller's TEACH connector. Connection to a computer is made via the computer's RS-232C connector.

The following configuration is used to enable communications via the TEACH connector. The same communication settings must also be made on the computer.

9600 bps, 8 bits, even parity, 2 stop bits

The console in TEACH mode

The device that sends commands to the robot is called the console. In TEACH mode, the console is assigned to the device connected to TEACH port on front panel of the controller. Also you can connect the PC to RS-232C (#20) port on back of the controller by setting the software switch. (for SPEL for Windows only)

NOTE

The #21 port of RS-232C can not be the console.

⚠ CAUTION

In case the SSW3-1 of software switch is on, the PC must be connected #20 of RS-232C port. If the PC is connected to TEACH port, the <Pendant> button on SPEL for Windows doesn't work.

• TEACH port

It is standard setting. The teaching pendant or PC is connected to the TEACH port on front panel of the controller.

• RS-232C port #20

You can connect the teaching pendant to the TEACH port on the front panel of the controller and simultaneously you can connect the PC with SPEL for Windows to the RS-232C port #20 on the rear panel of the controller. By changing the console, you can do the teaching work and making programs or debugging in parallel without changing the cable. This function is supported only by SPEL for Windows and teaching pendant TP-320/TP-320J.

When using RS-232C port #20, turn the software switch SSW3-1 on beforehand. By turning on the controller power after changing the software switch, RS-232C port #20 will be console. How to change the console is as follows.

1). In the case of changing the console from teaching pendant to PC, push the RELEASE key on teaching pendant.

2). In the case of changing the console from PC to teaching pendant, use the <Pendant> button on [Robot Control Panel] dialog box of SPEL for Windows.

⚠WARNING

Don't carry out teaching operation from PC connected #20 port of RS-232C. Because emergency stop switch with PC cable connected to #20 port does not function as emergency stop. Therefore, to move the robot is very dangerous.
While debugging a program with PC which is connected to RS-232C port #20, when you move the robot, it is required to have the teaching pendant (TP-320/TP-320J) or OPU-300 which is connected to TEACH port in hand, in order to press the emergency stop switch in case of an emergency.

The motor power status in TEACH mode

In TEACH mode, there is possibility that the worker is near the robot. Therefore the robot is usually in low power state in TEACH mode because of keeping the worker safe. In order to operate the robot in high power state (programmed speed) in TEACH mode, it is necessary to close the safeguard and cancel the low power state by POWER HIGH (LP OFF) command. Even if POWER HIGH (LP OFF) command is executed when the safeguard is opened, the motor power status doesn't change into high power state. In similar ways, even if the safeguard is closed, the robot keeps low power mode unless executing POWER HIGH (LP OFF) command.

When in TEACH mode, the robot is able to be operated at low speed even when the safeguard is open to enable teaching.

AUTO mode

This is the mode used for robots when they operate in the factory. The commands which are sent to the robot to start or pause a program can be sent via the OPU-300 operating unit, the REMOTE3 connector, or the RS-232C port on rear panel of controller.

The console in AUTO mode

The device that sends commands to the robot is called the console, and it is specified using the CONSOLE command. There are three ways to specify the console in AUTO mode.

- If using the OPU-300 (REMOTE2) as the console:

  This device is equipped with push-button switches for starting and pausing programs. It also includes a display for the I/O monitoring function and error messages.

- If using REMOTE3 as the console:

  Robot can be controlled using REMOTE3 (it can start, pause or stop a program and calibrate robot) via a sequencer or commercial control panel.

- If using the S. NET (RS-232C port #20, #21)

  The RS-232C port provides a communications port for sending commands to the robot from a computer. It also enables various kinds of data (production control data, etc.) to be sent to the computer from the robot so that the computer can process the data. Refer to "7.4 Communication between Host Computer and Robot" in applied section.

The motor power status in AUTO mode

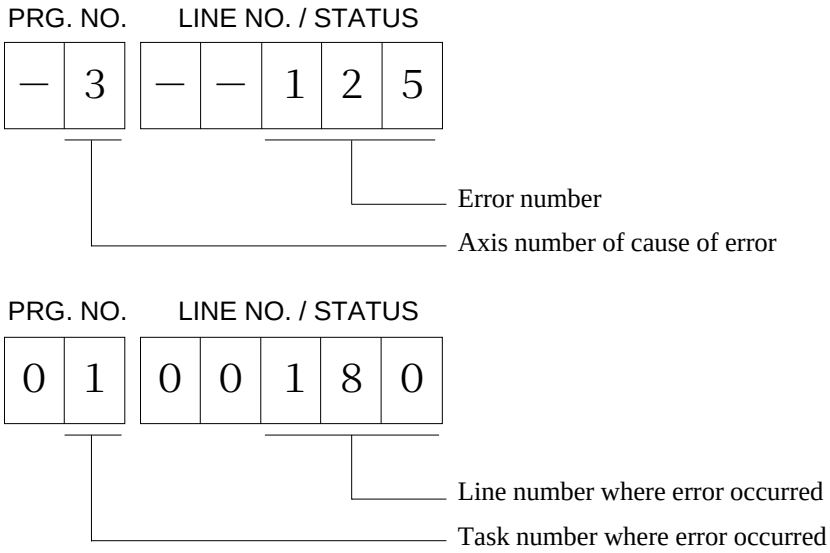
In AUTO mode, the robot is never able to operate when the safeguard is open because it means that there is someone near the robot. If you attempt to operate the robot when the safeguard is open, the robot immediately enters "quick pause" mode and cannot be operated. To cancel quick pause mode and enable operation, you must shut the safeguard and send a start signal.
Regarding motor power state in AUTO mode, refer to the transition diagram of controller manual. It is different between SRC-300 and SRC-320 controller.

1.5 Error Message

When an error occurs in the robot controller an error message will appear as follows. At the same time, an error message will appear on the operating unit OPU-300 and the programming unit (PC).

LED of controller indication panel

① 7-segment LED



The above mentioned contents are expressed alternatively.
The error number and the axis number is expressed with a hyphen (-).

② Other LEDs

E. STOP	Lit up when emergency stop is input. The LED keeps lighting until the robot is reset even if getting rid of cause of emergency stop.
ERROR	Lit up when error has occurred. Does not light in the case of an emergency stop or system error.
S. ERR	Lit up when the main CPU cannot function because of trouble in the hardware. In this case proper display will not appear on the seven LEDs.
SAFE GUARD	Lit up when safeguard is open. (when the status of the interlock switch that is connected to safeguard input terminals of REMOTE1 connector is open.)

Output to OPU-300 and REMOTE1 connector

① Liquid crystal display (OPU-300)

When an error occurs, an error number, line number, task number and error content will appear on OPU-300 as follows.

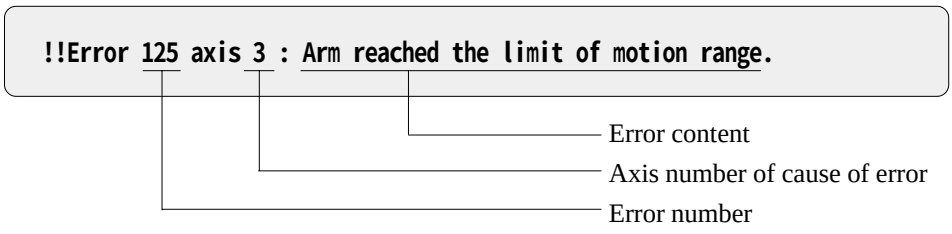
```
!!Error 125,line 160,task 1
Arm reached the limit of motion range.
```

② Output to OPU-300 LED and REMOTE1 connector

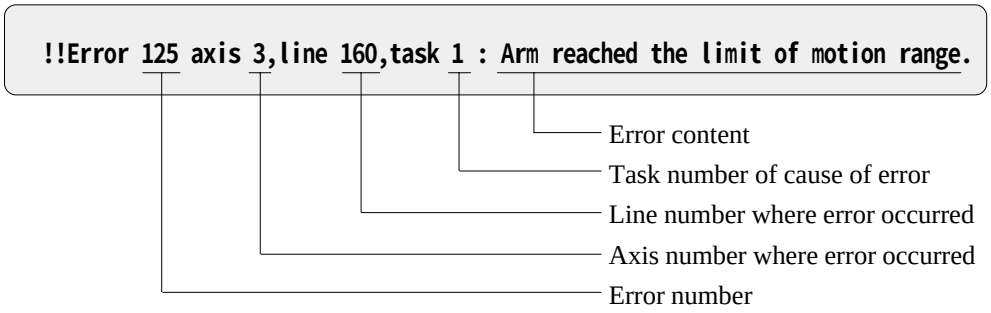
E. STOP	Lit up (output) when there is an emergency stop input.
ERROR	Lit up (output) when error has occurred. Does not light in the case of an emergency stop or system error.
S. ERR	Lit up (output) when the main CPU cannot function because of trouble in the hardware. In this case proper display will not appear on the seven LEDs.
SAFE GUARD	Lit up (output) when safeguard is open. (when REMOTE1's safeguard switch input is released.)

Output to programming unit

If the error is caused by inputting a command then it is shown as:



If it is an error while running a program then it is shown as:

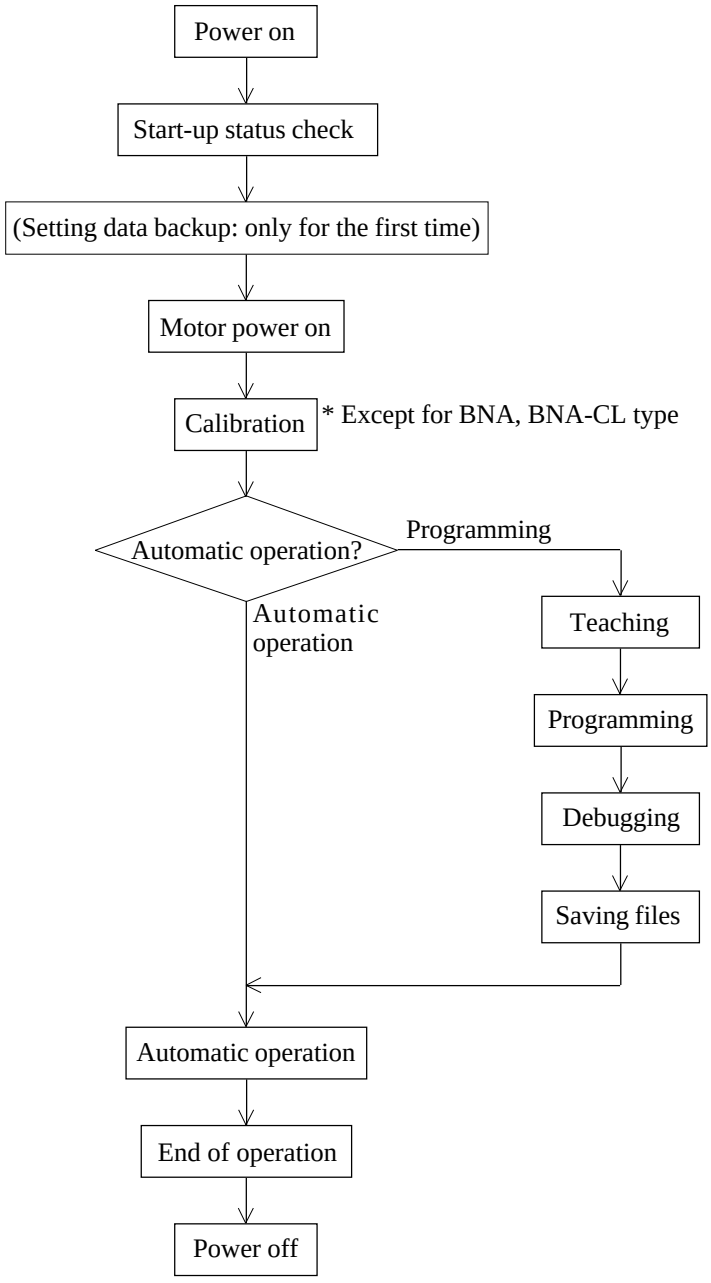


CHAPTER 2. BASIC OPERATION

In this chapter we will show the methods of basic operation and give examples of teaching and programming. An outline of how to use our robot is given in this chapter.

2.1 Basic Operation Flow

The basic operation is shown in the following flowchart.



Each content is described in the following page.

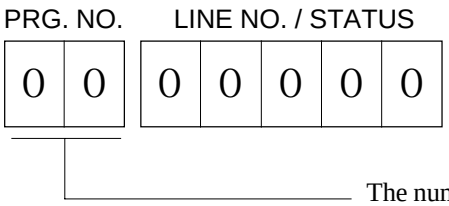
2.2 Checking the Start-up Status

① Make sure that the arm fixture of manipulator is removed.

② Turn on the POWER switch on front panel of controller.

③ There is the display equipment such as the LED on front panel of controller and operating unit OPU-300, check to see that there are no abnormalities. The proper start-up status is as follows:

• The LINE NO./STATUS on indicator panel of controller is shown as "0".



NOTE

In case of BNA or BNA-CL type manipulator, an error 119 may occur when the power is turned on. This means that position data when the power was cut off and when turned on differ.
If you moved the arm manually after the power is off, or cut off the power during operation, there is no problem. Execute RESET command to recover.
However in case the arm collide with something like peripheral system or after exchanging the motor or reduction gear, there are dangers like crashing into something. In case of that calibrate each axis as referring "3.12 Calibration" of maintenance manual.

• LEDs on controller and OPU-300 should be lit up as follows:

	LEDs	on/off status
Controller	E. STOP S. ERR	Light off
	SAFE GUARD	Light on or off
	MODE Light (TEACH/AUTO)	One light is on
OPU-300	RESET	Light on
	MOTOR POWER EMG. STOP ERROR SYSTEM ERROR PAUSE START	Light off
	SAFE GUARD HOME	Light on or off

④ When you turn the power on and the start-up status differs from the above, remove a possible cause of error.

2.3 Setting Data Backup

NOTE

There are important system setting data in the controller. Make sure to back up the original system data.

How to backup various kinds of setting data

- SPEL Editor : Execute MKVER command in command mode.
- SPEL for Windows : Execute [Tools]-[Maintenance] command and click the <MKVER> button on [Maintenance] dialog box.

The various setting data have been stored to the controller's internal file memory as a file by above step. The SETVER command can be used to reset setting data that has been stored to file memory. Refer to the pages in reference manual that describe MKVER and SETVER commands.

In addition, you can backup the files stored in file memory on the PC's floppy disk. For further description, see "Backing up and Restoring a File" in chapter 3 of SPEL Editor manual or "[File] menu" of SPEL for Windows manual.

The equipment, called the programming unit, is used to teach the points and to create programs in order to operate the robot. Generally, a personal computer is used with the programing software,

About initializing of the motion range

The motion range is set by using the RANGE command. Neither power off nor executing VERINIT command changes RANGE motion ranges. When restoring setting value of RANGE into the default value, reset the data as referring the backup data you got by above.

The RANGE is important command to move the manipulator safety. Read "8.3 Changing the motion range" of manipulator manual carefully when using the RANGE command.

2.4 Motor Power On

The motor will not engage by only turning the controller power on. To operate, it is necessary that the motor is engaged. To engage the motor, follow any of the methods below:

(a) From the programming unit.

- SPEL Editor : Execute MOTOR ON command in command mode. (Refer to SPEL Editor manual.)
- SPEL for Windows : Execute [Tools]-[Robot Control Panel] command and click the <MOTOR ON> button on [Robot Control Panel] dialog box. (Refer to SPEL for Windows manual.)

(b) From the operating unit OPU-300. (Refer to OPU-300 manual for detail.)

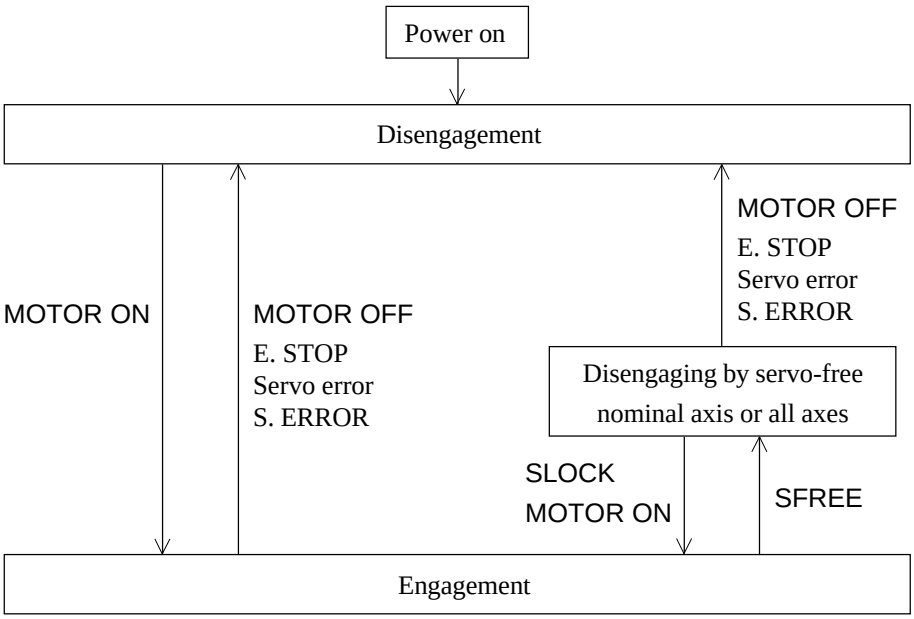
(c) From the teaching pendant TP-320/TP-320J. (Refer to TP-320/TP-320J manual for detail.)

(d) Input the motor power on signal to the REMOTE3. (Refer to controller manual.)

(e) Enter the MOTOR ON command in program.

The motor engagement/disengagement status

The motor engagement/disengagement status will switch as shown below:



2.5 Machine Calibration

Machine calibration must be executed after the motors are turned on, unless the manipulator is BNA or BNA-CL type. Use any of the following methods for machine calibration.

(a) From the programming unit.

- SPEL Editor : Execute MCAL command in command mode. (Refer to SPEL Editor manual.)
- SPEL for Windows : Execute [Tools]-[Robot Control Panel] command and click the <MCAL> button on [Robot Control Panel] dialog box. (Refer to SPEL for Windows manual.)

(b) From the operating unit OPU-300. (Refer to OPU-300 manual for detail.)

(c) From the teaching pendant TP-320/TP-320J. (Refer to TP-320/TP-320J manual for detail.)

(d) Input MCAL to the REMOTE3 as I/O-1. (Refer to controller manual.)

(e) Enter the MCAL command in program.

When you execute the MCAL when the arm is near the limit of motion range, the arm may go beyond the motion range and calibration becomes impossible. If this happens, execute the MOTOR OFF command, or turn off the controller, and manually return the arm to the center of the motion range before starting over. (Refer to "9.1 Calibration" of manipulator manual.)

2.6 Home (Standby) Position Setup

The home position can be set in an optional position. The arm is moved to the position using the HOME command. Set the home position if necessary.

The HOME command moves the robot to a home (standby) position in SPEL III Ver. 3. It calibrated robot in previous version.
Upon shipping, the home position is not defined. If the HOME command is executed before defining it, it will come up as an error 143.

The HOME LED on the OPU-300 and the HOME output to the REMOTE2 will be on while the homing operation is executed.

Defining the home position

The definition of the home position is executed by inputting the pulse value of the position you want as home position using the HOMESET command.

If you want to know the relation between motion range and pulse value, refer to "8. The Motion Range and Robot Coordinate" in manipulator manual.

• SPEL Editor:

a) When you know the pulse value of the position you want as HOME, then input that pulse value in command mode.
<Example> `HOMESET 0,0,0,0`

b) When the pulse value is unknown, you can easily define it using the PLS function. The procedure is as follows:
① Execute SFREE command.
② Move the arms manually to the position you want as HOME.
③ Enter `HOMESET PLS(1),PLS(2),PLS(3),PLS(4)`

• SPEL for Windows:

Execute [Project]-[Robot Parameters] command and set the pulse value of home (standby) position on [HOMESET] panel in [Robot Parameters] dialog box.

Arm moving order in homing

When homing, each axis is activated according to the order of the HORDR command setup.
Upon shipping, the default setup of HORDR is as follows. After the axis #3 is repositioned at HOME, the rest of the axes are repositioned at the same time.

<default value> `HORDR &B0010,&B1101,0,0`

If the repositioning order of the default setup is inconvenient for your application, change the setup values using HORDR.

• SPEL Editor:

Set the values with HORDR command in command mode.

• SPEL for Windows:

Set the value on [HORDR] panel in [Robot Parameters] dialog box of [Project] menu.

2.7 Teaching and Programming

Teaching

In order to drive the robot, it is necessary to teach the robot a target position. This is called "teaching". There are three types of teaching methods but in actual operation, they are combined.

Remote teaching : In remote teaching the manipulator is moved to the desired position using the jog keys of PC or teaching pendant and the position is taught.
The details for jog operation are explained in next chapter 3.

Direct teaching : In direct teaching the manipulator, gets the disengagement of motors using the SFREE command, is manually (directly) moved to the desired position and the position is taught. The details for direct teaching are explained in "Let's try the teaching" of next page.

MDI teaching : In MDI (Manual Data Input) teaching you directly enter the data for a desired position when you know the coordinate values of the target position. Refer to "Pn=Position Specification" of SPEL III reference manual when you want to know how to enter the position data.

In teaching PC (programming unit) or TP-320/TP-320J (teaching pendant) is used.

If you use the teaching pendant TP-320/TP-320J,
Refer to teaching pendant manual.

If you use a PC running SPEL Editor,
Refer to "6. Teach Key Mode" of SPEL Editor manual.

If you use a PC running SPEL for Windows,
Refer to [Jog and Teach] dialog box in [Tools] menu of SPEL for Windows manual.

Example of the teaching method

Let's try the direct teaching method. With SFREE command, the motors are turned off and the manipulator arms can be moved manually. Move the arms to a target position and register the position. Then turn on the motor power with SLOCK command.

• When the SPEL Editor is used:

① Turn on the controller and execute the MOTOR ON command in command mode of SPEL Editor.

```
>MOTOR ON
```

② Execute MCAL in command mode. (In case of BNA and BNA-CL, skip this step.)

```
>MCAL
```

③ Execute SFREE command. All axes will be freed.

```
>SFREE
```

④ Move the manipulator arms to a desired position manually. If the axis #3 has an electromagnetic brake, disengaging the motor will apply the brake to lock the axis #3. To move the axis #3, move the shaft with the brake release button held pressed. Refer to item 5.1 of manipulator manual.

⑤ Registering (teaching) the current position as P1.

```
>P1=P*
```

⑥ Move the manipulator arms manually to next target position.

⑦ Teaching the current position as P2.

```
>P2=P*
```

⑧ You have taught the positions by performing the above steps. Execute SLOCK command to engage motors.

```
>SLOCK
```

⑨ To check for sure, use the JUMP command to move the robot to the position that was taught as P1. Make sure there is no obstacle near the taught target positions and be out of safeguard.

```
>JUMP P1
```

⑩ Move the robot to P2 using the JUMP command.

```
>JUMP P2
```

• When the SPEL for Windows is used:

① Turn on the controller and engage the motors. When you execute the [Tools]-[Robot control Panel] command, the [Robot Control Panel] dialog box is displayed. Select manipulator as the point device. Click the <MOTOR ON> button. A confirmation message appears. Follow the instructions provided by this message.

② Click the <MCAL> button on [Robot Control Panel]. When calibration is complete, click the <Close> button. (In case of BNA and BNA-CL, skip this step.)

③ When you execute the [Tools]-[Jog and Teach] command, the [Jog and Teach] window appears. Disengage the motor of all axes using the [Free Axes] group box.

④ Move the manipulator arms to a desired position manually. If the axis #3 has an electromagnetic brake, disengaging the motor will apply the brake to lock the axis #3. To move the axis #3, move the shaft with the brake release button held pressed. Refer to item 5.1 of manipulator manual.

⑤ Register the current position at P1. Click the <Point Slider Right> button and change the current point to P1. Click the <Teach P1> button.

⑥ Move the manipulator arms manually to next target position.

⑦ Register the current position at P2. Click the <Point Slider Right> button and change the current point to P2. Click the <Teach P2> button.

⑧ You have taught the positions by performing the above steps. Engage the motors of all axes using the [Free Axes] group box.

⑨ To check for sure, use the JUMP command to move the robot to the position that was taught as P1. Make sure there is no obstacle near the taught target positions and be out of safeguard. Register 1 in the [Point #:] text box and click <JUMP P1> button.

⑩ Register 2 in the [Point #:] text box and click <JUMP P2> button.

Click the <Close> button and quit the [Jog and Teach] window.

Example of the programming

Let's make a simple program using the position data that we taught the robot. This is a short program but it enables the robot to go back and forth from point 1 (P1) to point 2 (P2). If you have taught more points then you may use them after line 30 in the same way.
Now input the following:

```
10 FUNCTION MAIN
20 JUMP P1
30 JUMP P2
40 GOTO 20
50 FEND
```

• When the SPEL Editor is used:

① Input the program shown above in command mode. Type [Return] (Enter) key at the end of the line to register.

② To make certain that each line has been inputted, you may list the program by entering the LIST command.

```
>LIST
10 FUNCTION MAIN
20 JUMP P1
30 JUMP P2
40 GOTO 20
50 FEND
```

• When the SPEL for Windows is used:

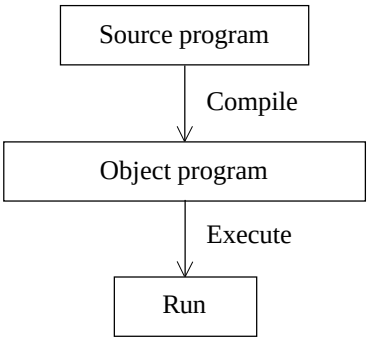
① Create a new project.
Execute the [Project]-[New] command. Input the name of the project in the [New Project Name] dialog box. For example, FIRSTAPP, etc. Click <OK> button.

② Register the program line shown above into the [MAIN.PRG] window.

2.8 Executing a Program

Compiling

The program we just made is called the "source program". The program executed by the controller is called the "object program", it differs from the source program. Therefore, the source program must be changed to the object program. This converting operation is called compiling. When the compiling is finished, the program can be executed from the PC and the operating unit.



- Compiling with SPEL Editor:

  Execute the COM command in command mode.
  >COM
  COMPILE END

- Compiling with SPEL for Windows:

  Execute the [Project]-[Build] command. The [Project Build Status] window opens and the build procedures are displayed. (The program created is transmitted to the robot controller and compiled.)

Execute the program from PC

Let's execute the program.

- Execute the program from SPEL Editor:

  Execute the XQT command in command mode.
  >XQT

  To abort, press the [STOP] key or [CTRL]+[C] key on PC's keyboard.

- Execute the program from SPEL for Windows:

  When you execute the [Run]-[Start] command, the [Run] window is displayed. When you click the <Start maingrp : MAIN> button, the program executed.

  To end execution, click the <Stop all> button on [Run] window.

Execute the program from operating unit

① Set the operation mode of the controller into AUTO by selecting the AUTO position of mode selector switch on OPU-300.

② Close the safeguard.

③ Press the START switch on OPU-300.

④ When stopping the program, make the program temporary halt by pressing the PAUSE switch of OPU-300 and press the RESET switch to stop it.

The LED light on top of each switch shows following condition.

LED	Condition
RESET	Stopping (reset) status of program
START	Executing status of program
PAUSE	Pause status of program

Execute the program from REMOTE3

In case of executing the program from PLC (programmable logic controller) or available other operating unit, use REMOTE3. When using I/O-1 of back of controller as REMOTE3, following steps must be executed.

① Turn on bit 1 of the software switch SS1.

② Set the necessary bit as REMOTE3 from input/output of I/O-1.

* The setting of software switch and setting of REMOTE3 are executed by SPEL Editor or SPEL for Windows. Refer to each manual regarding how to set.

③ Turn off the power of controller once, and turn on.

④ Input a START signal via REMOTE3, execute the program.
When stopping the program, make the program temporary halt by PAUSE signal and input the RESET signal to stop it.

* The pin assignment of REMOTE3 and the timing charts of input signals are described in "7. I/O Remote Set Up (REMOTE3)" of controller manual.

2.9 Selecting Program for Changing Lines and Products

Program must be changed when changing lines or products. In other words, the appropriate executable file (object file and symbol file) must be selected.
The method of selecting program for changing lines and products is described below.

Selecting program from REMOTE input

The appropriate executable file must be in the file memory when selecting and executing the program.

1) Make a executable file in file memory
   Save a source program and point file by using DSAVE command in file memory, and execute COMPILE command to create an object file and symbol file.
   <Example>

```
>DSAVE"01TEST"
>COMPILE"01TEST"
COMPILE END
>
```

   * When first two characters of file name is specified with numeral, the number is the "program number". The program number is used when selecting file with ↑/↓ key of OPU-300 or program number input of REMOTE 3.

2) Transfer the executable file to main memory
   Next, transfer the created executable file to main memory. The method of transferring file using OPU-300 or REMOTE 3 is as follows.

   • If using OPU-300

   The executable files are selected on file selection screen 1 or 2 of OPU-300.
   It is possible to send all executable files which are selected on file selection screen into main memory.
   How to operate the OPU-300 is described in OPU-300 manual.

   • If using REMOTE3

   The executable files are selected by specifying the program numbers.
   There are two ways to specify program number; binary method and up/down method. PRGNO command determines whether to apply the binary method or the up/down count method. The binary method is easier to control but only 15 files (program number 01 to 15) can be selected with this method. If it is necessary to select more than 15 files, please use the up/down method.

   After establish the program number input signal, when START is input, executable files are transferred to main memory and start the program. Refer to "Program execution timing chart" in "7.4 Timing chart" of controller manual.

Utilization of CHAIN command

Except for specifying program number by REMOTE input, CHAIN command is also the method of changing execution program. When creating the program in which changing lines or products is necessary, CHAIN command sometimes makes it effective.

CHAIN command is designed for changing products while executing program. CHAIN command can replace the running program by specified program on file memory.
Refer to “2.1 How to write large-scale programs” in chapter 2 in applied section of this manual regarding how to use the CHAIN command.

2.10 File Handling

- File handling with SPEL Editor:

  The created program and position data with SPEL Editor in "2.7 Teaching and Programming" are stored in "main memory" of controller. The number of program in main memory is only one and the number of position data group in main memory is also only one. Therefore, when creating new program and position data, the program and position data in the main memory must be saved as files into "file memory."

  See the steps in "3.4 Handling (saving and loading) files" of SPEL Editor manual.

  If you use the SPEL Editor, read the SPEL III reference manual regarding following commands related to file handling.

  Commands related to file handling

  DSAVE : Saves main memory source program and position data files in file memory.

  DLOAD : Loads specified files into main memory.

  DIR, FILES : Displays the file information in file memory.

  COPY : Copies file to another location.

  KILL, DEL : Deletes file(s)

- File handling with SPEL for Windows:

  In SPEL for Windows, the files of programs and position data are stored in the PC. Refer to SPEL for Windows manual for details.

CHAPTER 3. JOG OPERATION

In remote teaching the manipulator is driven to the desired position using the jog key of PC and teaching pendant, then the position is taught. The driving of the various axes of the manipulator using the jog keys is called "jog feeding".
In jog feeding you can change the direction in which the manipulator moves and the method by which it moves when the jog key is pressed by setting "coordinates used in jog feeding" and "jog feed operating mode". Select the best setting for the application to ensure efficient teaching.

In case of using TP-320/TP-320J, refer to "4. Teaching" of TP-320 or TP-320J manual which describes the contents of this chapter.

3.1 Coordinate System Used in Jog Feeding

Decide which of the coordinate systems you wish to use to perform jog feeding. The direction that the axes move when the jog keys are pressed depends on which coordinate system is selected.

Coordinate system	Content
BASE	Base coordinate system. Jog feeding is performed in accordance with the manipulator's base coordinates (BASE0). Because the coordinate system is fixed in the manipulator, you can tell at a glance the direction of each of the coordinate axes.
TOOL	Tool coordinate system. Choose a coordinate system for jog feeding that is in line with a tool coordinate system, assuming that the tool is mounted to the end of the manipulator's hand. The direction of coordinate axes changes depending on the rotation of the hand.
JOINT	Joint system. Perform jog feeding for each of the manipulator's joints.

Refer to the below each manuals regarding how to select coordinate system.

- SPEL Editor : Refer to "6. Teach Key Mode" of SPEL Editor manual.
- SPEL for Windows : Refer to "Inputting and Executing Simple Program" in chapter 3 and "[Tools] menu" in chapter 8 of SPEL for Windows manual.

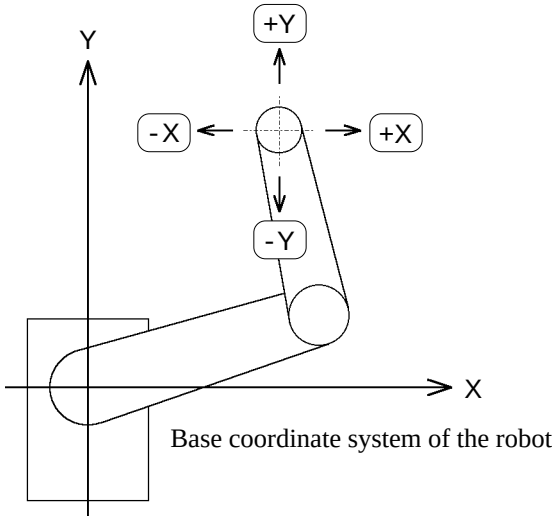
The explanation of each coordinate systems are as follows.

Jog feeding using the "BASE" coordinate system

When the base coordinate system (BASE) is selected, the direction that each axis moves when the jog key is pressed matches the base coordinate system (BASE0) of the robot.
In the BASE coordinate system the manipulator moves in accordance with the coordinate system fixed in the robot, so a fixed relationship is always maintained between the jogging direction of each axis and the orientation of the body of the manipulator. Because this makes it easy to tell the jog direction at a glance, this is the most frequently used coordinate system. Most rectangular coordinate robots can be taught using this coordinate system.

Different types of manipulators have different base coordinate systems. Please see the manual that came with your manipulator for information in its base coordinate system.

In general, the coordinate system is as follows:



Jog feeding using the "BASE" coordinate system

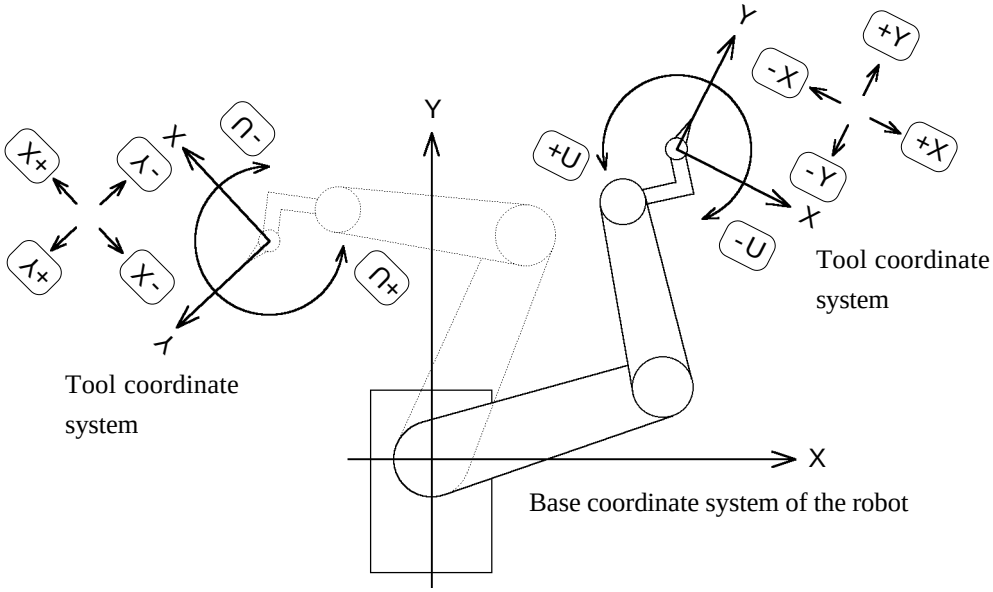
NOTE
Please check the coordinate system if the base coordinate system has been changed by execution of the BASE0 command or LOCAL 0 command.

Jog feeding using the "TOOL" coordinate system

NOTE

When the tool coordinate system has been selected, the direction of movement executed by each jog key is determined by the orientation of the hand. Even if the hand rotates, jog feeding is performed by fixed coordinates for the orientation of the hand. It can, therefore, be used for nozzles and other hands that have directionality.

The origin point of the tool coordinate system can also be set in a location away from the center of the axis #4 (rotating axis) of the manipulator. The origin point of the tool coordinate system is centered on the tool defined by the TLSET command.



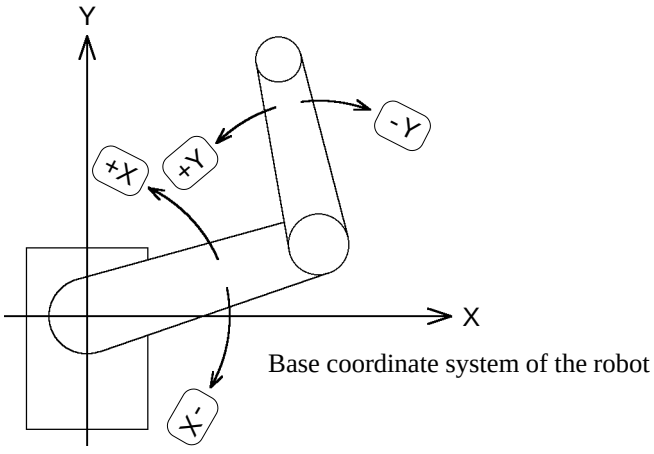
Jog feeding using the "TOOL" coordinate system

NOTE

For details about the tool coordinate systems, please see the items regarding the TOOL command and TLSET command in the SPEL III reference manual.

Jog feeding using the "JOINT" system

When the JOINT system is selected, the jog keys correspond to each joint. This system is good for jog feeding when you want to drive a particular axis of a jointed manipulator.



Jog feeding using the "JOINT" system

3.2 About Jog Movement

The travel distance of the jog movement can be change. We recommend that you set the big value firstly and move the arm to the target position roughly, after that move the arm as you set the small value gradually and get the fine position.

The unit for the travel distance of the jog movement may differ as follows depending on the coordinate system used for jog feeding or the manipulator mechanism.

	axis #1 (X/J1)	axis #2 (Y/J2)	axis #3 (Z/J3)	axis #4 (U/J4)
BASE coordinate system unit	mm	mm	mm	degree
TOOL coordinate system unit	mm	mm	mm	degree
JOINT system unit (direct acting axis)	mm	mm	mm	-
JOINT system unit (rotating axis)	degree	degree	degree	degree

Refer to the below each manuals regarding how to select the travel distance and other details.

- SPEL Editor : Refer to "6. Teach Key Mode" of SPEL Editor manual and SEL, SET command of SPEL III reference manual.
- SPEL for Windows : Refer to "Inputting and Executing Simple Program" in chapter 3 and "[Tools] menu" in chapter 8 of SPEL for Windows manual.

NOTE

In case SPEL Editor and teaching pendant TP-320/TP-320J are use together, make sure to read "4. Teaching" of TP-320/TP-320J manual. Note that jog movement distance which is set by SPEL Editor is changed into the setting value of TP-320/TP-320J when TP-320/TP-320J is used. In case SPEL for Windows and teaching pendant TP-320/TP-320J are use together, refer to chapter 7 of SPEL for Windows manual.

If you continue to hold down a jog key, the arm may move beyond the moveable range, initiating an error stop and disengaging the motor. If this happens take the following steps:

① Manually return the arm to a location within the moveable range.

② Press the RESET key to cancel the error.

③ Execute MOTOR ON. This engage the motor and resets the robot.

ELEMENTARY SECTION

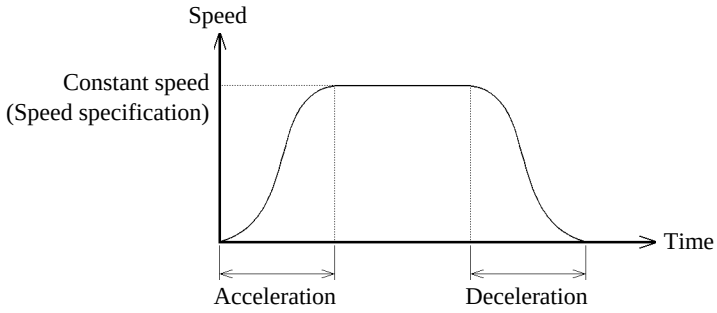
CHAPTER 1. MOTION SPEED

1. 1 Acceleration/Deceleration Speed

When executing the robot motion command, the motion changes as shown below:

Acceleration → Constant speed → Deceleration → Stop

The speed curve will be decided by the speed and acceleration/deceleration setting.



Commands related to the speed are divided into two categories depend on the motion command.

Speed	Accel./Decel.	Corresponding motion commands	
SPEED (%)	ACCEL (%)	JUMP GO PASS	PTP motion
SPEEDS (mm/s)	ACCELS (mm/s^2)	MOVE ARC CVMOVE	CP motion

* The unit of speed and acceleration/deceleration in ( ).

PTP (Point To Point) motion commands will be often used for pick-and-place operations. In this kind of operation, possible fastest motion to the target position is required, so the speed setting by mm/s does not have important meanings. In the case of the horizontal articulated (SCARA) robot, allowable maximum speed and acceleration/deceleration differ according to the moving position in the motion range. Therefore, SPEED and ACCEL values are specified by the percentage while the maximum speed that is allowed mechanically is regarded as 100.

CP (Continuous Path) motion commands will be often used for sealing operations. Sealing operation requires a constant speed motion without concerning the current position and target position relations. Therefore, SPEEDS and ACCELS valued should be specified by the unit of mm/s and mm/s^2.

Regarding the details of SPEED, SPEEDS, ACCEL, and ACCELS command, refer to SPEL III reference manual.

1.2 High-speed Operation in TEACH mode

When the controller is set to TEACH mode for programming or teaching, as a rule the robot is always set to low power mode and cannot operate at high speeds since there is a risk of someone entering the robot's operation area.
The following three constraints are applied for this reason, and all of these constraints must be canceled to enable operation at programmed (faster) speed.

· Safeguard constraint

· POWER (LP) command constraint

· TSPEED command constraint

Safeguard constraint

When the safeguard is open (in order words, when REMOTE1's safeguard switch input is released), the robot stays in low power mode.
As long as the safeguard remains open, the robot operates only at the speed set by either the fixed setting or the TSPEED (TSPEEDS) setting indicated in below, whichever is slower, even when a higher speed has been programmed.

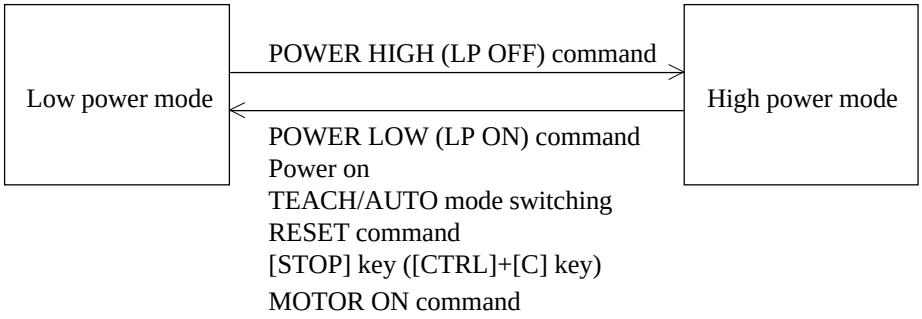
The safeguard must be closed before the robot can operate at the programmed speed.
Refer to "1.4 Mode" in introductory section of this manual and "4. REMOTE1" of controller manual for detail.

POWER (LP) command constraint

When the POWER LOW (LP ON) command is execute, the robot stays in low power mode. The POWER HIGH (LP OFF) command must be executed before the robot can operate at the programmed speed.

The robot's power mode is normally set to low power mode when the controller is turned on or when switching from AUTO mode to TEACH mode. Once in low power mode, even when the safeguard is closed, the speed of the robot is constrained to the fixed speed that is set for the particular robot model.

Since low power mode is the fixed power mode when in TEACH mode, and attempt to enter the POWER HIGH command while in TEACH mode results in an automatic return to low power mode, as illustrated below.



Specifically, a return to low power mode occurs under the following conditions.

- When an error requiring a reset or turning off the power once and on again occurs
- When TEACH/AUTO mode switching
- When the [STOP] key ([CTRL]+[C] key) combination is pressed to stop program execution via the XQT command
- When executing the MOTOR ON command

Although robot operation speed is constrained during low power mode, any speed settings or acceleration/deceleration rate settings made during low power mode are still saved to the controller's internal memory.

TSPEED (TSPEEDS) command constraint

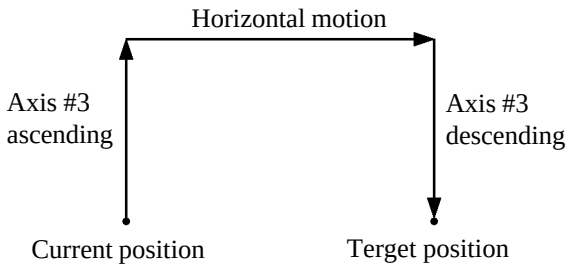
As mentioned earlier, when in TEACH mode it is possible for people to enter the robot's operation area while the robot is operating, so constraints are automatically set on the controller's control (via the safeguard and POWER command) of the robot's operation speed and power mode. These constraints are generally sufficient to ensure safety. However, as a further safety assurance, the robot handler is able to set an absolute maximum operation speed for TEACH mode.

The TSPEED (TSPEEDS) command is used to set an upper limit on robot operation speed set via the SPEED (SPEEDS) command while in TEACH mode. This maximum speed setting is retained in nonvolatile memory when the controller's power is switched off.

The speed set via TSPEED (TSPEEDS) also works as the absolute maximum speed when setting speed via the SPEED (SPEEDS) command.

1.3 Axis #3 Speed/Acceleration Control for JUMP command

The basic movement figure of the JUMP command is shown below:



The speed and acceleration/deceleration is effective to all the motion in the above. However, axis #3 ascending/descending motion speed is especially important for pulling out or inserting application, so you need to specify the most suitable speed. For example, if the robot picks up a device at the current position, and inserts it into the target position, you want to move fast the axis #3 ascending and horizontal motion, however, to move slowly when the axis #3 descends.
In order to satisfy this kind of purpose, axis #3 speed and acceleration/deceleration can be set separately from the horizontal motion speed.

Axis #3 motion speed can be specified with three parameters using the SPEED command.

Format: SPEED [A],[B],[C]

A : Speed specification value
B : Axis #3 upward speed specification value
C : Axis #3 downward speed specification value

Axis #3 acceleration/deceleration speed can be specified with six parameters using the ACCEL command.

Format: ACCEL [A],[B],[C],[D],[E],[F]

A : Acceleration specification value
B : Deceleration specification value
C : Axis #3 ascending acceleration specification value
D : Axis #3 ascending deceleration specification value
E : Axis #3 descending acceleration specification value
F : Axis #3 descending deceleration specification value

<例>
```
1000 FUNCTION MAIN
1010 SPEED 100,80,20
1020 ACCEL 100,100,100,100,80,20
  :
2000 JUMP P1
  :
3000 FEND
```

1.4 Transporting Objects Heavier than the Rated Weight

The robot's speed and rates of acceleration and deceleration varies depending on the load attached to the end of the robot arm.
Therefore, to set an appropriate speed or acceleration/deceleration rates, you should first measure the weight load and make the following setting via the WEIGHT command.
This load is the combined weight of the robot hand and the object it is holding.

Format: WEIGHT [Hand weight],[Arm length]

The arm length is the distance from the rotational center of axis #2 in a horizontally articulated robot to the center of gravity of the robot hand as the transported object combined. If this center of gravity is the center of axis #3, it does not need to be specified. Neither is there a need to specify it if the robot is not a horizontally articulated robot.

When the WEIGHT command is executed, the controller calculates the maximum speed and acceleration/deceleration rates possible for the mechanisms given the specified transportation weight. If a value of 100 has been specified via the SPEED or ACCEL command, this speed may be set if the controller's calculation shows that there is no mechanical hindrance.

NOTE

Regarding how to decide the value of WEIGHT command parameters depending on hand weight, refer to "5. Hand" of manipulator manual. Also, the usage of WEIGHT command is described in SPEL III reference manual.

⚠ CAUTION

Robot transportation of an object that greatly exceeds the rated weight while using the rated-weight setting (default setting), maximum speed setting, and maximum acceleration/deceleration rate settings can not only cause an error due to overloading but may cause an accident. Use the WEIGHT command to ensure appropriate weight settings.

CHAPTER 2. PROGRAMMING

2.1 Basic Constituents of Program

The basic unit of program is called a "line" and a series of lines constitutes a program. These lines are made using "line numbers" and "statements".

```
100 JUMP P7 C0 LIMZ-50
```

Statements

Line number (Integer from 1 to 32767)

Characters in one line: within 79 characters

The basic program from starts with the 1st line as FUNCTION and ends with the last line as FEND.

```
10 FUNCTION MAIN
     ⋮
1000 FEND
```

From FUNCTION to FEND we call "Function". After the FUNCTION, a function name is necessary. In the above example the function name is "MAIN".

SPEL III has 16 multi-tasking capabilities. "Multi-tasking" refers to a type of processing in which multiple tasks are executed simultaneously or in turns. The multi-tasking program is described as a continuation of the Function. The places where various functions are executed are called "tasks" and the controller has 16 tasks (Task number 1 to 16). The number of function can be made up to 69 in a program. The task can be executed, stopped when it is necessary.
See the item "Multi-tasking" in applied section for further description of multi-tasking.

The Function at the very beginning of the program is called the "Main Function" and is always executed as Task 1. The tasks other than the main function must be specified task numbers using the XQT command within the main function.

```
10 FUNCTION Function 1
20 XQT !2,Function 2
30 XQT !5,Function 3
  ⋮
980 FEND
990 '--------------------
1000 FUNCTION Function 2
  ⋮
1980 FEND
1990 '-------------------
2000 FUNCTION Function 3
  ⋮
3000 FEND
```

Main Function

Function name

The following cites the restrictions on the function names, but other than what is mentioned, the function names may be freely named.

Function name

- The usable characters are alphanumerics and underscores (_). There is no distinction between capital and small case letters.
- Within eight characters.
- The first character must be an alphabet other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. Reserved words with following underscore or numerics are also read as reserved words.

Comments

To make a program easier, you may add comments if necessary.
The symbol used to indicate a comment is the " ' " (apostrophe) and whatever characters entered after it are considered comments. During execution of a program, the content entered after the apostrophe is disregarded.
You may use any characters in the comment. However, the number of characters in one line, including the line no., statement, and comment may not exceed 79 characters.

<Example>

```
10 FUNCTION MAIN
20 ' **********************   ┐
30 ' *     HANDLER -1       *  │
40 ' * PROGRAMMED BY SATO   *  ├ Comment line
50 ' **********************   │
60 '                           │
70 ' GOSUB 1000                ┘ ← GOSUB 1000 will not be executed.
80 SPEED 100 ;ACCEL 50,100
90 JUMP P10 ;ON 1 ;WAIT 0.2  'Picks up device at Feeder 1 ← Comment
```

Multi-statement

When the " ; " (semicolon) is used, more than one command may be described on one line. This is called multi-statement.
With multi-statement the program becomes easier to read and the size of the program is also decreased. And the execution time is shortened. However, if unnecessary multi-statements are entered, the program can become clustered and hard to understand.

<Example>

```
100 JUMP P7 ; ON 1 ; WAIT 0.2
110 JUMP P17 ; OFF 1 ; WAIT 0.2
```

Labels

When you change the program execution order by using GOTO, GOSUB, or IF...THEN...ELSE, it will run according to the specified line number. However, the line number is just number and doesn't have other meanings. If you can use a word to indicate the movement, it would be much more convenient for the programmer as well as the user.
In SPEL III you may use a "label" (specify after statement line number) instead of using only a statement line number for the GOTO, GOSUB, or IF...THEN...ELSE statements.

<Example>

```
10 FUNCTION MAIN
20 LOOP:          ← Label
30    JUMP P1
40    JUMP P2
50 GOTO LOOP      ← Use label to branch program execution.
60 FEND
```

In the example above, line 20 is labelled with "LOOP". When labelling, a place a " : " (colon) after the label and that indicates a label.
Other than the restrictions given below, the labels may be freely maned.

Labels

- The usable characters are alphanumerics and underscores (_) except for colon (:). There is no distinction between capital and small case letters.
- Within eight characters.
- The first character must be an alphabet other than "P".

When you revise the program, there is a function which will correct the line numbers (renumber), so the specified line numbers by GOTO, GOSUB, or IF...THEN...ELSE will be automatically changed. It is easier to understand the program if the labels are used in such case.

2.2 Constants

Constants are actual values SPEL III uses during execution. There are two types of constants: string (or character) constants, and numeric constants.

Numeric constants

Numeric constants are integers and real numbers. It is necessary to place a minus sign (-) for negative numbers, but for positive numbers a plus sign (+) may be omitted.

- Integer constants
  Integer constants do not have decimal points. The following range expresses the boundaries of the integers.
  -32768 to 32767

  There are following description methods for integers.

  ① Decimal format
  The value is expressed in decimals (0 to 9).

  ② Hexadecimal format
  The value is expressed in hexadecimal (0 to F), and it is described with a prefix of "&H". The corresponding values of hexadecimal numbers and decimal numbers are shown in the following table.

Hexadecimal	0 to 9	A	B	C	D	E	F
Decimal	0 to 9	10	11	12	13	14	15

  <Example>
  &H25 : It is 37($2\times16^1+5$) in decimal format.
  &H3F : It is 63($3\times16^1+15$) in decimal format.

  ③ Binary format
  The value is expressed in binary numbers (0 and 1), and it is described with a prefix of "&B".

  <Example>
  &B1111 : It is 15($2^3+2^2+2^1+2^0$) in decimal format.
  &B101 : It is 5(2^2+2^0) in decimal format.

- Real constants
  The values which are expressed with decimal points, which are beyond integral boundaries, and which are represented in exponential form are called real numbers.

Character constants

Character constants are enclosed in quotation marks (").
You can use numerals as character constants by enclosing the numerals in quotation marks. If the character string has zero length, entering only quotation marks ("") will set an empty character constant (also called a "null string").

<Example>
```
>PRINT "HELLO"
HELLO
>PRINT HELLO
123                ← Indicates a value to replace the variable HELLO.
```

2.3 Variables

Variables are used as temporary substitutes for data. There are various types of variables, so before using a variable, a "type statement" must be entered. The data substituted to variable must be same with variable type.

The variable values are retained as long as the object area is not overwritten (they are retained when exiting from a program or when shutting off the power). However, they are deleted when the COM command or another executable file is executed. To save variable values even in these cases, use backup variables, as described below.

Specifiable number of variables

The specifiable number of variables can basically be determined via the following equation.

Specifiable number of variables = 448 - (number of functions)

This is because the variable name and function name are handled in the same area (symbol table). For example, in a single task program, you are able to specify up to 447 variables. However, the number of specifiable array variables differs depending on their size (when large arrays are declared, fewer variables can be used).
For variables other than array variables (such as numerical variables or character string variables), the variable size does not affect the number of specifiable variables.

However, the number of these other variables that can be specified is constrained by other factors. The area to which variable values are saved is also allocated as the object area for storing executable programs. Accordingly, when a particularly large program is being used, the object area available for storing variables may be used up by the program. In such cases, an error occurs when compiling. To fix this problem, you must either reduce the size of the program or reduce the number of variables being used.

Numeric variables

① Types of numeric variables

The types of numeric variables are given in the following table:

	Type declaration	Bytes	Available value/Valid digit
Integer type	BYTE	1	-128 to 127
	INTEGER	2	-32768 to 32767
	LONG	4	-2147483648 to 2147483647 *
Real number type	REAL	4	7 digits
	DOUBLE	8	14 digits

When declaring a variable, type declaration comes first and variable name follows. If several same type variables are declared, use a " , " (comma) and describe several variable names.

The type declaration must be in the beginning of the line. When declaring a different type of variable, create another line. If the variable type is not declared, it will be treated as REAL.

Format: [type declaration] [variable name]{,[variable name]}n

* On PRINT command the available value of LONG type data is restricted as follows.
  -9999999 to 9999999
  The displaying value is restricted, but internal value is calculated within above values and it is valid data.

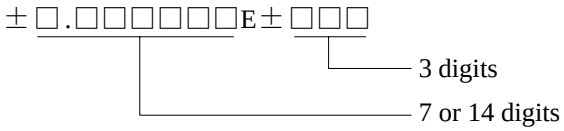
② Variable names

The following cites the restrictions on the variable names. Other than these restriction, the variable names may be freely named.

Variable names

- The usable characters are alphanumerics and underscores (_). There is no distinction between capital and small case letters.
- Within eight characters.
- The first character must be an alphabet other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. Reserved words with following underscore or numerics are also read as reserved words.

③ In the case of real type variable, the value of many digits will be expressed as follows : (normalized data expression)



String variables

String variables are declared using STRING statement.

Function: STRING [string variable name]{,[string variable name]}n

String variable names are written with a " $ " (dollar sign) at the last character to separate from the numeric variable. Again, comply with the following rules.

String variable names

- The usable characters are alphanumerics and underscores (_). There is no distinction between capital and small case letters.
- Within eight characters (including the $).
- The first character must be an alphabet other than "P".
- Reserved words (i.e. command, statement, and function) cannot be used. Reserved words with following underscore or numerics are also read as reserved words.

Array variables

Array variable is a group or table of values that are referred to with one name. Either numeric variables and string variables may be used as array variables. When defining an array variable, declare the name and its size enclosed with ( ). The size can be defined up to 3 dimension.
If the variable type is same, more than one variable name may be defined using the " , " (comma).

Format: [type declaration] [variable name](a,b,c)

a,b,c: Array size of each dimension
Integer from 0 to 254
a×b×c≦32767

<Example>

```
INTEGER A(10)        One dimensional array
BYTE B(8,5)          Two-dimensional array
LONG C(10,10,5)      Three-dimensional array
```

A variable table of array variable B(8,5) is created as below. The numbers in the ( ) are called subscriptions. It is necessary to specify each subscription when using each individual variable.

B(0,0)	B(1,0)	B(2,0)	B(3,0)	B(4,0)	B(5,0)	B(6,0)	B(7,0)	B(8,0)
B(0,1)	B(1,1)	B(2,1)	B(3,1)	B(4,1)	B(5,1)	B(6,1)	B(7,1)	B(8,1)
B(0,2)	B(1,2)	B(2,2)	B(3,2)	B(4,2)	B(5,2)	B(6,2)	B(7,2)	B(8,2)
B(0,3)	B(1,3)	B(2,3)	B(3,3)	B(4,3)	B(5,3)	B(6,3)	B(7,3)	B(8,3)
B(0,4)	B(1,4)	B(2,4)	B(3,4)	B(4,4)	B(5,4)	B(6,4)	B(7,4)	B(8,4)
B(0,5)	B(1,5)	B(2,5)	B(3,5)	B(4,5)	B(5,5)	B(6,5)	B(7,5)	B(8,5)

Listing of variable names

When declaring a variable name, you may need to know the variable names that are already used. Variable names and types will be listed (not including backup variable names) by executing VARIABLE command.

Format: VARIABLE{-A}

When a -A is added, the function name will also be listed. This function will only show the variable names registered on the symbol table, so the source program must be compiled first.

Backup variables

Usually when the program has been compiled, a symbol table is created and when that program is run, value of the variable is stored in the table. However, if that program is compiled again or if a different program is run, then it will be erased. And when the power is turned off, there is no guarantee that it can be restored.

When you want to store value of a variable, use SYS statement to register the variable as a backup variable before using. As a backup variable, it will not be registered in the symbol table but in the main memory backup variable area. This memory area is a battery backup and is not influenced by compiling or executing program.

Backup variable is useful to continue the operation which is stopped the day before, or to use variables commonly in different programs.

① Number of specifiable backup variables

The default value for the number of specifiable backup variables is 10. You can use the LIBSIZE command to set a maximum value of up to 1,000 backup variables. However, setting a high maximum value takes up space in the object area and proportionately reduces the space available for storing programs, so that large programs may no longer be executable. Therefore, it is best to increase the backup variable area as little as possible.
For further details, see the SPEL III reference manual.

② Registration program for backup variables

The symbol table, in which the normal variables are registered, are belonged to the object program. Contrarily, the backup variable registration table is separated from the individual programs. So generally, the backup variable registration programs are created separately from the other programs.

<Example>

```
10 FUNCTION B_UP
20 'SYS BYTE HELLO ;HELLO=0     ' Already registered backup variable
30 SYS INTEGER V(50)
40 SYS STRING ERR_M$(10)
50 FEND
```

If an attempt is made to register a variable name which has already been registered, an error (double definition of variable) will be issued. Use a comment mark similar line 20 if the variable has been already registered.

③ Registration of backup variables

With a general compiler, the registration of variables is completed by compiling the program, but with SPEL III the program must be run to register.
When using the variable, it is necessary to have the variable name and the memory to store its value. After being compiled, the variable name only is registered but the memory to store the value is not secured yet. Upon running the program, that memory is secured.
If a variable is attempted to use after only compiling the program, an error will occur. Be sure to run the program before using backup variables.

<Example>

```
10 FUNCTION V_EX
20 SYS INTEGER A ;A=1
30 SYS INTEGER B ;B=10      ' Register A and B as backup variables
40 PRINT A,B
50 FEND
```

Compile and execute this program.

④ Listing of backup variable names

All of the backup variable names and types which are being registered will be listed by executing LIBRARY command.

Format: LIBRARY

⑤ Deleting backup variables

To delete all of the backup variables which are being registered, execute CLRLIB command.

Format: CLRLIB

It is impossible to delete some specified backup variables. If some backup variables are to be deleted, delete everything once and reregister the necessary backup variable.

Caution on using variables

① Defining a variables (Type declaration)

· Generally, the description of the variable definition is done at the beginning of the program. If a variable is used before registration, it will be registered as a real variable (REAL) and when the variable definition comes up, it will cause an error (double definition of a variable).
Also if a variable is not registered beforehand, the variable may not be able to be used. So even if it is a real type variable, it should be declared.

· When you want to check if there is an undefined variable being used in a program, compile it by adding "-V" to COM command. If there is an undefined variable, error 2 will occur.

```
COM-V
```

· Type declaration statement must be at the beginning of the line. When a different type of variable is declared, make another line. The initial value must be specified by following the type declaration.

<Example>

```
50 BYTE S_1,S_2 ;S_1=0 ;S_2=0 ;INTEGER E_1 ;E_1=1  ' It is not allowed.
  ↓
50 BYTE S_1,S_2 ;S_1=0 ;S_2=0
60 INTEGER E_1 ;E_1=1
```

② Cases that the variable must be defined beforehand

· When a variable is used for the parameter of position data.

· When a variable is used in a parallel processing.

<Example>

```
50  REAL OFSET ;OFSET=1.573
60  INTEGER HAND_UD ;HAND_UD=1
 :
100 P0=P1+XOFSET
 :
200 JUMP P2 !D30;ON HAND_UD!
```

Define the variable OFSET, HAND_UD beforehand.

③ Cases that the variable cannot be used

· For the local coordinate number of position data.

· For the RS-232C port number and task number.

<Incorrect example>

```
P0=P1/LNO          HALT !TK
PRINT #RS,5        RESET #RAIOC
```

④ Case that an array string variable cannot be used

· For file name of CURVE, CVMOV commands.

2.4 Operations

Arithmetic operations

The arithmetic is used for operation of numeric values. Arithmetic operators are as follows:

Operator	Operation	Sample expression	Arithmetic sign
+	Addition	A+B	+
-	Subtraction	A-B	-
*	Multiplication	A*B	×
/	Division	A/B	÷
MOD	Modulo arithmetic	A MOD B	

(The MOD can be used only for an integer.)

Logical operators

The logical operators can be used for the integer operation and a result is also returned as an integer. The numeric value is expressed in binary digit inside the controller. The operation is performed on these sequences. That is, each bit of the result is determined by the corresponding bits in the two operands. Each operation are listed below by using the numeric value A and B.

① AND (Conjunction)
Format: A AND B

Bit of A	Bit of B	Result
0	0	0
0	1	0
1	0	0
1	1	1

② OR (Disjunction)
Format: A OR B

Bit of A	Bit of B	Result
0	0	0
0	1	1
1	0	1
1	1	1

③ XOR (Exclusive or)
Format: A XOR B

Bit of A	Bit of B	Result
0	0	0
0	1	1
1	0	1
1	1	0

④ NOT (Logical complement)
Format: NOT A

Bit of A	Result
0	1
1	0

⑤ Example: In case of A=1, B=0, C=1
A AND B OR C = 1
A AND B XOR C = 1
NOT A) AND C = 0

Relational operators

Operator	Sample expressions	Content
=	A=B	A and B are equal.
< > or ><	A<>B	A and B are not equal.
>	A>B	A is greater than B.
<	A<B	A is less than B.
>= or =>	A>=B	A is greater than or equal to B.
<= or =<	A<=B	A is less than or equal to B.

String operators

Operator	Content
+	Concatenation of the string.
=	True when all the characters of the string are the same.
< >	True when there is at least one character is different.

<Example>

```
>PRINT "HE"+"LLO"
HELLO
```

Order of operators precedence

There is the order of precedence of operators. When more than one operators are used, the operation is carried out from the higher order of precedence. However, operations within parentheses are performed first. Operations at the same level are performed in left-to-right order.

The order of precedence of operators are as follows:

① ( )

② *, /, MOD

③ +, -

④ AND, OR, XOR, NOT

⑤ =, >, <, >=, <=, <>

Operations are performed according to the above order, but the use of parentheses will make the program easier to read.

Integer operation and real number operation

SPEL III distinguishes integer operation from real number operation in order to make operation faster. According to the type of operand, operation method and result will differ as follows.

Operand type	Result
All integers	Integer
All real numbers	Real number
Integers and real numbers	

In the operations above, note the following points.

① For integer division, the result will be rounded to an integer.

② When the result of integer operation exceeds the integer range (-32768 to 32767), it will come up as an error.

③ When a real number is set to an integer variable, the number of the tenth's place is round.

Ex.	Integer operation	Real number operation
①	>PRINT 3/2 1 >PRINT TAN(1/2) 0	>PRINT 3.0/2 1.5 >PRINT TAN(1.0/2) .5463025
②	>PRINT 1000*1000 !!Error 21	>PRINT 1000.0*1000 1000000
③	20 INTEGER A 30 A=TAN(1.0/2) >PRINT A 1	

Return value from the Function

The types of the value returned from the Function as follows.

Type	Function name			
Real number	SIN( ) COS( ) TAN( )	ATAN( ) ATAN2( )	CX( ) CY( ) CZ( ) CU( )	AGL( ) SQA( ) TMR( ) VAL( )
Integer (2 byte)	SW( ) IN( ) INBCD( ) SW($) IN($)	OPORT( ) ZEROFLG(0) NOT( ) LSHIFT( ) RSHIFT( )	MYTASK(0) LOF( ) DSW( ) STAT( )	CTR( ) TIME( ) JS(0) LEN( ) ASC( )
Integer (4 byte)	PLS( )			
Argument	SGN( )	INT( )	ABS( )	

Case where operation and function can be used

Basically it is impossible to use operation and function in the parameter portion of each command. In such case, set the result of operation or function to a variable, and use the variable name for the parameter.

But the following condition command can operate the logical operation.

IF...THEN...ELSE, SELECT...SEND, WHILE...WEND, SENSE, TILL, WAIT, PRINT

<Example>

```
P0=P1:ZCZ(P0)   →   A=CZ(P0);P0=P1:ZA
```

2.5 Program Control Statements

SPEL III has following program control statements. Refer to SPEL III reference manual for details.

FOR...NEXT	(Executes a series of statements a specified number of times)
GOTO	(Branches unconditionally to a desired unconditional statement)
IF...THEN...ELSE	(Executes instructions based on a specified condition)
GOSUB...RETURN	(Branches to, executes, and returns from subroutine)
CALL	(Calls a subroutine)
SELECT...CASE...SEND	(Specifies branching formula and corresponding branch instruction sequence)
WHILE...WEND	(Executes a series of statements while the specified condition is satisfied)
TRAP	(Defines an interrupt process)

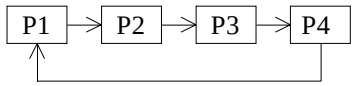
FOR...NEXT

Executes a series of statements from FOR to NEXT for a specified number of times.

GOTO

GOTO branches unconditionally to the specified line number or label.

<Example>

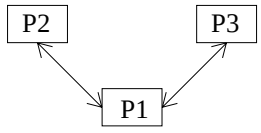



```
10 FUNCTION MAIN
20 LOOP:            ' Labels line 20 as "LOOP"
30   FOR I=1 TO 4
40     JUMP PI
50   NEXT I
60 GOTO LOOP        ' Branch to line 20 labeled LOOP
70 FEND
```

IF [conditional expression] THEN [statement 1] ELSE [statement 2]

When [conditional expression] is true, [statement 1] is executed, and it is not true, [statement 2] is executed.

<Example>



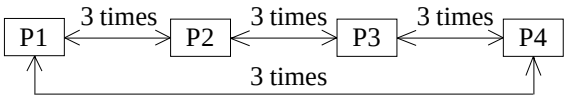
Repeats the jump motion between P1 and P2, and between P1 and P3 in turn.

```
10 FUNCTION MAIN
20 I=1
30 IF I=1 THEN JUMP P2 ELSE JUMP P3
40 JUMP P1
50 I=0-I
60 GOTO 30
70 FEND
```

GOSUB...RETURN

When GOSUB command is found, it branches to specified line number or label.
When RETURN command is executed, it directs program execution to return to the line following GOSUB.

<Example>

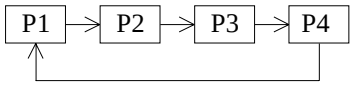


Repeat the motion three times each between P1 and P2, P2 and P3, P3 and P4, P4 and P1. Repeated motion between two positions is made as subroutine.

```
10 FUNCTION MAIN
20   I=1;J=2
30     GOSUB I_J
40   I=2;J=3
50     GOSUB I_J
60   I=3;J=4
70     GOSUB I_J
80   I=4;J=1
90     GOSUB I_J
100   END
1000   I_J:       ' Subroutine label
1010     FOR LOOP=1 TO 3
1020       JUMP PI;JUMP PJ
1030     NEXT
1040   RETURN
1050 FEND
```

GOTO, FOR...NEXT, GOSUB...RETURN, IF...THEN...ELSE

<Example>



Motion speed is increased each time by moving to the next position. If speed becomes SPEED 100, it turns to SPEED 10. Axis #3 moves up and down three times at each position.

```
10 FUNCTION MAIN
20  SPEED 10;A=10
30  FOR I=1 TO 4
40    JUMP PI
50    GOSUB H_L
60    A=A+10
70    IF A>100 THEN A=10
80    SPEED A
90  NEXT I
100  GOTO 30
200 '
210  H_L:
220   FOR J=1 TO 3
230     JUMP PI
240   NEXT J
250  RETURN
300 '
310 FEND
```

CALL

CALL calls a function (defined in FUNCTION...FEND) as a subroutine.

<Example>

```
100 FUNCTION MAIN
110   OFF $0          ' Memory I/O for exclusive control
120   XQT !2 SUB
     :
200   CALL ERROR
     :
300 FEND
305 '
310 FUNCTION SUB
     :
350   CALL ERROR
     :
400 FEND
405 '
410 FUNCTION ERROR
420   ON $0; IF ZEROFLG(0)=1 THEN WAIT SW($0)=0; GOTO 420
     :
490   OFF $0
500 FEND
```

* ZEROFLG(0)
  It returns value of memory I/O previous to it last being switched on or off.

SELECT...CASE...SEND

```
SELECT [formula]
   CASE [item 1] ; [statement 1]
   CASE [item 2] ; [statement 2]
SEND
```

If any one CASE item is equivalent to SELECT formula result, that CASE item statement is executed.

<Example>

```
110 FUNCTION MAIN
120 INTEGER I
130 FOR I=0 TO 10
140   SELECT I
150     CASE 0; OFF1; ON2; JUMP P1
160     CASE 3; ON1; OFF2
170             JUMP P2; MOVE P3; ON3
180     CASE 7; ON4
190     DEFAULT; ON7
200   SEND
210 NEXT
220 FEND
```

WHILE [condition] ...WEND

If WHILE condition is true, executes statements between WHILE and WEND, then once again checks WHILE condition.

<Example>

```
110 I=1
120 WHILE I<60
     ⋮
300 I=I+2
310 WEND
```

TRAP

When input condition is satisfied, executes interrupt process which is specified by GOTO, GOSUB, or CALL command.

<Example 1>: Error process defined by customers

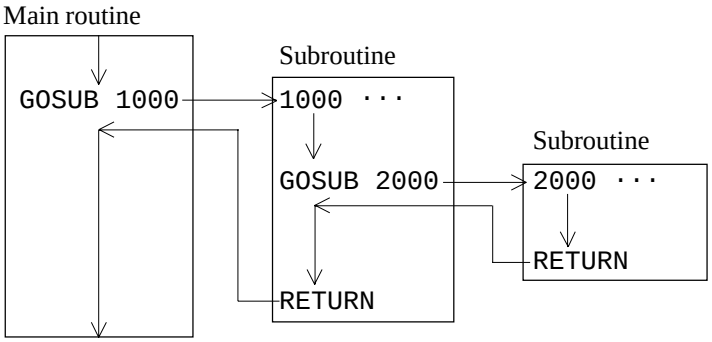
```
100 FUNCTION MAIN
110 TRAP 1 SW(0)=1 GOTO ERROR        ' Defines TRAP
 ⋮
500 ERROR:
510 ON 3                             ' Signal tower lights
520 PRINT #20,"Error is issued."
530 FEND
```

<Example 2>: Usage like multi-tasking

```
100 FUNCTION MAIN
110 TRAP ERROR CALL MSGOUT
 ⋮
500 FEND
700 FUNCTION MSGOUT
710 PRINT #20,ERRMSG$(ERR(0))
720 FEND
```

Nesting

A function called nesting can be used for some statements. It is explained using GOSUB as an example in this item.
GOSUB branches the program control to the specified line number (or the labelled line), executes the subroutine, and returns to the next command after GOSUB using RETURN.
By using another GOSUB...RETURN within the subroutine, you can execute another separate subroutine. This is what is called nesting.



- Subsets of nesting

  The subsets of nesting is described as "steps" and the above example has "2 steps". There are restrictions as to the number of subsets but it differs according to the statement. This manual will describe it as follows.

  Nesting : Up to 10

- Below are the commands which allow nesting.

  #include
  CALL
  FOR...NEXT
  GOSUB...RETURN
  SELECT...SEND
  WHILE...WEND

2.6 Pseudo Command

The pseudo command is used to make program editing and compiling easier in the first step of compiling. Using the pseudo command makes it possible to replace identifiers in the program, install other files, or compile a part of programs.
You must use the "#" as the first letter of pseudo command and use small case letters.
SPEL III provide the following pseudo commands:

#define
#include
#ifdef
#ifndef
#endif

The pseudo command must be described right after the line number. Also, after the pseudo command, there must be at least one space between the following letters.

The position of the pseudo command in the program may be placed anywhere. Even when pseudo command is inserted in the middle of the program, it will be valid for the whole program.

```
1000 FUNCTION MAIN
1010 #define LED1 8
1020 #define LED2 9
1030 #define LED3 10
     :
2000 ON LED1; WAIT 1; OFF LED1
2010 ON LED2; ON LED3
     :
3000 FEND
```

The usage and examples of pseudo commands are described in SPEL III reference manual.

CHAPTER 3. FILES

Files are collections of data. Programs that have been created or teaching-related position data are all handled as files.
The controller includes a main memory area as the work area for performing the creation and execution of programs, plus a file memory area for storing files.
This chapter describes how files are handled between these two memory areas.

3.1 Main Memory and File Memory

The sizes of the main memory's source program area, position data area, and backup variable area can by changed via commands that set area sizes.

The default sizes of main memory and file memory are listed below.

Main memory	No. 1	No. 2	No. 3	Command to set area size
Source program area	64 KB	64 KB	128 KB	PRGSIZE
Position data area	200 points, 5.5 KB			PNTSIZE
Backup variable area	10 variables, 0.5 KB			LIBSIZE
Symbol area	7 KB			
Object area	97 KB	97 KB	233 KB	Can be increased or reduced via above commands

File memory	100 KB	1100 KB	900 KB

No. 1: Standard setting

No. 2: With additional RAM, and dip switch SD2-1 set to ON (on controller's MPU board)

No. 3: With additional RAM, and dip switches SD2-1 and SD2-2 set to ON (on controller's MPU board)

Memory area in the main memory

The main memory is an area which operates the movement of the robot and is divided into the following five areas.

① Source program area
The input program is called the source program and is memorized here.
To execute the program, compile it to an "object program".

② Position data area
The position data created by teaching has been memorized here.
The existing position data is used as the necessary position data for program execution.

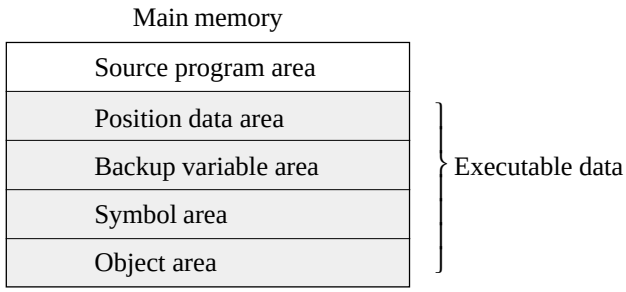
③ Backup variable area
This is the backup variable registration table area.

④ Symbol area
This is the registration table area for function names and variable names. When the source program in the main memory is compiled, it is automatically registered here.

⑤ Object area
This is the area where the actual executable object program is kept.

Program execution area

When running a program, data is used from each area of the main memory excluding the source program area. This data must be combined to conform with the robot operation. After file handling, the files in the main memory may not be related each other. Be sure to confirm that the data is related each other before executing a program.



File memory

The file memory is a memory that stores each type of file. The number of files which can be stored in the file memory is as follows.

File memory	Capacity	Number of file
No. 1	100 KB	64
No. 2	1.1 MB	192
No. 3	900 KB	192

No. 1: Standard setting

No. 2: With additional RAM, and dip switch SD2-1 set to ON (on controller's MPU board)

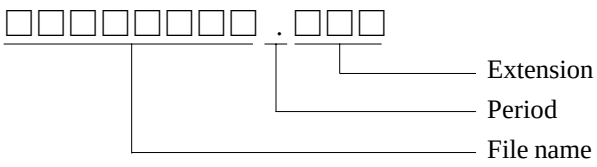
No. 3: With additional RAM, and dip switches SD2-1 and SD2-2 set to ON (on controller's MPU board)

3.2 File names

In each file, a file name must be given. When handling a file, the file name must be specified.

The constituents of a file name

The file name is comprised of file name, extension and " . " (period).



The file name and extension is separated by the period.
Even if the file name is the same, as long as the extension differ, it is considered a different file.

How to make a file name

When saving a source program and position data which was made in the main memory, a file name must be put.
Make file name according to the following rules.

File name

① Within eight characters.

② Usable characters:
- Alphabet (A to Z, a to z). There is no distinction between capital and small case letters.
- Numbers (0 to 9)
- Symbols (! # $ % & ( ) - ^ @ ~ { } _)

Extension

The extension indicates the file type.
Generally, the user doesn't have to put the extension but the controller automatically does it.
If the file name is the same but the extension is different, it is treated as a different file.

The file type of the following extensions have been decided beforehand, the controller automatically puts it and recognize its type.

Extension	File	File type
PRG	Source program file	Created program
OBJ	Object file	Executable intermediate code program
SYM	Symbol file	Variable and function name registration file
PNT	Position data file	Position data file
CRV	Free curve file	Free curve data using CURVE command
BAT	Batch file	File that registers commands to be executed continuously
SYS	System environment setting file	File that establishes the system constituents

When editing the text file with the programming unit (PC) or when using WOPEN and VSAVE command, a user specifies an extension. In those cases, the above-mentioned extensions should not be used for any other purposes.
Make extension according to the following rules.

Extension

① Within three characters.

② Usable characters:
- Alphabet (A to Z, a to z). There is no distinction between capital and small case letters.
- Numbers (0 to 9)
- Symbols (! # $ % & ( ) - ^ @ ~ { } _)

③ File names must be always named but extensions may be omitted. When omitting an extension, omit the period also.

Special file names

The files with the file names listed below are automatically executed when the power is turned on. Do not use those file names other than this purpose.

AUTO.BAT

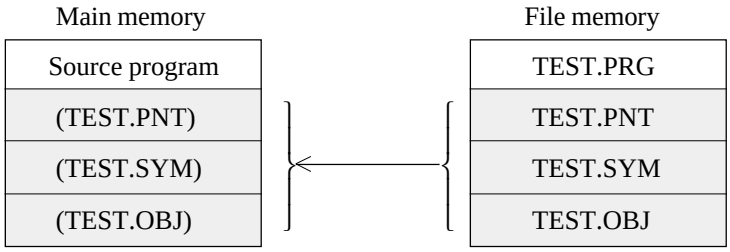
IPL.OBJ

CNFG.SYS

3.3 Files Loaded when Execution

When running a program with specifying the file, the following three files will be loaded from the main memory into the file memory. However, if even one of those programs does not exist, it will cause an error.

Note that the source program will not be loaded.



Once the necessary files for execution are loaded into the main memory, the program can be started by just inputting XQT command (do not specify the file name) or pressing the start switch on the operating unit without selecting the file.

CHAPTER 4. DIRECTORY

4.1 Directory

When a file is saved in file memory, in addition to the file name and its extension, the file size and the date created will be saved in the "directory".

To manage the files, the directory sorts out and registers the files into various groups.
Note that the directory is a place to store files and file and directory are different.

There are two kinds of directory, which are "root directory" and "sub directory".

Root directory

The basic directory, which is automatically processed in the file memory when the file memory is formatted, is called "root directory". All of the files and directories are made in this root directory.

The root directory is expressed by back slash "\", and this symbol cannot be changed.
In addition, it is impossible to delete the root directory.

Sub directory

If many types and a large number of files are registered in the root directory, file management becomes inconvenient and might induce incorrect file use. The capacity of the root directory is previously decided, there is a limit of the number of files that can be registered.
In order to manage files as a group of files, directories can be created or deleted by users according to their necessities. These directories are called "sub directories".

The number of files that can be registered in the sub directory, depend on the size of file memory.

Creating the sub directory

The sub directory is created with a name, in the same way as a file. The sub directory is created using MKDIR or MD (make directory) command.
The characters and the number of characters which can be used for directory name is as same as file names.

Directory name

① Within eight characters.

② Usable characters:
- Alphabet (A to Z, a to z). There is no distinction between capital and small case letters.
- Numbers (0 to 9)
- Symbols (! # $ % & ( ) - ^ @ ~ { } _)

<Example> When creating a sub directory called DATA in the root directory

```
>MKDIR DATA
```

When creating a sub directory called TEXT within a DATA directory in the root directory (The DATA directory is already in existence):

```
>MKDIR \DATA\TEXT
```

You can do the same work by changing the current directory from root directory to DATA directory.

```
>CHDIR DATA          'Changing directory
>MD TEXT
```

It is impossible to create more than two directories at one time. Create one at a time.

Deleting a sub directory

Deleting a sub directory is done using RMDIR or RD(remove directory) command. It is impossible to delete the directories which include files or sub directories.

<Example> When deleting the DATA directory in the root directory, make sure that the contents of the DATA directory are empty and execute the following command.

```
>RD DATA
```

<Example> When deleting the TEXT directory within the DATA directory in the root directory, confirm that the TEXT directory is empty and execute the following command.

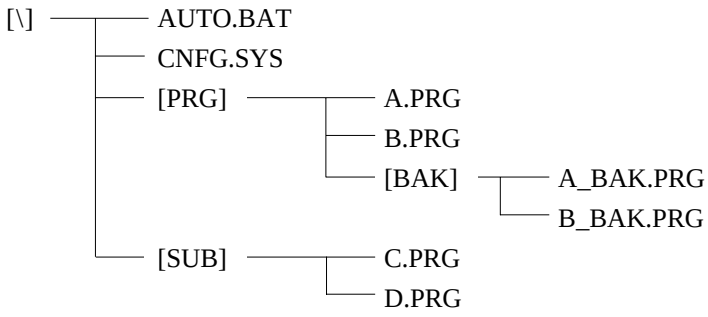
```
>RD \DATA\TEXT
```

You can do the same work by changing the current directory from root directory to DATA directory.

```
>CD DATA          'Changing directory
>RD TEXT
```

Tree-structured directories

When creating several sub directories, they are made in hierarchy. It is called the directory hierarchy. When the directory is laid-out, it looks like the branches of a tree, so can be also called tree-structured directories.



[ ] represents directory name in the figure above.

Parent directory, child directory

In directory hierarchy the directory which is one level higher than currently existing directory is called parent directory, and which is one level lower is called child directory.

In the directory hierarchy shown above, root directory is regarded as the parent directory of the TEXT directory, and the DATA directory is regarded as the child directory.
This relationship is relative. The TEXT directory is considered the parent directory of DATA, but is also considered the child directory of the root directory.

One period [.] indicates currently existing directory, and two periods [. .] indicates its parent directory. They are used when typing pass name or when file names are displayed by DIR command.

Current directory

The current directory is the presently selected directory.
If a directory is not specified, SPEL III starts searching from the current directory. When starting SPEL III, the root directory of the file memory is automatically selected as current directory.

When changing the current directory, use CHDIR or CD (change directory) command.

When turning the power on, the root directory will always be the current directory.

Specifying path

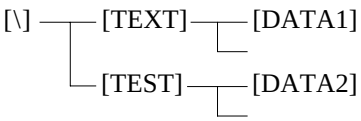
When changing a directory or specifying a directory by a parameter of command, proper operation will not take place by only specifying the directory. The specified directory must be expressed based on the root and current directories according to the directory hierarchy. The directories which are arranged and separated by the back slash " \ " is called "path name".

There are two types of path names: absolute and relative. An absolute path name specifies the directory from the root directory and a relative path name specifies the directory from the current directory.

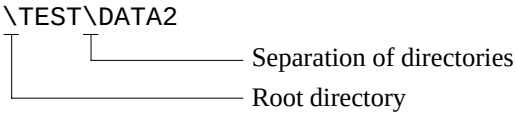
• Absolute path name

An absolute path name tells how to find its way to the desired directory from the root directory. The back slash must be described at the beginning of the path name. The start point of an absolute path name is always the root directory, so when specifying a specific directory or a file in that directory, the path name is always the same no matter where the current directory is.

<Example>



When the current directory is DATA1, you specify the DATA2 directory.



The back slash is used as both root directory and separation of directories, so be careful not to mix the two up. In the case of an absolute path name, the back slash at the beginning indicates the root directory, and others indicate the separation of directories.

The route for finding the directory of this absolute path name is as follows:

Root directory (start point) → Sub directory (TEST) → Sub directory (DATA2)

• Relative path name

A relative path name tells how to find its way to the directory from the current directory. In the case of relative path name, the start point is the current directory, so a path name differs according to where the current directory is. If you use the symbol " . . " (two periods) in a path name, it tells to move upward one level in the tree.

<Example> The expression in absolute path name in the example above corresponds to the following expression in relative path name.

In the case of a relative path name, all back slashes indicate the distinction of directories.

The . . (two periods) heading indicates the parent directory. In other words, the TEST is the parent directory of the DATA2 directory. The following two periods indicate the parent directory, or the root directory.

The route for finding the directory of this relative path name is as follows:

Current directory1 (DATA1 as start point) → Parent directory (TEXT) → Parent directory (root) → Sub directory (TEST) → Sub directory (DATA2)

Environment variable

It is possible to specify the optional parameter in COM and PLI command. The optional parameter can be specified automatically without specifying them whenever executing the command.

When you have many sub directories, you must specify the path name of the file you want to call up if it is not in the current directory. You many specify the environment of the directory beforehand for executing the file used frequently, so you don't have to keep specifying the path name.

To specify such executing environment, use SETENV command to specify the environment variable. The environment variables below are valid.

COM	Set the optional parameter (-V, -L) of COM command.
PLI	Set the optional parameter (/W) of PLI command.
XQT	Specify the path name of the file which is executed by XQT "file name". When not specified, the file will be searched out from the current directory.
PATH	Specify the path name (directory) used for batch file. When not specified, the batch file cannot be executed.

The [path name] on the right of the XQT and PATH can be written several times on the same line. Separate them using the " ; " (semicolon). Make sure that the character limit per line is 79 characters.
You can also specify the PATH without using SETENV command.

Refer to SPEL III reference manual for derails of each command.

<Example of environment variables>

```
>SETENV COM=-V          ' Specifies the compiling condition.
>SETENV PLI=/W          ' Specifies display format for PLIST.
>SETENV PATH=\;\BIN     ' Specifies path for executing of batch file.
>SETENV XQT=\           ' Specifies the path for executing files.
>SETENV                 ' Displays current setting.
COM=-V
PLI=/W
PATH=\;\BIN
XQT=\

>PATH                   ' Displays current path for executing a batch file.
PATH=\;\BIN
>PATH=\                 ' Specifies path for executing a batch file.
>PATH                   ' Displays current path.
PATH=\
```

NOTE

Even when the PATH and XQT environment variable has been established, if a directory is specified with the file name, it will not search using the environment variable.

<Example> `XQT=\BIN;\TEST`

① When executing `XQT "TMP"` (TMP is file name), it will search "TMP" first, and "\BIN\TMP", then "\TEST\TMP".

② When executing `XQT "\TMP"`, it will search "\TMP" file only.

APPLIED SECTION

CHAPTER 1. MULTI-TASKING

1.1 What is Multi-tasking

When a series of commands to be executed are divided up into smaller units by processing function, each of those processing functions is called a "task". "Multi-tasking" refers to a type of processing in which multiple tasks are executed simultaneously or in turns.
There are two ways to achieve multi-tasking. One is called "multiprocessing", a mode of operation in which two or more connected processing units each carry out one or more processes concurrently. The other is "timesharing", a mode of operation in which a single processing unit allots time to the execution of two or more tasks so that operations seem to be executed simultaneously.
Our robot controllers realize multi-tasking by means of timesharing.

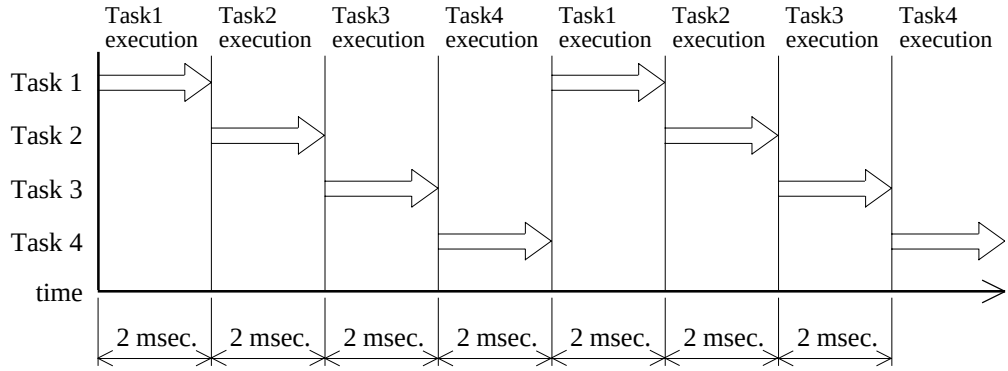
Advantages of multi-tasking

A multi-tasking system has the following advantages:

- Shortened tact time — Single tasking carry out one task only, multi-tasking carry out plural tasks simultaneously. This means plural works are done simultaneously and shorten the tact time (working time) substantially.
- Productivity — In addition to shorten the tact time, multi-tasking can control peripheral equipments simultaneously, this means that all system work efficiently and the productivity is improved.
- Ease of maintenance — By dividing the program into each task, it is easy to check the program and it is convenient to maintain the program by checking out each task.
- Expendability — In order to add new work to the program, modification of the program can be done easily by adding new task.

Multi-tasking in SPEL III

The SRC-300/310A/320 controllers can concurrently execute a minimum of 16 tasks by means of timesharing.
Each task is allowed 2 msec. for execution, and tasks are switched every 2 msec under the control of the system.



SPEL III makes FEND from the FUNCTION of the program one task.
The XQT command of TEACH mode and the START switch of AUTO mode execute FUNCTION... FEND at the top of the program as task 1, and thereafter tasks are launched in accordance with the XQT command in the program.

<Example>

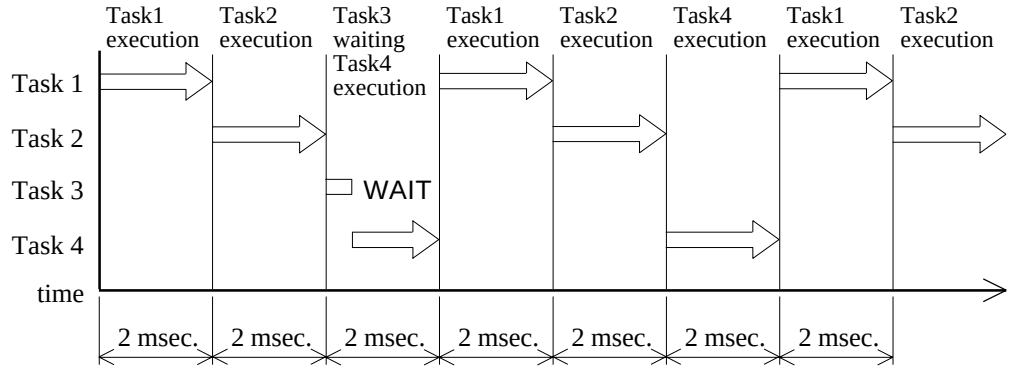
```
1000 FUNCTION MAIN     ' task 1
1010 XQT !2,TASK2      ' start of task 2
1020 XQT !3,TASK3      ' start of task 3
1030 XQT !4,TASK4      ' start of task 4
1040 XQT !5,TASK5      ' start of task 5
  :
1500 FEND
2000 FUNCTION TASK2    ' task 2
  :
2500 FEND
3000 FUNCTION TASK3    ' task 3
  :
3500 FEND
4000 FUNCTION TASK4    ' task 4
  :
4500 FEND
5000 FUNCTION TASK5    ' task 5
  :
5500 FEND
```

Tasks during execution of WAIT command, INPUT command and movement commands

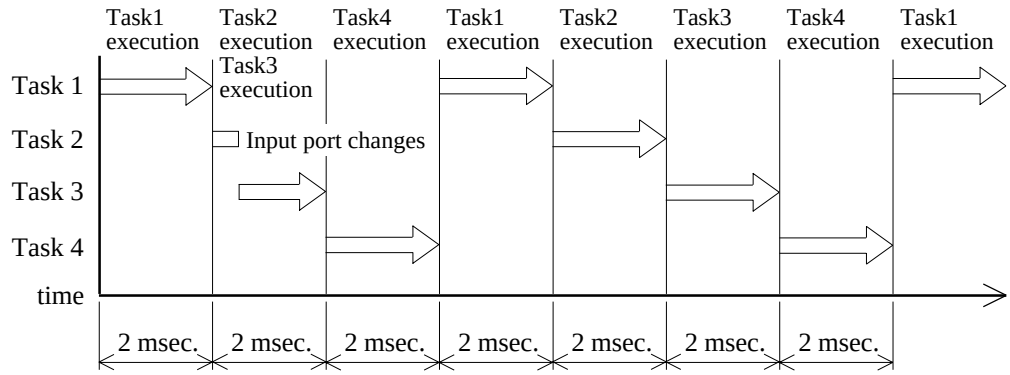
The tasks that execute such commands as the WAIT t command, wait WAIT SW ( ) command, INPUT command, operation commands (JUMP, and so on) are switched by time, and separated from the group of immediately executable tasks. This arrangement is used because assigning an execution time to tasks whose only job is to wait would only contribute to decreasing the processing efficiency of the CPU.

For these tasks, the system supervises the satisfaction of the input condition, entry of data, or the completion of movement, and when the required condition is met, the system executes them with priority over other tasks. Thereafter, they will again be handled as tasks to be switched by time for execution.

<Example> Task 3 executes the WAIT command.



<Example> Task 2 input port changes.



WAIT command and IF sentence

Based on the above, the following two programs have different meanings:

List 1

```
1000 FUNCTION TASK
  ⋮
1200 WAIT SW(1) = 1
  ⋮
1300 FEND
```

List 2

```
1000 FUNCTION TASK
  ⋮
1200 IF SW(1) = 0 THEN GOTO 1200
  ⋮
1300 FEND
```

Both of them wait for input 1 to change. List 1 is excluded from the time switching of tasks, and List 2 repeats the evaluation of equation `SW (1) = 0` for a period of 2 sec., after a certain time has elapsed. If there are multiple tasks like List 2, the majority of the CPU processing time will be spent in evaluating conditional expressions.

Moreover, if, for example, there are 16 tasks, they will be executed one every 32 msec. In some cases, List 2 may not be able to respond to signals that are less than 30 msec. wide.

Timing to switch tasks

The switching of tasks takes place in a completely arbitrary position after 2 msec. have elapsed. The values of variables are not guaranteed in the following program:

<Example>

```
1000 FUNCTION MAIN
  ⋮
1200 IF A = 1 THEN ·····    ' A may not be 1 or 2 either.
1210 IF A = 2 THEN ·····
  ⋮
1400 FEND
2000 FUNCTION TASK2
  ⋮
2100 A = 1
  ⋮
2200 FEND
3000 FUNCTION TASK3
  ⋮
3100 A = 2
  ⋮
3200 FEND
```

When data is written from multiple tasks to one variable in an asynchronous fashion, the tasks are switched while `2100 A = 1` is being executed (for example, immediately after data has been written to one byte out of four), and then `3100 A = 2` is executed. Moreover, when the tasks are switched again and the continuation of `2100 A = 1` is executed, variable A assumes a value that is utterly unpredictable.
Moreover, the value of A is rewritten during execution of the part subsequent to THEN, which makes the conditional expression of IF meaningless.

1.2 Interlock among Tasks

Interference of controller

In multi-tasking, processing is usually carried out while mutual interference among the tasks is being avoided and the progress state of the other task is mutually checked. This is called interlock among tasks.

To secure interlock, the SPEL III makes use of memory I/O. Memory I/O is a group of 512 flags (1-bit variables), which are numbered from 0 to 511. It is operated using the On $n/OFF $n, IN ($m)/OUT $m, and d commands.

<Example>

```
1000 FUNCTION TASK1
  :
1100 LOOP1:
1110 ON $0                          'Command for Task 2 to initiate processing.
  :
1200 WAIT SW($1) = 1; OFF $1        'Wait for a processing-complete indication from
                                     Task 2.
                                    'From here on, interference with the device is con-
                                     trolled by Task 2.
  :
1300 GOTO LOOP1
1310 FEND
2000 FUNCTION TASK2
  :
2100 LOOP2:
2110 WAIT SW($0) = 1; OFF $0        'Wait for an command to initiate processing.
                                    'From here on, interference with the device is con-
                                     trolled by Task 1.
  :
2300 ON $1                          'Processing-complete indication.
  :
2400 GOTO LOOP2
2410 FEND
```

Only one device used by multiple tasks

The output of data to the single device available (Display to OPU-300, RS-232C, etc.) should essentially be assigned to the dedicated tasks. If that is not viable, some ingenious scheme must be devised to prevent simultaneous outputs to a single device. Such a scheme is known as “exclusive control.”

<Example 1> No. 0 (zero) of memory I/O is assigned to use/nonuse of the output device.

```
1000 FUNCTION TASK1
1010 XQT !2,TASK2
1020 WAIT SW($0) = 0
1030 ON $0
1040 PRINT "1234567890"
1050 PRINT "1234567890"
1060 PRINT "1234567890"
1070 OFF $0
1080 GOTO 1020
1090 FEND
2000 FUNCTION TASK2
2010 WAIT SW($0) = 0
2020 ON $0
2030 PRINT "abcdefg"
2040 PRINT "abcdefg"
2050 PRINT "abcdefg"
2060 OFF $0
2070 GOTO 2010
2080 FEND
>
abcdefg
1234567890
abcdefg
abcdefg
1234567890
```

In <Example 1> program, if a task is switched immediately after `WAIT SW ($0) = 0`, a character string from the other task will be delivered through interruption in the middle of the output of the three-character strings. To prevent this, the SPEL III provides the ZEROFLG (0) function.

<Example 2>

```
1000 FUNCTION TASK1
1010 XQT !2,TASK2
1020 ON $0
1030 IF ZEROFLG(0) = 1 THEN WAIT SW($0) = 0; GOTO 1020
1040 PRINT "1234567890"
1050 PRINT "1234567890"
1060 PRINT "1234567890"
1070 OFF $0
1080 GOTO 1020
1090 FEND
2000 FUNCTION TASK2
2010 ON $0
2020 IF ZEROFLG(0) = 1 THEN WAIT SW($0) = 0; GOTO 2010
2030 PRINT "abcdefg"
2040 PRINT "abcdefg"
2050 PRINT "abcdefg"
2060 OFF $0
2070 GOTO 2010
2080 FEND
>
1234567890
1234567890
1234567890
abcdefg
abcdefg
abcdefg
1234567890
1234567890
1234567890
```

The ZEROFLG (0) function returns the n-th status of the memory I/O that exists before the immediately preceding `ON (OFF) $n` is executed.

CHAPTER 2. PROGRAM TECHNIQUES

2.1 How to Write Large-scale Programs (Efficient use of CHAIN/LINK)

A small-scale program normally consists of a single file. As programs become larger, however, the following problems appear:

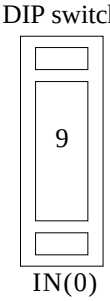
- Cannot fit in program areas.
- Cannot be edited by the editor.
- Require a long time to compile.
- Require a long time to output LIST.
- All variables and labels need to be assigned different names (it is troublesome to create appropriate names).

To be able to cope with such problems, the CHAIN command and LINK command are provided.

The former is used for switching an equipment type, and the latter is used in all other cases. There is no clear-cut criteria for using the CHAIN and LINK commands. Instead, it is all left to the discretion of the programmer, except when some special circumstances requiring the use of those commands arise, such as when a program does not completely fit in a program area. As a general rule of thumb, it is advisable to consider using the LINK and CHAIN commands in writing programs that exceed 3,000 lines (if a line number is assigned to every 10 lines starting from 10, the line numbers go up to 32,767 and, therefore, the limit is 3,276 lines).

Case in which the CHAIN command can be used:

Suppose a program in which different types of IC are to be specified by DIP switches, which represents a case in which routines to be executed are clearly separated.



Work (IC) to be handled is selected by input IN(0), and branched to each processing routine. Each routine is independent of the others, and does not share any subroutine with them.

IN(0)= 1 : PLCC
2 : DIP
3 : SOP

<Example 1> When CHAIN is not used:

```
100  IF IN(0)=1 THEN GOTO PLCC       ' PLCC selection
110  IF IN(0)=2 THEN GOTO DIP        ' DIP selection
120  IF IN(0)=3 THEN GOTO SOP        ' SOP selection
  :
3000 PLCC:
  :
3900 GOTO PLCC
4000 DIP:
  :
4900 GOTO DIP
5000 SOP:
  :
5900 GOTO SOP
```

<Example 2> When CHAIN is used:

File: MENU.PRG

```
100  IF IN(0)=1 THEN CHAIN "PLCC"   ' PLCC selection
110  IF IN(0)=2 THEN CHAIN "DIP"    ' DIP selection
120  IF IN(0)=3 THEN CHAIN "SOP"    ' SOP selection
```

File: PLCC.PRG

```
1000 FUNCTION MAIN
1010 PLCC:
  :
1900 GOTO PLCC
2000 FEND
```

File: DIP.PRG

```
1000 FUNCTION MAIN
1010 DIP:
  :
1900 GOTO DIP
2000 FEND
```

File: SOP.PRG

```
1000 FUNCTION MAIN
1010 SOP:
  :
1900 GOTO SOP
2000 FEND
```

In a case such as the one given in this example, programs can be developed separately for each individual equipment type. This offers good debugging efficiency and, therefore, serves to shorten the development time.

Moreover, in other cases, such as when a program to be activated is selected by REMOTE3 (LCD touch panel, sequencer, etc.), and not through the OPU-300, the CHAIN can also be used.

Case in which the LINK command can be:

The LINK command is used under the following circumstances, among others:

- A program is too large to fit in the program area.
- A program is too large to be edited by the editor.
- It is desirable to use local variables.
- Programming work needs to be shared among several persons.
- A program should be made up of modules so that it can be reused.
- Shortening compiling time to correct the programs.
- The same names can be used for variables or labels, if the files are different.

If any of the above cases, use of the LINK command's recommended. However, you should also bear in mind that using the LINK command necessarily involves by the following disadvantages:

- To share variables, extra procedures, such as ENTRY/EXTERN, are necessary.
- The number of procedures as a whole increases. (In addition to compiling, the LINK command must be executed, and if there is any mismatch (non-correspondence between ENTRY and EXTERN), the procedures need to be repeated starting with the compiling.)
- Various files need to be handled at the same time.

2.2 Movement to Multiple Points Spaced Equidistantly

The PALET command is effective for moving to multiple points arranged equidistantly, as illustrated below:

Definition of pallet

Format : PALETn Pa,Pb,Pc, [No. of divisions 1], [No. of divisions 2]

n: Pallet No. (0 to 15)
a, b, c: Point No. (0 to 199)
[No. of Divisions 1]: Number of divisions between Pa and Pb
[No. of Divisions 2]: Number of divisions between Pa and Pc

<Example>

```
PALET1, P1,P2,P3,5,3
```

When input is done as shown in the example, the pallet illustrated at right is set, and numbers are assigned as shown there.

P3 • 11	• 12	• 13	• 14	• 15
• 6	• 7	• 8	• 9	• 10
P1 • 1	• 2	• 3	• 4	P2 • 5

* The corner of the pallet formed by the three points must be situated at "Pa".
* If the standard three points do not form a right angle, the pallet will be deformed.
* Z-coordinates are distributed equidistantly on a plane formed by three points.

One-string pallet:

For a one-string pallet, teaching is done on the two points on both sides, and is therefore defined as follows:

Format : PALETn P1,P2,P1,[No. of divisions],1

<Example>

```
PALET1 P1,P2,P1,5,1
```

P1 • 1	• 2	• 3	• 4	P2 • 5

Positional designation inside pallet:

Prior to executing the program, directly input the JUMP command and move the robot to each point inside the pallet and make sure it is the right position. If you have a pallet image drawn beforehand, it will facilitate checking the movement to each point.
Specify the position inside the pallet as follows.

Format : PALETn (C)

n = 0 to 15 (Pallet No.)
C = Grid No. on the pallet

NOTE

An error occurs when the pallet is not defined. Prior to executing the program, set the three points.

<Example> The work is switched from Pallet 1 to Pallet 2.

```
10  FUNCTION MAIN
20    PALET1 P1,P2,P3,3,5
30    PALET2 P11,P12,P13,5,3
40    FOR I=1 TO 15
50      JUMP PALET1(I)
60      ON 1 ; WAIT 0.5
70      JUMP PALET2(I)
80      OFF 1; WAIT 0.5
90    NEXT I
100  FEND
```

The I/O output port numbers for operating the hand and the numbers of pallet divisions should be set individually.

2.3 Techniques for Shortening Cycle Time

SPEL III has various commands for shortening cycle time. There are four approaches to reducing cycle time:

- move at the highest speed possible
- perform concurrent operations
- eliminate wasteful movements
- eliminate wasteful positioning

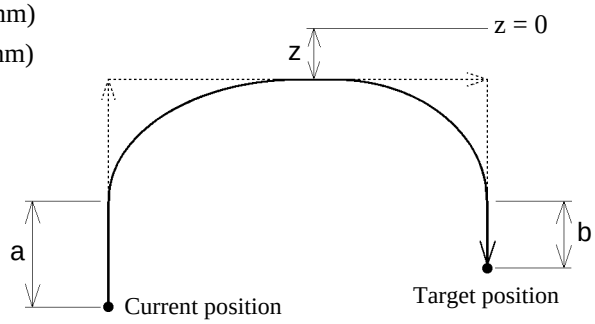
Using arch motion

Horizontal motion is initiated during vertical movement by operation of the JUMP command.

The shape of movement can be selected freely.

Vertical rising distance (a : mm)
Vertical lowering distance (b : mm)
Horizontal movement height (z : mm)

Commands: JUMP, ARCH, LIMZ



Free setting of the timing of position completion

Set the value for completing the positioning of a motion, and execute the following command in front of the target position. For example, if the following command is ON/OFF, delay in such things as compressed air operation can be absorbed.
The final position of the movement destination does not change even if the FINE command is used. The timing for executing the next command is merely speeded up.

Select the best value for your system, somewhere around 500 to 1000 pulses for normal operations and 100 pulses for assembly operations.

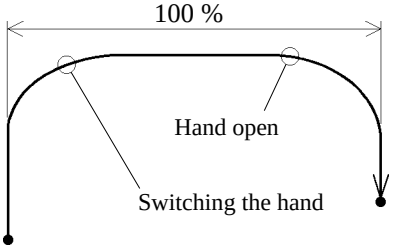
Command: FINE

Parallel processing

During operations, input and output of the I/O and RS-232C, etc. can by performed in the place of your choice (horizontal direction).

<Example> • Opening and closing hand during pick-and-place operations (see following figure).
• Turning the nozzle in painting operations on and off

Command: ! ... !

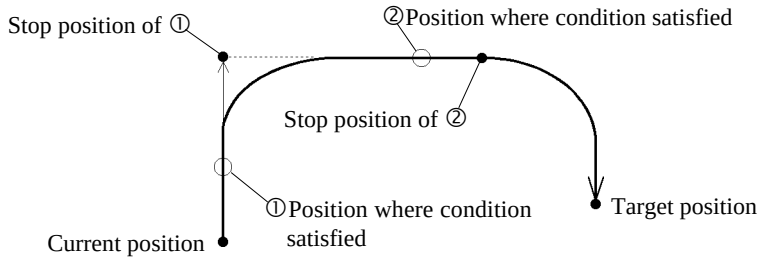


Conditional stop during motion

When input conditions are established during motion, the robot immediately slows and stops. Here is the example that stop occurs when the condition "no work-piece" is satisfied during pick-and-place operation. When the stop occurred in the absence of a work-piece, wasted motion in retrying is eliminated.

Commands: JUMP, SENSE, TILL

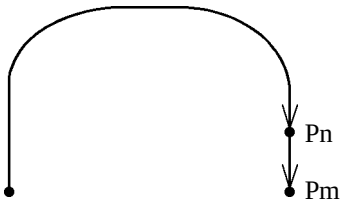
<Example> JUMP with TILL modifier



Assembly operations at low speed

When it is necessary to slow down during assembly, one often sees instances of programming with the JUMP command and GO command, as shown in <Example 1>. However, since positioning is performed once, it cannot match a single movement command, even if FINE is roughened.

<Example 1>
```
JUMP Pn
GO Pm
```



In this case, it is better to shorten cycle time by skillfully using the ACCEL command, making it a JUMP command.

<Example 2>
```
ACCEL 100,100, 100,100, 100,10 ;JUMP Pm
```

Acceleration/deceleration during descending.
Acceleration/deceleration during rising
Acceleration/deceleration during horizontal movement

Associated commands

Five examples were given above. The associated commands are listed below. See the SPEL III reference manual for details about commands.

- move at the highest speed possible: SPEED, SPEEDS, ACCEL, ACCELS, WEIGHT
- perform concurrent operations: !...! (parallel processing), multi-tasking
- eliminate wasteful movements: ARCH, LIMZ, SENSE, TILL
- eliminate wasteful positioning: PASS, FINE, CMOVE, CARC

2.4 Using Position Data

When you have to switch position data because a single program handles two or more work-pieces, it is convenient to store work-piece specific data in the position data.
For example, it is convenient to do this when there are two kinds of work-pieces that differ in the number of work-pieces that can be lined up on a single pallet; or when the operating speed is changed depending on the work-piece to be handled.

To obtain coordinate values from the point data, use functions CX (Pn), CY (Pn), CZ (Pn), and CU (Pn).

<Example>
Enter the number of work-pieces in X of point No. 190 of the work-piece 1 point file (e.g. 01WORK.PNT) and the operating speed in X, Y, Z, and U of No. 191.

```
  :
P190=10,0,0,0
P191=100,100,20,50
```

Just as for work-piece 1, enter the number of work-pieces in X of point No. 190 of the work-piece 2 point file (e.g. 02WORK.PNT) and the operating speed in X, Y, Z, and U of No. 191.

```
  :
P190=12,0,0,0
P191=80,80,20,50
```

Obtain data in the program as follows.

```
  :
1100 COUNT = CX(P190)                    'Obtain the number of work-pieces
                                          to be loaded on one pallet from
                                          P190
1110 SPD1 = CX(P191);ACL1 = CY(P191)     'Obtain speed and acceleration/de-
                                          celeration values from P191.
1120 SPD2 = CZ(P191);SPD3 = CU(P191)
  :
1200 SPEED SPD1; ACCEL ACL1,ACL1
  :
1500 FOR I = 1 TO COUNT
1510   JUMP PALET1(I)
  :
1590 NEXT
  :
1700 SPEED SPD2
1710 GO P10
1720 SPEED SPD1
  :
1900 SPEED SPD3
1910 GO P11
1920 SPEED SPD1
  :
```

CHAPTER 3. DEBUGGING

3.1 Multi-tasking Debugging

Convenient debugging commands

SPEL III has a number of commands that help debug multi-tasking programs. These commands are executed in command mode of SPEL Editor or monitor window of SPEL for Windows.

- XQT: Task execution
- TSTAT: Display of task status
- TON/TOFF: Display of execution line No. to indicator panel of controller and programming unit (PC)
- PRINT: Display of text strings, variables, I/O, memory I/O

XQT command

The XQT command executes a program.

When this command is input from a programming unit (PC), the FUNCTION...FEND at the start of the program is executed as task 1, and the XQT command is complete when task 1 is finished. Therefore, the next command cannot be input from the programming unit until task 1 is finished.

With the XQT command, you can specify the task number of the program to be executed, specify the FUNCTION, and specify the line number. You can execute a part of a program by specifying the line number, thus enabling you to check particulars.
You can also use the TSTAT command, explained in "TSTAT command" below, if you specify the task.

<Example 1>

```
>XQT !2,MAIN                ' Start FUNCTION MAIN as task 2
>                           ' Possible to input commands when program is running
```

<Example 2>

```
>XQT !2,MAIN,2100-2200      ' Execute FUNCTION MAIN lines 2100 to 2200
>
```

TSTAT command

Displays the status of tasks 1 to 16 when TSTAT command is executed.
You can check the execution status of all tasks using this command.

From SPEL Editor's command mode, specify any task number from 2 to 16 using the above-mentioned XQT command and execute the program.

<Example>

```
>XQT !2,MAIN
>TSTAT
 Task       1      2      3      4      5      6      7      8
Status    QUIT    RUN    RUN   QUIT   QUIT   QUIT   QUIT   QUIT
 Line        0    100    250      0      0      0      0      0

 Task       9     10     11     12     13     14     15     16
Status    QUIT   QUIT   QUIT   QUIT   QUIT   QUIT   QUIT   QUIT
 Line        0      0      0      0      0      0      0      0
```

Displays the last line number to have been executed prior to status of each task, QUITE, RUN, HALT, and TSTAT execution.

In SPEL for Windows, you can check the status of tasks in the [Task Manager] window. Please see “Task Manager” in the SPEL for Windows manual for details.

TON/TOFF command

You can find out the program sequence from the line number currently being executed.

There is an LED on the front panel of the controller that displays the line number. Normally the task 1 line number being executed is displayed here. If you use the TON command you can view the line numbers of tasks other than task 1 that are being executed.

<Example>

```
>XQT !2,MAIN
[1010]      ' Display the line number being executed
[1020]
  :
```

PRINT command

The simplest debugging technique is to display the text string midway through the program and check its operation. PRINT is a command that displays on the programming device such things as the text string, variable value, and function value.

<Example>

```
1000 FUNCTION MAIN
1010 XQT !2,TASK2
1020 PRINT "Start Task2"
  :
1200 A = A + B + C
1210 PRINT "A = ",A
  :
```

They are displayed as follows on the screen when you execute this program.

```
Start Task2
A = 100
```

CHAPTER 4. BATCH PROCESSING

In MS-DOS, there is a processing called "batch processing" which automatically executes commands and statements in series. The file that registers the continually executed commands is called the "batch file".

SPEL III offers the usage of batch file as MS-DOS does.
The SPEL III commands and statements can be described in the batch file.

The file extension of the batch file must be BAT. By naming the extension BAT, the file is recognized as batch file, and the commands and statements registered in the file will be executed automatically.

When executing the batch processing, input the batch file name as command. However, the extension (.BAT) is not necessary to be input.

EDIT command can be used for editing the batch processing file. Regarding the EDIT command, refer to SPEL III reference manual, and "6.2 Editing Files" in this section of this manual.

4.1 Batch Processing Command

In SPEL III, there is an "ECHO" command to control the display of commands and messages during batch processing.
The format is as follows:

Format:
```
ECHO ON
ECHO OFF
ECHO [message]
```

Usually the specified command in the batch file will be displayed to the console. The ECHO command controls this display. When displaying the command which is executed (echo back), specify "ECHO ON", and when not displaying the command, specify "ECHO OFF". Usually it is in the "ECHO ON" status. In other words, even if you input ECHO OFF, after the batch file is completed, it will automatically return to ECHO ON status.

If the "ECHO [message]" is input, that message will be displayed to the console. Even when ECHO OFF is specified, the message can be displayed, so it is useful to output the message as a record of the command executed. In [message], excepting the control codes (character codes 00H to 1FH), all of the characters may be used.

In the case of MS-DOS, when ECHO command without ON, OFF, nor [message] is input, the current ECHO status will be referred. However, SPEL III does not offer this function. Therefore, when inputting ECHO only, the space characters are regarded as message, and a line is changed.

<Example> File name "ECHO.BAT".

```
10 ECHO OFF
20 ECHO MESSAGE1
30 ECHO
40 ECHO ON
50 ECHO MESSAGE2
```

When execute the example program above, the execution result will be as shown below:

```
>ECHO
MESSAGE1      ' Displays message
              ' Changing line by ECHO
MESSAGE2      ' Displays message
```

The point which differentiates the SPEL III "ECHO" command from the one of the MS-DOS is that even in the ECHO ON status, the ECHO command display will not appear. For instance, in the above example, if the same batch file is run in MS-DOS, the execution results will appear as shown below:

```
>ECHO
ECHO OFF          ' 1)
MESSAGE1
ECHO <OFF>        ' 2)
ECHO MESSAGE2     ' 3)
MESSAGE2
```

1) Before executing ECHO OFF, it was in the ECHO ON status, so that status is displayed.

2) This shows the current set up status of ECHO.

3) Display due to ECHO ON status. (The string of ECHO ON will not be displayed since it is in the ECHO OFF status.

Regarding the batch processing and the display of batch file, refer to the MS-DOS manual.

4.2 Batch File

There are many commands like COMPILE command which cannot be executed in SPEL III program. However, if it is described in a batch file, these commands can be executed.

Creating the batch file

When editing batch files, you must first enter the EDIT mode by specifying the EDIT command. Refer to "6.2 Editing Files" in this section of this manual.

The batch file is named with the BAT extension. A file cannot be named with the same word as a SPEL command. It is because the command has priority to be executed.

The commands which are effective as direct commands can be described in a batch file. Those commands are executed in order when running. You cannot call up another batch file from a batch file.

CHAPTER 5. Automatic Program Execution at Power On

It is possible to start program execution automatically when turning the power on.

This automatic execution can be done by creating an "AUTO.BAT" file or "IPL.OBJ" file in the root directory of the current drive.

5.1 AUTO.BAT File

Create this file in the same way as an usual batch file with EDIT command. By naming this file "AUTO.BAT", you are able to automatically execute the commands registered in this file. (See "6.2 Editing Files" for details of EDIT command.)

<Example> An example to execute the executable file "MAINGRP" automatically after turning on the controller.

```
>EDIT AUTO.BAT
New file
>10 XQT "MAINGRP"
>END
Edit End...file save
>
```

This batch file can be executed regardless of what the starting mode is.

5.2 IPL Program

By creating the file with "IPL" file name, XQT"IPL" can be executed when the power is turned on. However, the start up mode must be in AUTO mode.

CHAPTER 6. SYSTEM CONFIGURATION FILE

6.1 CNFG.SYS File

In order to use the computer efficiently, you may need to define the system information according to the hardware and program used. This kind of information is described in the "CNFG.SYS" file, and specifies the system environment.
The requirements which can be defined in SPEL III are the capacity of error history buffer, the capacity of line history of each task, and the maximum number of user tasks.

When the "CNFG.SYS" file does not exist, default values are used for the system configuration.

The "CNFG.SYS" file must be registered in the root directory. Even if the "CNFG.SYS" file exists somewhere other than the root directory, it will be disregarded.

TASK=k	Specifies the available number of user tasks in k. k: Integer from 1 to 16 [Default value :16]
ERRBUF=m	Memorizes the error history up to m. m: Integer from 1 to 20 [Default value :20]
LINBUF=n	Memorizes line number history of each task up to n. n: Integer from 1 to 512 [Default value :10]

As to the requirements above, system memory area is assigned to the memory capacity in byte as shown below.

	Necessary memory [unit: byte]
User task	(Specified number) × (approx. 1600)
ERRBUF	(Specified number) × 8
LINBUF	(Specified number) × 6 × (Number of tasks specified by TASK)

The system memory area is used to secure the necessary RAM for running tasks. Therefore, if too large amount is set for "ERRBUF" or "LINBUF", task may not be run. Be careful not to specify more space than is necessary.

Also make sure to define the "TASK=k" in the CNFG.SYS file before the "LINBUF=n". If it isn't, the area secured for LINBUF will be the one for 16 tasks.

6.2 Editing Files

SPEL III offers the EDIT command which is for editing the text file (excepting program). The format of EDIT command is as follows:

Format: EDIT [path name][file name]

By using the EDIT command, it switches to the edit mode, and the files can be edited. To edit a file, create lines with line numbers in the same way that a program is created. However, it is not necessary to describe "FUNCTION [Function name]" and "FEND".

In the edit mode it is possible to use the commands as listed below.

LIST, RENUM, DELETE(DEL), NEW, FREE, COPY, RENAME(REN), CHDIR(CD), RMDIR(RD), MKDIR(MD), RENDIR, DIR, QUIT, END

When entering the edit mode, you specify the path name and file name. Then if this file name exists in the directory, the file will be read.

```
End of input file
```

And the above message will be displayed. When the specified file name does not exist,

```
New file
```

the above message will be displayed. Then file editing becomes possible.

To get out of the edit mode, you may use QUIT command (end without saving) or END command (save then end). The following messages will appear in each case.

```
>QUIT
Edit End...

>END
Edit End...file save
>
```

The program editing area for the edit mode and for the robot program is different. Therefore, the program in the robot program area will not be obstructed by editing a file in the edit mode.

EDIT command deletes line numbers when saving files, and when reading the file, the line numbers will be added as 10, 20, 30.... Therefore, the line numbers before and after saving will differ.

<Example>

```
>EDIT CNFG.SYS
New file
>10 TASK=8
>20 ERRBUF=20
>30 LINBUF=256
>END
Edit End...file save
>
```

NOTE

The number of characters in one line is up to 79 including line numbers. Be aware that the number of characters that will be read (not including the line numbers) is 74, and anything over that will be cut.

The file size that can be edited is at the most 6 Kbyte.

CHAPTER 7. RS-232C

7.1 Overview of RS-232C

The robot controller SRC-300/310A/320 has an RS-232C interface as its standard interface, and its communication is supported by the SPEL III robot programming language.
The RS-232C interface can be used for the following type of communications.

- Communication port for the SPEL Editor or the SPEL for Windows
- Communications between robots
- Communications with user equipment
- Communications with a host computer

Configuration

Because SPEL III uses asynchronous communications, the various communication settings (such as the communication mode, protocol, and baud rate) must be the same among all communicating devices. This group of settings is called the "configuration."

Communication mode
The communication mode includes the data bit, parity, and stop bit settings which are made for data transferred via the RS-232C interface.

Protocol
The protocol is the sequence in which data is transferred between communicating stations.
SPEL III supports "BASIC protocol" and "TTY protocol" settings. The "BASIC protocol" is recommended to ensure the reliability of communication data.

Configuration for SPEL Ⅲ

The SPEL III configuration can be set via the following command.

Format: CONFIG #[port no.],[mode no.],[protocol no.],[timeout period],[baud rate no.]

Port No. : 20 or 21 (an integer from 20 to 23 for extended ports)
Mode No. : Integer from 0 to 47
Protocol No. : Integer from 0 to 19
Baud rate No.: Integer from 0 to 7

The following values are set as initial values for the controller's RS-232C port. These same values are also set when the VERINIT command is executed.

Mode No. : 2 (7 bits, even parity, 1 stop bit)
Protocol No. : 1 (BASIC protocol, secondary station)
Time-out period: 3 (3-second timeout period)
Baud rate No. : 0 (9,600 bps)

Port No.

This is the RS-232C connector number that is shown on the back of the controller.

Mode No.

Mode No.	Character bits	Parity	Stop bits	Mode No.	Character bits	Parity	Stop bits
0	7	EVEN	2	24	6	EVEN	2
1	7	ODD	2	25	6	EVEN	1.5
2	7	EVEN	1	26	6	EVEN	1
3	7	ODD	1	27	-	-	-
4	8	NONE	2	28	6	ODD	2
5	8	NONE	1	29	6	ODD	1.5
6	8	EVEN	1	30	6	ODD	1
7	8	ODD	1	31	-	-	-
8	7	EVEN	1.5	32	6	NONE	2
9	-	-	-	33	6	NONE	1.5
10	7	ODD	1.5	34	6	NONE	1
11	-	-	-	35	-	-	-
12	7	NONE	2	36	5	EVEN	2
13	7	NONE	1	37	5	EVEN	1.5
14	7	NONE	1.5	38	5	EVEN	1
15	-	-	-	39	-	-	-
16	8	EVEN	2	40	5	ODD	2
17	8	EVEN	1.5	41	5	ODD	1.5
18	-	-	-	42	5	ODD	1
19	8	ODD	2	43	-	-	-
20	8	ODD	1.5	44	5	NONE	2
21	-	-	-	45	5	NONE	1.5
22	8	NONE	1.5	46	5	NONE	1
23	-	-	-	47	-	-	-

Character bits

This is the number of bits used to represent text characters during data transmission.

Parity

The parity bit is a bit that is used to ensure the reliability of data communications.

EVEN : A parity bit is added so that the total number of 1 in the bit string becomes an even number.

ODD : A parity bit is added so that the total number of 1 in the bit string becomes an odd number.

NONE : No parity bit is added.

Stop bits

The stop bit indicates the end of transmitted data. Actually, it indicates a bit time value, so that "1.5 stop bits" means the amount of time required for 1.5 bits is necessary as the stop bit.

Protocol No.

Protocol No.	Protocol	Station	Buffer busy control	Terminator
0	TTY	-	None	CR
1	BASIC	2	-	-
2	BASIC	1	-	-
3	TTY	-	XON/XOFF	CR
4	TTY	-	XON/XOFF	CR-LF
5	TTY	-	XON/XOFF	LF
6	TTY	-	None	CR-LF
7	TTY	-	None	LF
8	BASIC2	2	-	-
9	BASIC2	1	-	-
10	TTY	-	CS	CR
11	BASIC	2	CS	-
12	BASIC	1	CS	-
13	TTY	-	CS & XON/XOFF	CR
14	TTY	-	CS & XON/XOFF	CR-LF
15	TTY	-	CS & XON/XOFF	LF
16	TTY	-	CS	CR-LF
17	TTY	-	CS	LF
18	BASIC2	2	CS	-
19	BASIC2	1	CS	-

NOTE

Protocol Nos. 10 to 19 are disabled for SPEL III Version 3.2 and earlier versions.

Timeout period

When using the BASIC protocol, if data is not received within the specified period, a communication error (error code 31) occurs. This specified period is called the “timeout period.”

Baud rate No.

Baud rate No.	Baud rate (bps)
0	9600
1	4800
2	2400
3	1200
4	600
5	300
6	19200
7	38400

Computer configuration

The computer that is connected to an RS-232C port must have the same communications configuration as the robot controller.
The method for setting the configuration differs according to the computer. Check the manuals for the connected computer and its operating system such as MS-DOS (*1), then make the same configuration settings on the computer and controller.

<Example>

Computer: IBM PC Series (or compatible)
Operating system: MS-DOS

The device driver provided by MS-DOS may have to be installed to enable RS-232C communications. Install the device driver by adding the following line to the computer's CONFIG.SYS file.

```
DEVICE = ANSI.SYS
```

RS-232C communications is now enabled using the computer's disk on which the device has been installed as shown above. However, you must still set up the communication configuration before you can execute RS-232C communications.
Make sure the configuration settings are the same for the computer and the controller. For example, if using the initial controller settings, set up the following configuration at the computer's DOS prompt (A>).

```
A>MODE COM1 : 9600, E, 8, 2
```

(*1) MS-DOS is a registered trademark of Microsoft Corporation.

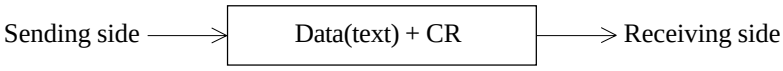
TTY protocol and XON/XOFF control

TTY protocol

The TTY protocol is referred to as “no protocol” since it is a protocol that simply sends data without checking whether or not the receiving side is ready.

TTY adds CR (&H0D) to data as a terminator. If transmission errors or other problems occur, it is impossible to recover transmitted data, so one must be extra cautious when using TTY.

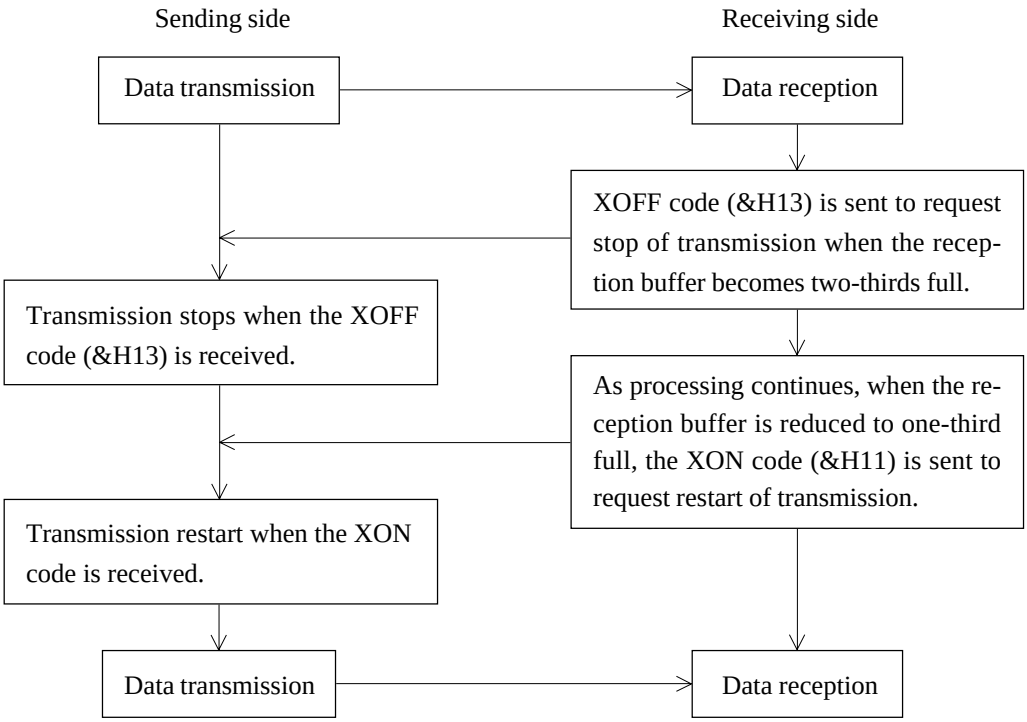
Instead, we recommend using the BASIC protocol since it provides ready-status checking and error detection such as parity checking, which help ensure more reliable communications.
The detail of BASIC protocol is described in next page.



XON/XOFF control

Under the TTY protocol, when data processing at the receiving side does not go smoothly, the received data overflows the reception buffer. XON/XOFF control is a control function that prevents such overflows.

Flowchart of XON/XOFF Control



BASIC protocol

BASIC protocol

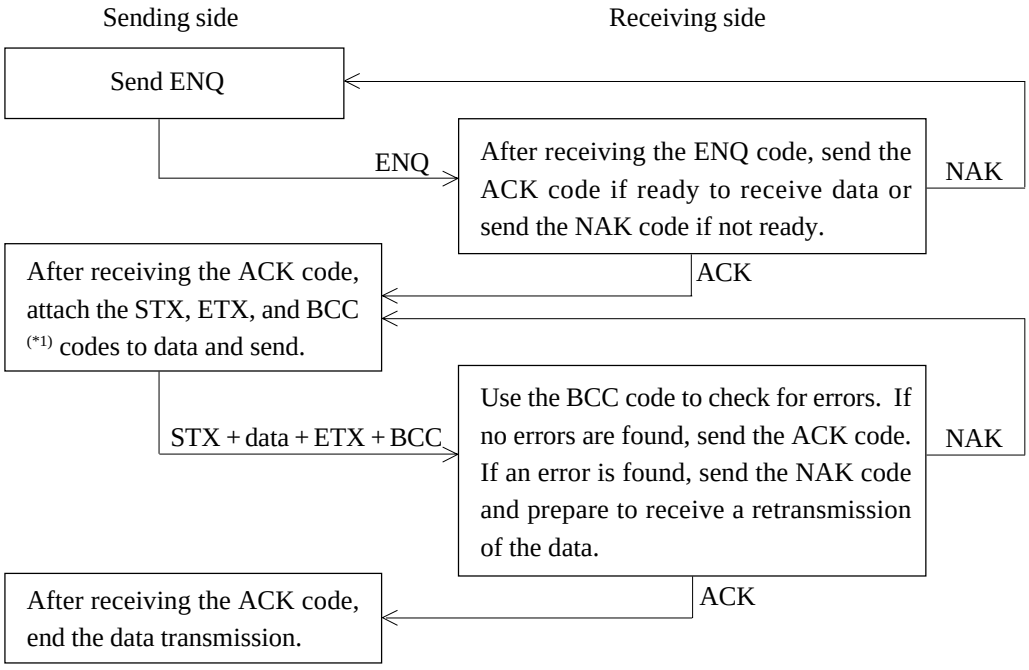
The following rules have been established for the BASIC protocol to make for higher reliability.

- Before the sending side can send any data, it must confirm that the receiving side is ready to receive the data.
- A parity bit is added to transmitted data. The receiving side uses the parity bit to check for transmission errors and notifies the sending side when an error is detected.

These basic rules are collectively called "transmission control." The BASIC protocol uses the following codes to implement transmission control.

Notation	Code	Name
STX	&H02	Start of Text
ETX	&H03	End of Text
EOT	&H04	End of Transmission
ENQ	&H05	Enquiry
ACK	&H06	Acknowledge
NAK	&H15	Negative Acknowledge

Flowchart of BASIC Protocol



(*1) BCC : Stands for “Block Check Character.” This code is used to detect transmission errors, as determined by the horizontal parity value of the text and ETX code (a value that includes the text and ETX code as that is calculated as an exclusive OR).

<Example> When sending the text "RUN" (&H52,&H55,&H4E)

BCC = (R) XOR (U) XOR (N) XOR (ETX)

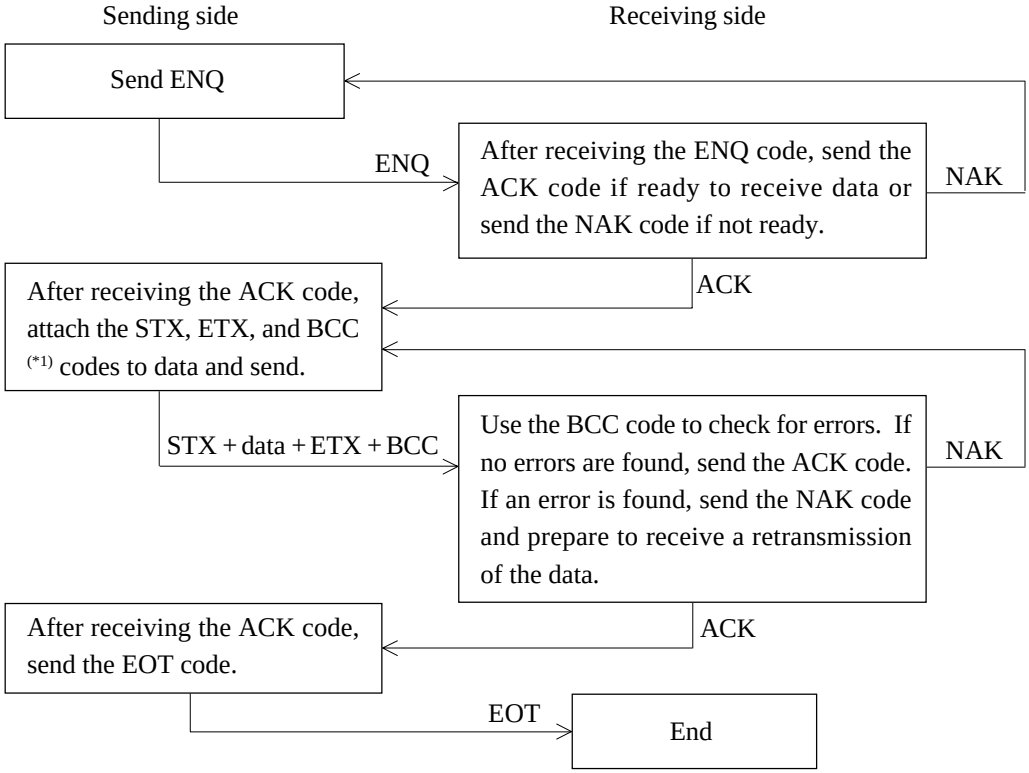
= (&H52) XOR (&H55) XOR (&H4E) XOR (&H03) = &H4A

	R	0101	0010
	U	0101	0101
	N	0100	1110
XOR)	ETX	0000	0011
	BCC =	0100	1010 = &H4A

BASIC2 protocol

The only difference in the BASIC2 protocol is that after the sending side receives an ACK code from the receiving side to acknowledge normal reception, the sending side sends an EOT code.

Flowchart of BASIC2 Protocol



Station (send data collision)

When two people begin a conversation at the same time, it is not possible for either to convey a message. One person must stop talking and allow the other to speak. Similarly, when using the BASIC protocol, it may happen that both sides begin a data transmission at the same time, in which case the two sets of transmission data "collide" with each other. To be precise, two ENQ codes are sent at the same time and they collide.

To avoid such collisions, the two stations are assigned a "primary station" or "secondary station" status. When a collision occurs, the primary station takes priority in resending the data. The secondary station puts its attempted data transmission on pause until it has received the transmission from the primary station, after which it resends the data.

Transmission control via the CS pin

As mentioned earlier, when data processing at the receiving side does not go smoothly, the received data overflows the reception buffer. The TTY protocol's XON/XOFF control is one method for preventing such overflows by controlling "buffer busy" conditions. It does this via the transmission and reception of software codes (DC1 and DC3). The CS pin provides a hardware method for controlling buffer busy conditions.

The CS pin refers to the RS-232C interface pin number 5, which controls the input signal for confirming "clear to send" status. When the CS input is set to "L," data for which transmission has been cleared (enabled) can be output. When this input is set to "H," transmission is suppressed (disabled).

Usually, when not using CS control, both the controller and the other connected equipment feed their RS output (pin number 4, which is always set to "L") directly to their CS pin, so that transmission is set as permanently enabled (see the description below). However, when using CS control, the connected equipment's CS output is connected to the controller's CS pin. This output is set to "L" to enable transmission and "H" to disable transmission.

NOTE


For protocol numbers 0 to 9, the CS pin cannot be used to control transmission: even when the CS input is set to "H" it cannot control transmission.

NOTE


Transmission control via the CS (Clear to Send) pin is not enabled in SPELL III Ver 3.2 or earlier versions.

RS control

RS output is always set to "L" regardless of whether or not RS control is being used. Therefore, it is not possible to have the controller control transmission to the connected equipment at the receiving side. When using the TTY protocol without XON/XOFF control, note that data congestion may occur when sending large amounts of data from the connected equipment to the controller.

Remark: CS transmission control when using the BASIC protocol

The robot controller uses a special-purpose serial chip for controlling RS-232C communications. This chip receives instructions from the CPU and sends data to connected equipment according to a communication protocol, receives data from connected equipment according to a communication protocol, and transfers data to the CPU. As such, this serial chip reduces the CPU's workload during communication control operations.

Transmission control using the CS signal can be done via this serial chip, but in this case the BASIC protocol sends two ENQ codes, as described below.
If the CPU sends an ENQ code when the CS signal goes high (is set to "H"), the ENQ code is received by the serial chip but it cannot be sent to the connected equipment and is instead held internally. The CPU then determines that no answer was received from the connected equipment, and after the specified timeout period has elapsed it sends another ENQ code. Once the serial chip receives an ENQ code (which it cannot send), it remains in standby status until the ENQ can be sent. When the CS signal goes low (is set to "L"), two ENQ codes sent from the CPU are sent from the serial chip to the connected equipment.

RS-232C interface

The RS-232C signal lines are listed below.

Pin No. (D-sub 9)	Pin No. (D-sub 25)	Notation	I/O	Signal name
1	8	CD	Input	Carrier Detect
2	3	RD	Input	Receive Data
3	2	SD	Output	Send Data
4	20	ER	Output	Equipment Ready
5	7	SG	-	Signal Ground
6	-	-	-	-
7	4	RS	Output	Request to Send
8	5	CS	Input	Clear to Send
9	-	-	-	-
-	6	DR	Input	Dataset Ready
-	1	FG	-	Frame Ground

FG (Frame ground)
This is a ground connection for maintenance and safety.

SD (Send Data)
This is the serial data signal line for sending data to the receiving side.

RD (Receive Data)
This is the serial data signal line for receiving data from the sending side.

RS (Request to Send)
Whenever output is low (set to "L"), RS control is disabled. When the CS signal is not being used for transmission control, the RS output is input to the CS pin.

CS (Clear to Send)
This signal provides notification that transmission is enabled. When the CS signal is being used for transmission control, transmission is enabled when this signal goes low (L) and is suppressed it goes high (H).

ER (Equipment Ready)
This signal informs the connected equipment that data transmission is enabled; it is set to ON (L) when the robot controller is powered up. When using MS-DOS, it is set to ON when the RS-232C device driver is installed.

DR (Dataset Ready)
This signal is set to ON when the connected equipment is ready for data transmission.

CD (Carrier Detect)
This signal is set to ON when the remote side’s carrier is being received normally.

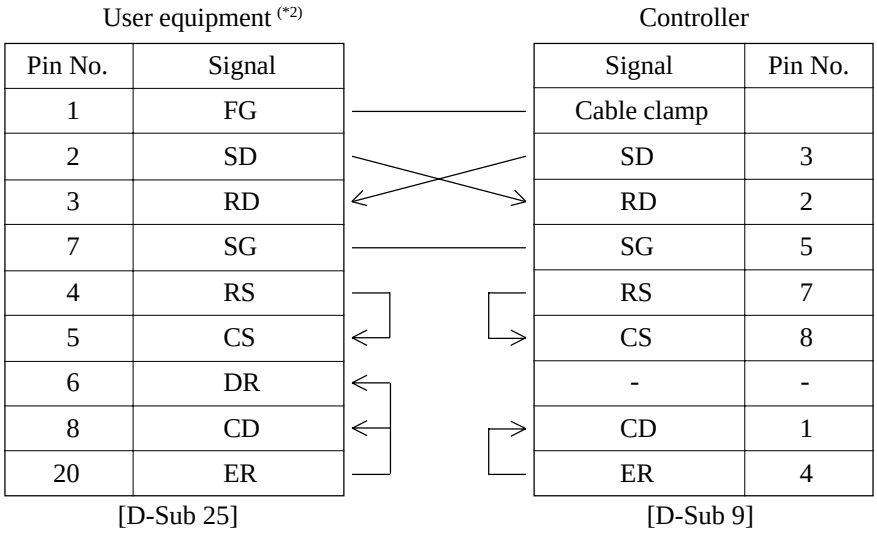
• Manufacture of cross cable

The RS-232C interface is a standard that was developed for connecting DTEs and DCEs (*1). However, the RS-232C interface is used for serial communications between various kinds of equipment. Naturally, different types of equipment use the signal lines differently and therefore require different cross cable routing.

(*1) DTE stands for “Data Terminal Equipment” such as computers.
DCE stands for “Data Circuit Terminating Equipment” such as modems.

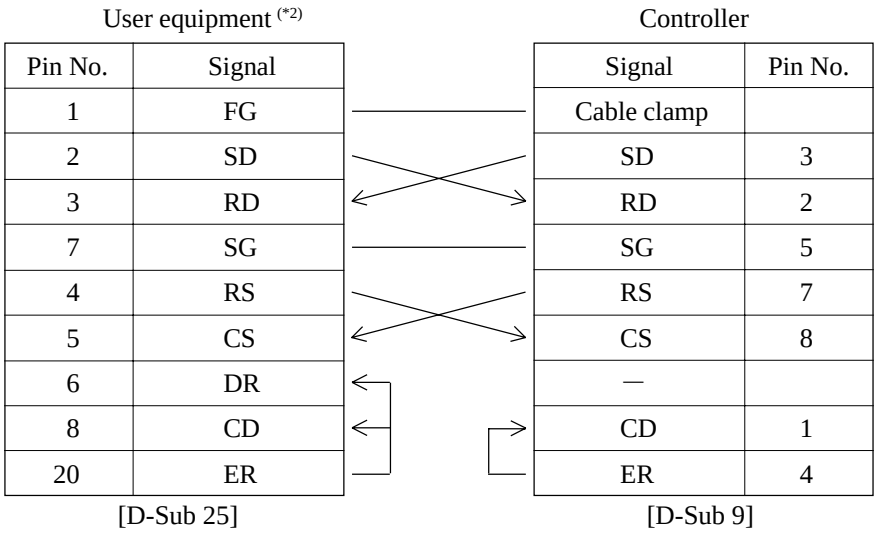
Cross cables should be made according to the following cable routing diagram.

<Routing diagram 1: Routing when CS control is not used>



(*2) The "user equipment" in this above diagram refers to equipment (such as PCs or vision systems) that uses the RS-232C interface.

<Routing diagram 2: Routing when CS control is used>



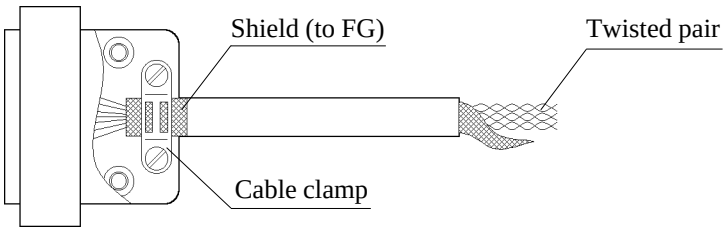
In routing diagram 2, the CS input on the controller side uses the RS output on the user equipment, but this routing can be changed to suit the user equipment’s control method.
Since the controller does not perform RS control, the controller cannot be used to control transmissions from the user equipment.

• Manufacture of connectors on cable

The shield should be handled as follows.

1). After turning back the shield, fasten the cable clamp onto the cable.

2). Fasten the hood onto the connector.



7.2 Communications between Robots

When designing a system that includes several robots, some sort of interlocking mechanism is needed to prevent interference in data transfer between robots or other operations among two or more robots. Connecting the robots via the RS-232C interface enables robot operations to be specified and implemented via user programs.

Configuration settings

NOTE

When using the RS-232C interface for communications, the same configuration must be set at both communicating stations. Use the CONFIG command (described in "7.1 Overview of RS-232C") to set the configuration.
When using the BASIC protocol, be sure to set one station as the primary station and the other station as the secondary station in order to avoid collisions during data transmission.

Communication-related commands

Use communication-related commands that are supported by SPEL III to set up and execute communications between robots. These SPEL III-supported commands are as follows.

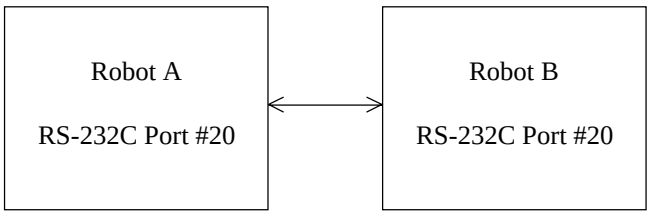
PRINT # : Data output to com port
INPUT # : Data input from com port
LINE INPUT # : Insert one line of input data from the com port as a character string variable
CONFIG : Set configuration of RS-232C port
LOF( ) : Function for returning the number of lines of received data in the RS-232C buffer
CONSOLE : Console specification when in S.NET mode
STAT( ) : Function for returning the status of the controller that is connected via an RS-232C connection.

Specific use methods for communication-related commands

This section describes four typical examples of communication-related commands.

- Data transfer
- Data input check
- Read status
- Interlock between robots

All four operate in the following environment.



• Data transfer

Use the PRINT and INPUT commands for transfer of numerical data between robots.

<Example 1>
Robot A's output bit (1 or 2) is set to ON according to the numerical value (1 or 2) sent from robot B.

Robot A

```
100  INPUT #20,A
110  IF A=1 THEN ON1
120  IF A=2 THEN ON2
```

Robot B

```
100  PRINT #20,A
```

<Description>
Robot A receives data from robot B and robot A's output bit is set to ON according to the received data value. If robot A executes the INPUT command before it receives data from robot B, robot A will wait at the INPUT command execution step until it receives the data from robot B. Alternatively, robot B can use the PRINT command to send data in advance, before robot A begins its data input operation.

· When using the BASIC protocol, up to 20 messages of data can be sent in advance. If robot B sends more than 20 messages in advance, it waits at the PRINT command step until robot A has taken in the messages via the INPUT command.

· When using XON/XOFF control with TTY protocol, up to 20 messages plus 80 bytes of data can be sent in advance. However, when this amount is exceeded, robot B waits at the PRINT command step until robot A has taken in the messages via the INPUT command.

· When not using XON/XOFF control with TTY protocol, an overflow error occurs when more than 20 messages plus 120 bytes of data is received.

· When using the PRINT or INPUT commands, the number of PRINT command executions must be the same as the number of INPUT command executions. Likewise, the number of numerical data strings processed by the PRINT command must be the same as the number processed by the INPUT command. When the number of data strings do not match, it is handled as an error (error 30).

• Data input check

When the INPUT command is executed, the robot waits at the INPUT execution step until data is received. However, this is not a very efficient way to implement the program. A more efficient method is to use the LOF( ) function to return the amount of received data.

<Example 2>
Use the LOF( ) function to check whether or not data was sent from robot B to robot A. If data was sent, the robot executes the INPUT command to input the data. If no data was sent, it executes other tasks.

Robot A

```
100 IF LOF(20)=0 THEN GOSUB 200
    ELSE GOSUB 1000
110 GOTO 100
200 'PROGRAM1
  :
900 RETURN
  :
1000 INPUT #20,A
  :
1100 RETURN
```

Robot B

```
100 PRINT #20,A
```

<Description>
Before executing the INPUT command, robot A uses the LOF( ) function to check for received data.
If data was received, it executes the INPUT command at line 1000 to insert the data as variable A, then executes the program after line 1000.
If data was not received, robot A executes the subroutine after line 200 and then returns to line 100.

• Read status

When performing operations that use two or more robots, programming one robot's actions may require being able to control the other robot(s)'s actions. In this case, we can use the STAT( ) function to check the status of other robots connected via an RS-232C interface.

When the STAT( ) function is executed, the status information from the target robot is received as 3-byte data. Use the AND command to mask the status bits in the data and extract the information.

Bit	Controller status when bit status is ON
0 to 15	Memory I/O bit 0's status is ON to Memory I/O bit 15's status is ON
16	Status of REMOTE2's start output is ACTIVE
17	Status of REMOTE2's pause output is ACTIVE
18	Status of REMOTE2's error output is ACTIVE
19 to 23	Undefined

<Example 3-1>

Set a conditional branch based on the other robot's status bit (memory I/O bit 7) information.

Robot A

```
100 A=STAT(20) AND &H80
110 IF A<>0 THEN GOTO 1000
 :
1000 'PROGRAM1
```

Robot B

```
100 IF SW(3)=1 THEN ON $7
    ELSE OFF $7
```

<Description>

After robot A receives the status information from robot B, use the AND command to mask the status bits and check the status of memory I/O bit 7. If robot B's memory I/O bit 7 is set to ON, execute the program beginning at line 1000. Alternatively, the status of robot B's input bit number 3 can be used to set robot B's memory I/O status to ON or OFF.

<Example 3-2>

Set a conditional branch based on the other robot's status bit (error output) information.

Robot A

```
100 A=STAT(20) AND &H40000
110 IF A<>0 THEN GOTO 1000
 :
1000 'ERR1
```

Robot B

```
 :
 :
 :
 :
 :
 :
```

<Description>

Robot A checks robot B's error output. If an error has occurred at robot B, execute the error processing program described after line 1000.

• Interlock between robots

When two or more robots share the same operation point data, an interlock is needed to prevent robot arm collisions. The memory I/O can be used to establish such an interlock.

<Example 4>

Robot A and robot B, which share operation point P1, can both operate using the following interlock.

Robot A

```
100 FUNCTION ROBOT1
110 HOME
120 ON #20,$2
130 WAIT SW($1)=1
140 OFF $1
150 JUMP P1;JUMP P2
160 GOTO 120
170 FEND
```

Robot B

```
100 FUNCTION ROBOT2
110 HOME
120 WAIT SW($2)=1
130 OFF $2
140 JUMP P1;JUMP P3
150 ON #20,$1
160 GOTO 120
170 FEND
```

Note: P1 is the operation point shared by robot A and robot B. P2 and P3 are operation points within the safety zone for preventing robot arm collisions.

<Description>

Robot A’s memory I/O bit 1 and robot B’s memory I/O bit 2 are used as flags for interlocking.

When robot A reaches line 120 and after robot B’s memory I/O is set to ON (enabling operation of robot B), robot A waits at line 130 until its own memory I/O is set to ON. After robot B completes its operation at the shared operation point (P1), it jumps to the operation point (P3) that is within the safety zone, then robot A’s memory I/O is set to ON (enabling operation of robot A).

Thus, an interlock can be established by using "ON #m, $n" or "OFF #m, $n" to change the other robot’s memory I/O.

7.3 Communication between Robot and User Equipment

Any system that uses a robot may require transfer of numerical data and/or text data between the user equipment (such as computers or vision systems that use an RS-232C interface) and the robot. This item describes the sending and receiving of numerical data and/or text data between the user equipment and the robot.

Data format

ASCII code is the data format used for each numerical value and character that is transmitted between the user equipment and the robot.

<Examples 1> The ASCII code for "8" is "&H38" and the one for "+" is "&H2B".

NOTE

The received ASCII code is taken directly as the character data, so no particular attention is needed concerning the data format. However, when sending or receiving numerical data, it is important to note the expression format and significant digits. Furthermore, the sending side must send numerical data that is within the range that can be verified by the receiving side. The robot's output format for sending data and its input format for receiving data are both predetermined, so numerical data sent to or received from the user equipment must be within the range provided by these formats.

<Example 2>

	Sending side			Receiving side		
Character data transfer:	A	→	&H41	&H41	→	A
Numerical data transfer:	-12.345E-1	→	ASCII data	ASCII data	→	?

During transfer of numerical data, the sending side converts the numerical data to ASCII data and then sends the data to the receiving side. The receiving side must then convert the ASCII data back into numerical data. However, if the receiving side has received the ASCII data in a format that does not enable accurate conversion to numerical data, the resulting numerical data may differ from the numerical data at the sending side.

The robot's input format and output format are described below.

- Input format : Refer to the input format when sending data from the user equipment to the robot
- Output format : refer to the output format when sending data from the robot to the user equipment.

• Input format

Numerical expressions

The following ranges of numbers can be handled.

Formal numbers: 7 digits

Significant digits: 6 digits

The following expression formats are supported.

<Example>

Input format	Input data
+123.12	123.12
0.12345E1	1.2345
-97.156E-1	-9.7156
-97	-97
.1267	0.1267

Data expressions

The numerals contained in one line of input data are handled as variables according to the following rules.

Except for "+", any character or space that precedes a numeral is ignored.

When there are two or more numerical values, at least one delimiter (*1) must be inserted between each pair of numerical values.

If a numeral is followed by a character such as a letter, all data until the next delimiter will be ignored. When "E" is used to represent an exponent, it is regarded as a numeral.

(*1) Spaces or commas can be used as delimiters.

<Example 2>

Input data	Recognized numerical value
DV-123.56E-3	-123.56×10^{-3}
+123.45E-3 -97.45	123.45×10^{-3} and -97.45
A0=95.62 A1=87.654	0 and 1

• Output format

The robot controller expresses output numerical data according to the following rules.

Integer-type numerical values are output as integers. A space is inserted for positive values.

Real-type numerical values are output as real numbers, but if the number of digits is too large, the output data expression is normalized as follows (assuming that the formal number part exceeds seven digits).

+□.□□□□□□E+□□□

A space is inserted between each set of numerical values when several sets of numerical data are output.

Program for communicating with user equipment

• Communications with TTY protocol

As was described in item 7.1, communications with TTY protocol is a method in which a CR code (&H0D) is added to data as a terminator.

Sending side → Data (text) + CR → Receiving side

<Example>

Content of transmission	Representation of sent data
Send "8" and "7"	(&H38) (&H20) (&H37) (&H0D)
Send "12"	(&H31) (&H32) (&H0D)
Send "-1.3"	(&H2D) (&H31) (&H2E) (&H33) (&H0D)

NOTE

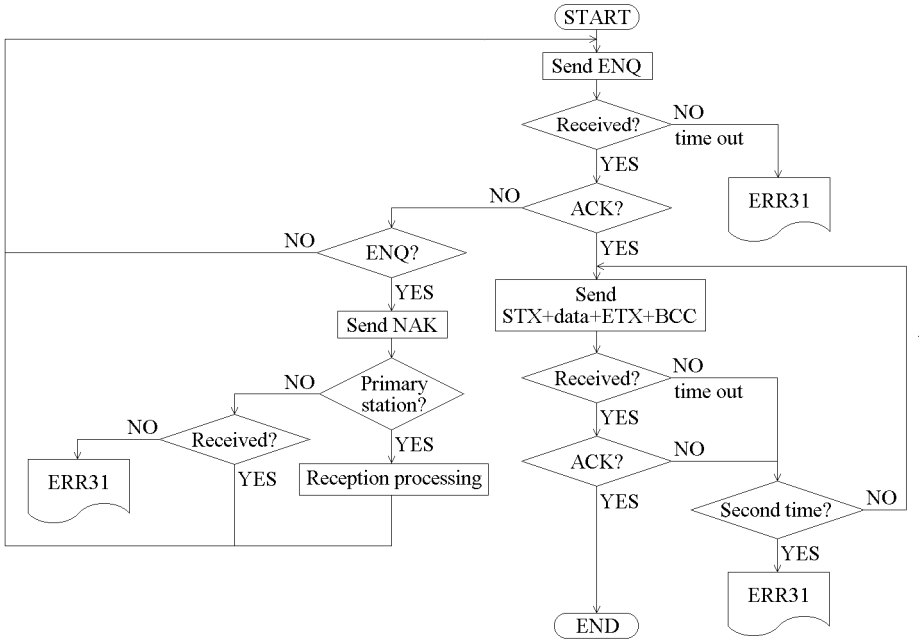
In communication with TTY protocol, there is no error correction method available for when problems such as transmission errors occur, so the BASIC protocol should instead be used whenever possible.

• Communications using BASIC protocol

Communications using BASIC protocol is a more reliable communication method that adds an enquiry code (ENQ) and control characters (STX, ETX, and BCC) to transmission data (see item 7.1).

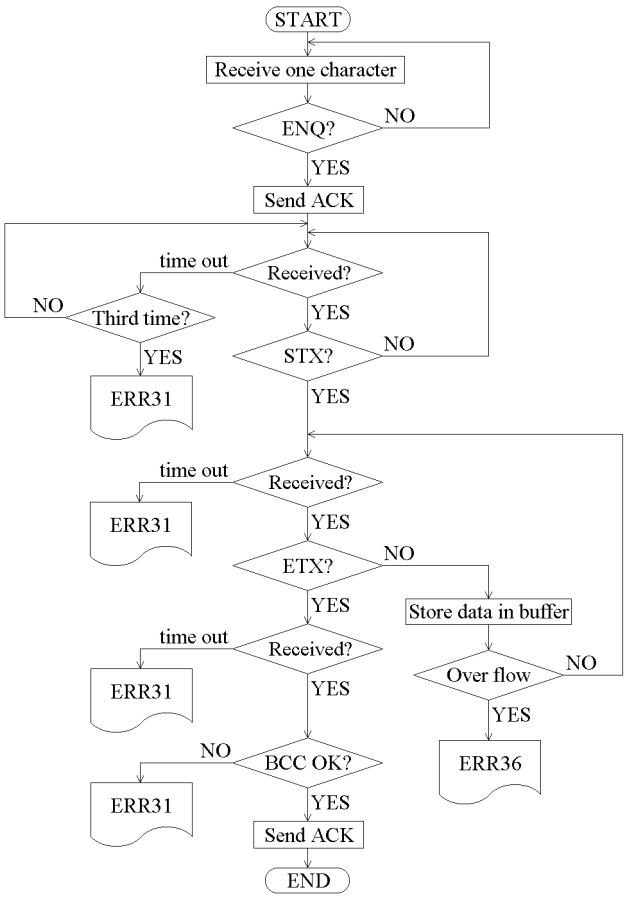
The two flowcharts shown below describe a robot communications program that use this protocol. When transferring numerical data between robots, create a program based on these flowcharts.

Flowchart 1: for sending data via the BASIC protocol



After sending an ENQ code, the sending side waits for an ACK code to be returned. If an ACK is not returned within a specified time period, a timeout error occurs. When the ACK is returned, the sending side sends a string of data surrounded by STX and ETX codes and followed by a BCC code. Finally, the sending side accepts the ACK and ends the transmission (normal end). When the ACK is not returned, the sending side tries three times to send the string of data surrounded by STX and ETX codes.
If, after sending an ENQ, an ENQ is returned instead of an ACK, a transmission data collision has occurred. In this case, the sending side sends a NAK and then, if it is the primary station, it sends another ENQ after the NAK is received. If it is the secondary station, it executes receive processing once, receives all of the transmission data from the primary station, and then sends another ENQ.

Flowchart 2: for receiving data via the BASIC protocol



First, the receiving side waits for the ENQ code, and when it receives the ENQ, it sends an ACK. Next, the receiving side receives data surrounded by STX and ETX codes and uses the BCC checksum code to check for errors. If no errors are detected, it sends an ACK and ends the operation. If the BCC checksum code indicates an error, it sends a NAK and waits for another string of data surrounded by STX and ETX codes to be sent.
If no data is sent within a specified period of time, a timeout error occurs. If the ETX code is not received, the timeout error or the buffer overflow error occurs.

• Example of BASIC program

The following is a communications program that was written in the BASIC programming language and uses the BASIC protocol. To send data, this program sets the transmission data to the character string "SEND$" and calls a transmission subroutine that starts at line 2000.
To receive data, it calls a reception subroutine that starts at line 3000 and sets the received data to the character string "RCVCHR$".

<Example of communications program (written in BASIC)>

```
5 GOTO 5000
10 '***********************************
20 ' Basic Data Transmission Procedure
30 '***********************************
40 '
1000 '<Subroutine #1>:  Initialize Procedure
1010 ' ASCII control character definition
1020 SOH$=CHR$(1):DC3$=CHR$(&H13):ENQ$=CHR$(5):STX$=CHR$(2):ETX$=CHR$(3)
1030 ACK$=CHR$(6):NAK$=CHR$(&H15):ESC$=CHR$(&H1B):CR$=CHR$(&HD):LF$=CHR$(&HA)
1040 '
1050 OPEN "O",#1,"COM0:(E7E1F)"      :'Open RS-232C 9600 bps, 7 bit
1060 OPEN "I",#2,"COM0"              :'    even parity, 1 stop bit
1070 RETURN
1080 '
1090 '
2000 '<Subroutine #2>:    Transmit 1 flame subroutine
2010 '     Input  SEND$...1 flame data
2020 '     Output none
2030 '
2040 PRINT #1,ENQ$; : ANS$=INPUT$(1,#2)
2050 RETRY=0
2060 IF ANS$=ENQ$ THEN PRINT #1,NAK$; : GOSUB 3000 :GOTO 2040
2070 IF ANS$<>ACK$ THEN PRINT "Transmit Error !":END
2080 BCC=0
2090 FOR I=1 TO LEN(SEND$)
2100 BCC=BCC XOR (ASC(MID$(SEND$,I,1)))
2110 NEXT I
2120 BCC=BCC XOR ASC(ETX$)  :BCC$=CHR$(BCC)
2130 PRINT #1,STX$+SEND$+ETX$+BCC$;
2140 ANS$=INPUT$(1,#2)
2150 IF ANS$=ACK$ THEN RETURN
2160 RETRY=RETRY+1 : IF RETRY <=3 THEN GOTO 2080
2170 PRINT "Transmit error!" : END
2180 '
2190 '
3000 '<Subroutine #3>:     Receive 1 flame from RS232C
3010 '    Input none
3020 '    Output RCV$...1 flame data received
3030 '
3040 RCVCHR$=INPUT$(1,#2)
3050 IF RCVCHR$<>ENQ$ THEN GOTO 3040
3060 PRINT #1,ACK$;
3070 RCVCHR$=INPUT$(1,#2)
3080 IF RCVCHR$=ENQ$ THEN GOTO 3060
3090 IF RCVCHR$<>STX$ THEN PRINT "Recieve error !": END
3100 BCC=0:RCV$=""
3110 RCVCHR$=INPUT$(1,#2) : BCC=BCC XOR ASC(RCVCHR$)
3120 IF RCVCHR$<>ETX$ THEN RCV$=RCV$+RCVCHR$:GOTO 3110
3130 RCVCHR$=INPUT$(1,#2) : IF ASC(RCVCHR$)=BCC THEN PRINT #1,ACK$;:RETURN ELSE GOTO 3070
3140 '
3150 '
5000 '*************************************
5010 ' Test Program for Communication
5020 '*************************************
5030 '   << Initialize >>
5040 GOSUB 1000 :SEND$=""
5050 '    << If data arrived, read 1 line and print out. >>
5060 IF LOC(2)<>0 THEN GOSUB 3000:PRINT RCV$:GOTO 5060
5070 '    << Scan keyboard and if key in data exists, send 1 line. >>
5080 A$=INKEY$:IF A$="" THEN GOTO 5060
5090 PRINT A$; :SEND$=SEND$+A$
5100 A$=INKEY$:IF A$="" THEN GOTO 5100 ELSE PRINT A$;
5110 IF A$=CR$ THEN PRINT LF$;:GOSUB 2000:SEND$="":GOTO 5060
```

7.4 Communication between Host Computer and Robot

When configuring a system that includes one or more robots, the robot may need to communicate (i.e., transfer numerical data) not only with user equipment as described in the previous item, but also with a computer in the system that functions as a host computer controlling robot operations such as the following.

- Activating and shutting down the robot
- Transferring programs
- Reading the robot's status, etc.

This item describes how to create a program that enables a computer to function as a host computer controlling a robot.

Console

To enable a computer to function as a host computer controlling a robot, the computer must be specified as a console (a main device for input and output) in AUTO mode. In this case, "console" simply means a device for controlling a robot. Refer to "1.4 Mode" in introductory section for outline of console.

NOTE

The host computer should be connected to the RS-232C port #20 or #21 of controller, you must assign the port connected to the PC as a console.

The optional expansion RS-232C ports (port Nos. 22 and 23) cannot be used for a console.

The following is a sample procedure in order to assign the RS-232C port (#20 or #21) as a console.

<Example> RS-232C port #20 console assignment procedure

① Use the SPEL Editor or SPEL for Windows to set the configuration of RS-232C ports for the console.

```
>CONFIG #20,2,1,3,0   ' The values after #20 depend on PC.
```

② Specify the console for AUTO mode.

```
>CONSOLE #20
```

③ Connect the host computer to #20 port of RS-232C, set the operation mode of the controller into AUTO, the console is assigned to #20 port of RS-232C. In this case, the mode is called as S. NET.

⚠WARNING

Don't carry out teaching operation from PC connected #20 or #21 port of RS-232C. Because emergency stop switch with PC cable connected to #20 or #21 port of RS-232C does not function as emergency stop. Therefore, to move the robot is very dangerous.
While debugging a program with PC which is connected to RS-232C port #20 or #21, when you move the robot, it is required to have the teaching pendant (TP-320/TP-320J) or OPU-300 which is connected to TEACH port in hand, in order to press the emergency stop switch in case of an emergency.

Difference between S. NET and TEACH mode

As far as controlling robot by sending commands from PC, there is no difference between S. NET mode and TEACH mode.

However, since S. NET mode is designed for AUTO mode, features regarding with safety, for example, power state, pause state and safeguard are different. In S. NET mode, program can not be executed with safeguard open. PAUSE input from REMOTE is valid. Refer to the following table.

	TEACH mode	S. NET
Safeguard input	Can operate robot in low power mode	PAUSE state
Emergency stop switch of PC cable	enable	disenable
PAUSE input from OPU-300	disenable	enable
PAUSE input from REMOTE	disenable	enable

Robot control program

When controlling a robot via a host computer, a communications program must be created to enable transfer of data between the host computer and the robot. The following eight items describe this type of robot control program.

- Reset
- Basic procedure
- Control codes
- Procedure of LIST, PLIST, and DIR commands
- Procedure of SAVE and MERGE commands
- High-speed SAVE procedure
- LOAD procedure
- Jog Feeding Procedure

- Reset

Send the following reset code to reset the controller.

(SOH) (DC3) = (&H01) (&H13)

This method works regardless of the controller's current status. The controller returns the following code after it is reset.

(>) = (&H3E)

When the reset code is sent, it is sent by itself, without any other control characters such as CR, ENQ, STX, or ETX. The reset code is effective only when it is sent from the console.

This reset operation can be used to halt program executions, set output ports and memory I/O to OFF, and initialize SPEED and ACCEL settings, but it cannot be used for error recovery or for clearing emergency stop status. Execute the RESET command if error recovery and/or clearing of emergency stop status is needed. (Refer to "Basic procedure" in next page.)

• Basic procedure

To execute the controller's functions (SPEL III commands), each command must be sent to the controller in ASCII code. The controller translates the ASCII code it receives and executes the corresponding command.

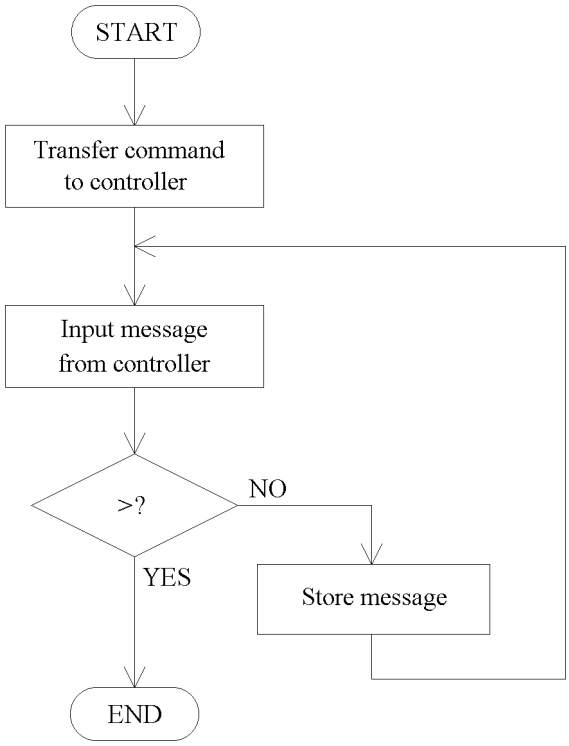
Either TTY (no protocol) or the BASIC protocol can be used, but the latter is recommended because of its higher reliability.

The flowchart 3 shows the basic procedure for communicating control messages.

First, a command is entered via the computer and sent as ASCII code to the controller. The computer then waits for a message from the controller. If the message is ">", the controller returns to waiting for input from the computer. If it is not ">", the computer accepts the message and then waits for further input from the controller.

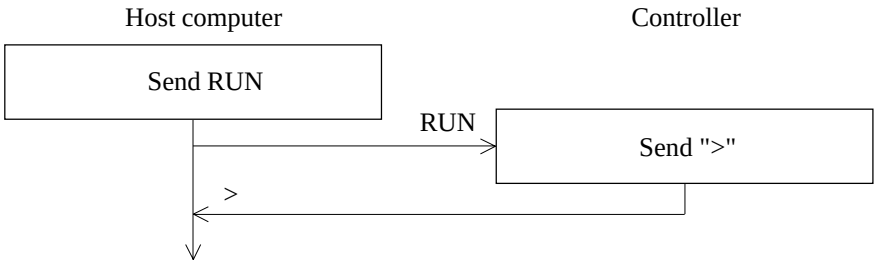
With regard to "Transfer command to controller" in the flowchart, use the sending program described in item 7.3 and use the receiving program from 7.3 for "Input message from controller".

Flowchart 3: The basic procedure for communicating control messages

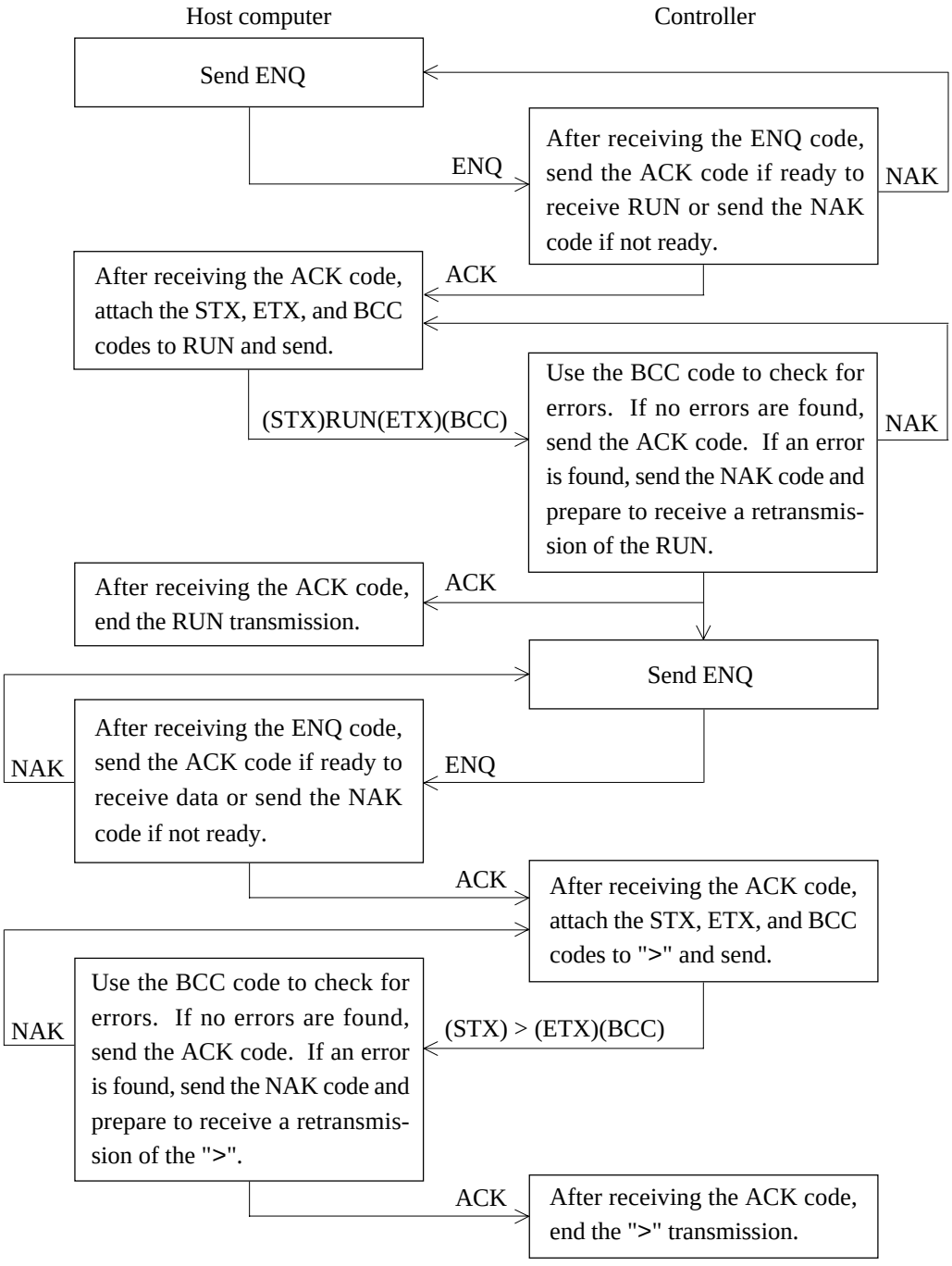


<Example> Sending RUN Command to Robot

TTY protocol (No protocol)



BASIC protocol



• Control codes

Control codes are sent to stop and restart execution of programs. The following control codes must be sent when using either the "TTY protocol" method or the "BASIC protocol" method.

Pause, continue, execute one step, or stop

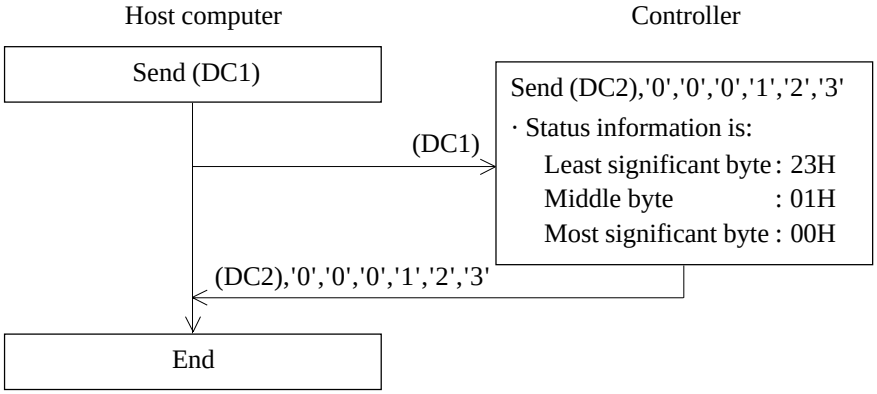
When controlling a program that is being executed, send any of the following control codes.

Control code	Description
(DC3) (ESC)	Pause program
(DC3) (A)	Continue program execution after pause
(DC3) (space)	Execute one step after pause
(DC3) (Q)	Stop program execution after pause

Status transmission request and response

Status transmission request and response corresponds to SPEL III's STAT( ) function. Use the following procedure to request sending of the status to the controller.

① The host computer sends DC1 to the controller.
② When the controller receives the DC1, it sends DC2 followed by a 6-byte hexadecimal version of the 3-byte status data to the host computer.



Change memory I/O

Change memory I/O corresponds to SPEL III's ON #m, $n and OFF #m, $n. Use the following procedure to change the controller's memory I/O.

The host computer sends DC4 (data) to the controller. The data consists of one byte, whose bits have the following meanings.

(data) = (0A1BBBBB)

A : Flag for distinguishing between ON and OFF
ON : A = 1
OFF : A = 0

B : B corresponds to the memory I/O's bit number. The five "B" bits express a numerical value from 0 to 31.

<Example> `ON #20,$15` → (DC4)(6FH)
`OFF #20,$3` → (DC4)(23H)

NOTE You can not use the control code except the control code (00H to 1FH) described in manual and its combination.

• Procedure of LIST, PLIST, and DIR commands

The flowchart 4 shows the procedure of LIST, PLIST, and DIR commands. Execution of these commands can be temporarily paused and restarted or stopped completely. The data to be transmitted must be managed one line at a time. In other words, the host computer uses a handshaking procedure while communicating with the controller.

Send the following codes to enter handshaking mode.

(DC3) (ESC)

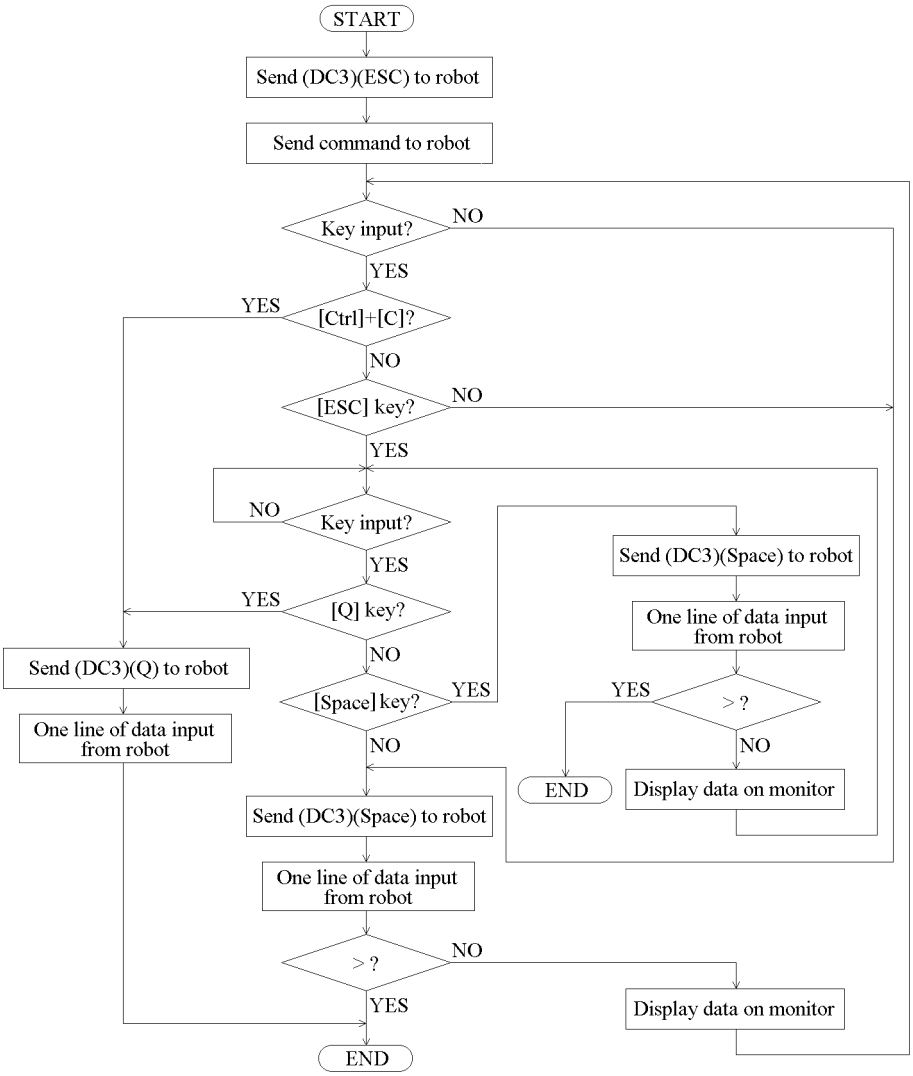
After sending the above codes, send a command (such as LIST).

Then send a space code, after which you will receive one line of the program (or other data) from the robot.

End if the received data is ">". If it is not ">", the received data will be shown on the host computer's monitor.

Before the space code is sent, the program checks for key input. This key input check is for the LIST processing control functions (pause, continue, forced end, etc.).

Flowchart 4: Procedure of LIST, PLIST, and FILES commands



• Procedure of SAVE and MERGE commands (program downloading)

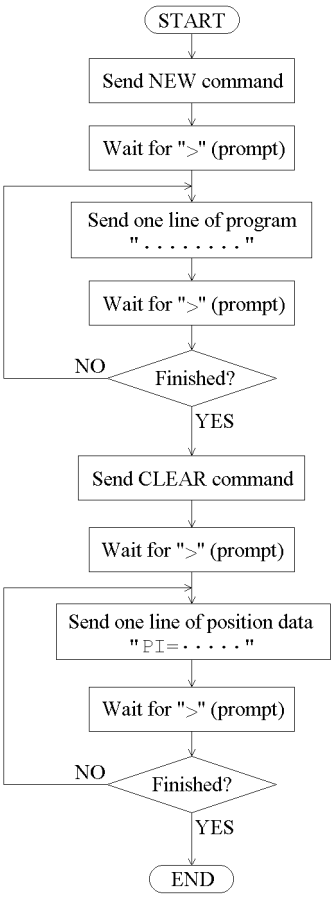
The flowchart 5 shows the procedure of the SAVE and MERGE commands.

In this flowchart, the NEW command is sent first. After the controller's program area is cleared, the program is sent to the controller one line at a time. When the program transfer is completed, the position data transfer is executed.

If transfer of the NEW and CLEAR commands is omitted, MERGE command processing will be executed.

These transfers can use either the TTY-protocol or BASIC protocol method.

Flowchart 5: Procedure of SAVE command



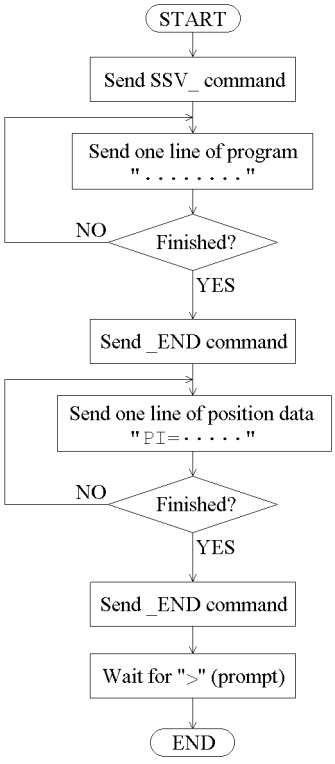
• High-speed SAVE procedure

During SAVE command processing, when one line of the program is sent to the controller, the controller checks the program line number and then must search the program storage area for a storage location for that line, which is a time-consuming operation.
High-speed SAVE processing speeds up program downloading by ensuring that programs are sent in the order of their program line numbers.

The flowchart 6 shows the procedure of high-speed SAVE command.

First, the "SSV_" command is sent, followed by one line of the program.
The underscored part (_) corresponds to ASCII code "&H5F".
When the entire program has been transferred, the "_END" command is sent and the position data is sent line by line, similar to the program data.

Flowchart 6: Procedure of high-speed SAVE command



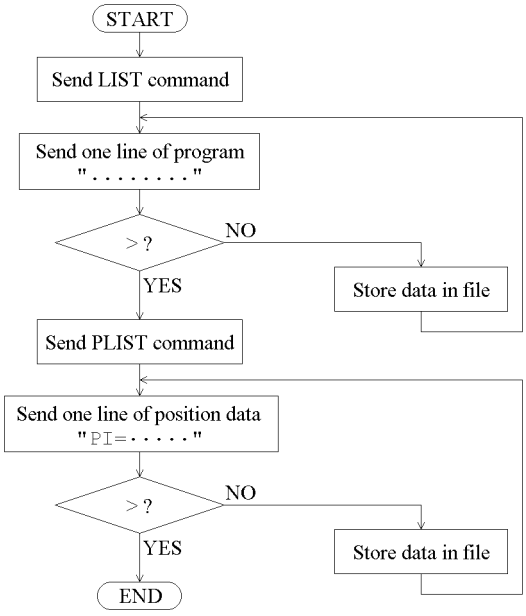
• LOAD procedure (program uploading)

The flowchart 7 shows the procedure for the LOAD command.

The LIST command is sent to the controller, after which the controller receives one line at a time of a program that is stored in the program storage area. The received program is stored in the host computer's program storage area.

Next, the PLIST command is sent to the robot and the position data is processed in the same way as the program data.

Flowchart 7: Procedure of LOAD command



• Jog feeding procedure

The host computer can be used to program the robot to perform "jog feeding" motions.
Jog feeding motions are motions to a position specified via remote control, such as via a computer keyboard. As such, it is an essential teaching function.

The method of jogging the robot is as follows. It is the same method as SPEL Editor is using.

There are two modes for jog feeding motions: motion mode and coordinate system mode. Use both modes in combination.

Motion mode	
StpJog (Step Jog)	Step jog feeding
CntJog (Continuous Jog)	Continuous jog feeding

Coordinate system mode	
Base	Motion according to robot's base coordinates
Joint	Motion defined for each joint
Tool	Motion according to tool coordinates

For further description of jog feeding motion, see "6. Teach Key Mode" of SPEL Editor manual.

Step jog feeding (StpJog mode)

Either BASIC protocol or TTY protocol can be used for communications related to step jog feeding processing.
The step distance can be determined by executing the SEL command to select a step size. However, note that the step direction and distance may vary depending on the selected coordinate system.

Base coordinate system mode

Code	Description	Previous code (*1)
P1J 0	Positive direction along X axis of robot coordinate system	PSX
N1J 0	Negative direction along X axis of robot coordinate system	NGX
P2J 0	Positive direction along Y axis of robot coordinate system	PSY
N2J 0	Negative direction along Y axis of robot coordinate system	NGY
P3J 0	Positive direction along axis #3	PSZ
N3J 0	Negative direction along axis #3	NGZ
P4J 0	Positive direction along axis #4	PSU
N4J 0	Negative direction along axis #4	NGU

(*1) : The previous code is the code used with 42-series controllers. This previous code can also be used for step jog feeding programming according to the base coordinate system.

Joint coordinate system

Code	Description
P1J 1	Positive direction along axis #1
N1J 1	Negative direction along axis #1
P2J 1	Positive direction along axis #2
N2J 1	Negative direction along axis #2
P3J 1	Positive direction along axis #3
N3J 1	Negative direction along axis #3
P4J 1	Positive direction along axis #4
N4J 1	Negative direction along axis #4

Tool coordinate system

Code	Description
P1J 2	Positive direction along x axis in tool coordinate system
N1J 2	Negative direction along x axis in tool coordinate system
P2J 2	Positive direction along y axis in tool coordinate system
N2J 2	Negative direction along y axis in tool coordinate system
P3J 2	Positive direction along axis #3
N3J 2	Negative direction along axis #3
P4J 2	Rotation in positive direction along the u axis and centered on the origin of the tool coordinate system
N4J 2	Rotation in negative direction along the u axis and centered on the origin of the tool coordinate system

Continuous jog feeding (CntJog mode)

For continuous jog feeding processing, send processing codes directly without using a protocol (BASIC or TTY).

Continuous jog feeding processing uses the following two kinds of codes.

Code	Description
Continuous jog start code	Code that is sent to start continuous jog
Jog direction code	Code that determines the direction of jog feeding

The continuous jog start code includes two bytes of data. The send data differs as follows depending on the jog feeding coordinate system.

Jog feeding coordinate system	Code
Base	DC3+’0’
Joint	DC3+’1’
Tool	DC3+’2’

The jog direction codes are shown below.

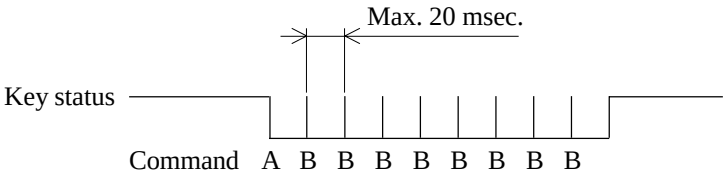
	bit 7							bit 0
X/Y jog directions	0	0	1	SP	-Y	+Y	-X	+X

	bit 7							bit 0
Z/U jog directions	0	1	0	SP	-U	+U	-Z	+Z

- Set 1 for the bit that you need to specify the jog direction
- SP stands for "speed". A bit value of 1 for this sets high speed. Usually, pressing the [Shift] key sets this bit to 1.

<Example> Pressing "+X" and "-Y" sends the code "&H29".

When executing continuous jog feeding processing, first send the continuous jog start code, followed immediately by the jog direction code. Keep sending jog direction codes to maintain continuous jog processing. Jog feeding processing ends if the subsequent jog direction code is not sent within 20 ms. Once jog feeding processing ends, another continuous jog start code must be sent to restart it.



A : Continuous jog start code
B : Jog direction code

7.5 Extended Functions

SPEL Ⅲ: Extended function calls

In addition to the RS-232C related functions explained so far, there are also several extended functions.

• Format of extended functions

Format : EX_CALL (l,m,n)
l : Function code m: Parameter 1 n : Parameter 2

Function code 2 : Terminator output ON/OFF

This control the terminator output when the RS-232C communication protocol is "TTY protocol."

Input parameters

l (function code)	:	2
m(port No.)	:	20 to 23
n (control flags)	:	When n is 0, output terminator When n is 1, do not output terminator

Return value

error code (normally 0)

NOTE

This setting is not backed up. A reset will restore the default value (output terminator), so be sure to include this command in your program.

```
100 B=EX_CALL(2,20,11)      ' Do not output terminator (#20)
110 PRINT #20,123
120 INPUT #20,A             ' Input numerical value from #20
```

Format : INKEY$(n)
n : 20 or 21(port No.)

This function returns one character of the data input to the RS-232C port.
A value of CHR$ (0) is returned if a time-out occurs (the time-out period is about two seconds).

NOTE

This function will not operate correctly unless the specified port is in RAW mode.

• Extended function commands

Format : SETRAW #n

n : 20 or 21(port No.)

NOTE

This sets the specified RS-232C port to RAW mode.

RAW mode

When using the TTY protocol, terminators are not added to data output via the PRINT command. Also, the INPUT command cannot be used for data input, so the INKEY$( ) function is instead used to return one character.

Format : COOKED #n

n : 20 or 21(port No.)

This sets the specified RS-232C port to COOKED mode.

COOKED mode

When using the TTY protocol, terminators are added to data output via the PRINT command. The INPUT command can be used for data input. This is the default setting.

ASCII code chart

	00	10	20	30	40	50	60	70
00	NUL	DLE	SP	0	@	P	`	p
01	SOH	DC1	!	1	A	P	a	q
02	STX	DC2	"	2	B	R	b	r
03	ETX	DC3	#	3	C	S	c	s
04	EOT	DC4	$	4	D	T	d	t
05	ENQ	NAK	%	5	E	U	E	u
06	ACK	SYN	&	6	F	V	f	v
07	BEL	ETB	'	7	G	W	g	w
08	BS	CAN	(	8	H	X	h	x
09	HT	EM	)	9	I	Y	i	y
0A	LF	SUB	*	:	J	Z	j	z
0B	VT	ESC	+	;	K	[	k	{
0C	FF	FS	.	<	L	\	l	|
0D	CR	GS	-	=	M	]	m	}
0E	SO	RS	.	>	N	^	n	~
0F	SI	US	/	?	O	_	o	DEL

7.6 Transmission Errors

Transmission error codes

The error numbers below indicate the following transmission errors.

Error No.	Description
30	Number of received data and that of variable for INPUT is not equal
31	Unable to establish communications with device connected via RS-232C interface or TEACH connector
33	Data without terminator received via RS-232C and buffer overflow
34	Parity, overrun, or framing error in RS-232C communications
36	Input from RS-232C exceeds 80 characters

Causes and countermeasures

Error 31	This occurs when the power is off or when the configuration is incorrect.
Error 30 or 36	This is usually caused by a program input error.
Error 33	This error occurs when large amounts of data are being transmitted.
Error 34	This error occurs when the transmission is affected by line noise or other interference. Determine the cause of the interference (such as an extra-long cable or a crimped cable, or a nearby noise-generating device) then retry the transmission.

When a transmission error occurs, use the communications checking flowchart shown in next page to determine the cause.

Flowchart: Communications error checking