Cat. No. W206-E1-04

SYSMAC
CV500-BSC11/21/31/41/51/61

BASIC Units

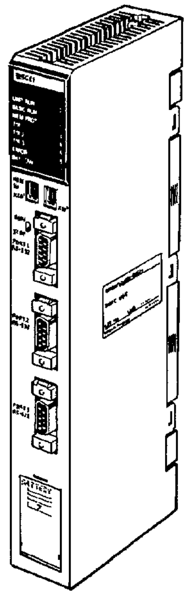
OPERATION MANUAL

OMRON

CV500-BSC11/21/31/41/51/61 BASIC Units

Operation Manual

Revised August 2003



Notice:

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

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

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

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

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

OMRON Product References

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

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

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

Visual Aids

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

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

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

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

TABLE OF CONTENTS

TABLE OF CONTENTS

About this Manual:

This manual describes the installation and operation of the BASIC Unit and includes the sections described below. The BASIC Unit is a CPU Bus Unit that connects to the CPU bus of a SYSMAC CV-series Programmable Controllers. This Unit can be mounted to the CV500, CV1000, CV2000, or CVM1.

Note that this manual is not meant to be a substitute for a manual on BASIC programming. We suggest that you read a manual on BASIC programming before attempting to operate the BASIC Unit.

Please read this manual completely and be sure you understand the information provide before attempting to install and operate the BASIC Unit.

Section 1 provides an introduction to the BASIC Units and describes the general features of the Units. The system, hardware, and memory configurations are also provided.

Section 2 provides the basic steps to install a BASIC Unit and initiate operation for the first time. It also explains the methods that can be used to start and stop program execution in the BASIC Unit.

Section 3 provides information relating to the memory areas of the BASIC Unit. The memory switch settings and specifications are also provided for the proper operation of the Unit.

Section 4 provides an overview of BASIC programming and is not meant to provide a comprehensive explanation of BASIC programming.

Section 5 provides information on data management and operations for the BASIC Units.

Section 6 advances further into BASIC programming and provides information on interrupts, multitasking, and machine language for the purposes of advanced programming.

Section 7 information relating to the use and programming for the peripheral devices. The GB-IB Interface programming is also provided for use with the peripherals.

Section 8 provides the error messages and indications required for troubleshooting as well as general maintenance procedures for the BASIC Unit.

Appendix A provides the standard models of the BASIC Unit and its supporting options/peripherals.

Appendix B provides the specifications of the Unit.

Appendix C provides information on hardware interface connection and assembly.

Appendix D provides various programming examples for the BASIC Unit.

Appendix E provides a list of BASIC instructions.

Appendix F provides a description of machine language commands.

Appendix G provides a list of reserved words.

Appendix H provides information on controlling RS-232C communication lines.

Appendix I provides information on programming with Windows 95 HyperTerminal.

Appendix J provides information on setting memory switches.

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

PRECAUTIONS

This section provides general precautions for using the Programmable Controller (PC) and the BASIC Units.

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

1 Intended Audience

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

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

2 General Precautions

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

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

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

This manual provides information for programming and operating the BASIC Units. Be sure to read this manual before attempting to use the software and keep this manual close at hand for reference during operation.

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

3 Safety Precautions

⚠ **WARNING** Do not attempt to take any Unit apart while the power is being supplied. Doing so may result in electric shock.

⚠ **WARNING** Do not touch any of the terminals or terminal blocks while the power is being supplied. Doing so may result in electric shock.

⚠ **WARNING** Do not attempt to disassemble, repair, or modify any Units. Any attempt to do so may result in malfunction, fire, or electric shock.

⚠ **WARNING** Provide safety measures in external circuits (i.e., not in the Programmable Controller), including the following items, to ensure safety in the system if an abnormality occurs due to malfunction of the PC or another external factor affecting the PC operation. Not doing so may result in serious accidents.

- Emergency stop circuits, interlock circuits, limit circuits, and similar safety measures must be provided in external control circuits.
- The PC will turn OFF all outputs when its self-diagnosis function detects any error or when a severe failure alarm (FALS) instruction is executed. As a countermeasure for such errors, external safety measures must be provided to ensure safety in the system.

- The PC outputs may remain ON or OFF due to deposition or burning of the output relays or destruction of the output transistors. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.
- When the 24-V DC output (service power supply to the PC) is overloaded or short–circuited, the voltage may drop and result in the outputs being turned OFF. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.

4 Operating Environment Precautions

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

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

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

- Locations subject to static electricity or other forms of noise.
- Locations subject to strong electromagnetic fields.
- Locations subject to possible exposure to radioactivity.
- Locations close to power supplies.

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

5 Application Precautions

Observe the following precautions when using the PC system.

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

- Always ground the system to 100 Ω or less when installing the Units. Not connecting to a ground of 100 Ω or less may result in electric shock.
- Always turn OFF the power supply to the PC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting Power Supply Units, I/O Units, CPU Units, Memory Units, or any other Units.
  - Assembling the Units.
  - Setting DIP switches or rotary switches.
  - Connecting cables or wiring the system.
  - Connecting or disconnecting the connectors.

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

- Fail-safe measures must be taken by the customer to ensure safety in the event of incorrect, missing, or abnormal signals caused by broken signal lines, momentary power interruptions, or other causes.
- Interlock circuits, limit circuits, and similar safety measures in external circuits (i.e., not in the Programmable Controller) must be provided by the customer.
- Always use the power supply voltages specified in this manual. An incorrect voltage may result in malfunction or burning.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied. Be particularly careful in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.
- Do not apply voltages to the Input Units in excess of the rated input voltage. Excess voltages may result in burning.
- Do not apply voltages or connect loads to the Output Units in excess of the maximum switching capacity. Excess voltage or loads may result in burning.
- Disconnect the functional ground terminal when performing withstand voltage tests. Not disconnecting the functional ground terminal may result in burning.
- Be sure that all the mounting screws, terminal screws, and cable connector screws are tightened to the torque specified in this manual. Incorrect tightening torque may result in malfunction.
- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction if foreign matter enters the Unit.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Double-check all wiring and switch settings before turning ON the power supply. Incorrect wiring may result in burning.
- Wire correctly. Incorrect wiring may result in burning.
- Mount Units only after checking terminal blocks and connectors completely.
- Be sure that the terminal blocks, Memory Units, expansion cables, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.
  - Changing the operating mode of the PC.
  - Force-setting/force-resetting any bit in memory.
  - Changing the present value of any word or any set value in memory.
- Resume operation only after transferring to the new CPU Unit the contents of the DM Area, HR Area, and other data required for resuming operation. Not doing so may result in an unexpected operation.
- Do not pull on the cables or bend the cables beyond their natural limit. Doing either of these may break the cables.
- Do not place objects on top of the cables or other wiring lines. Doing so may break the cables.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals. Connection of bare stranded wires may result in burning.

- When replacing parts, be sure to confirm that the rating of a new part is correct. Not doing so may result in malfunction or burning.
- Before touching a Unit, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.

SECTION 1
Introduction

This section provides an introduction to the BASIC Units and describes the general features of the Units. The system, hardware, and memory configurations are also provided.

1-1 Features

Interfaces

Choose from three different sets of interfaces to connect to the peripheral devices required by your system.

RS-232C (two) and RS-422 Interfaces

CV500-BSC11 (without EEPROM) or CV500-BSC21 (with EEPROM)

RS-232C (two) and Centronics Interfaces

CV500-BSC31 (without EEPROM) or CV500-BSC41 (with EEPROM)

RS-232C (one) and GP-IB Interfaces

CV500-BSC51 (without EEPROM) or CV500-BSC61 (with EEPROM)

BASIC Programming

The BASIC Units employ a high-speed intermediate executable, interpreter-type BASIC, eliminating the need of compiling operations, so that programming can be carried out easily and quickly. The Program area is divided into three sections, each which can be programmed independently. The program can be developed or edited from a commercially available terminal or computer and then saved to memory cards in the CPU Unit.

Debugging

Program execution can be traced by `TRON` instruction. Program execution can be paused or resumed by `STOP` or `CONT` instructions. Program execution can be stopped at or resumed from a specified line by `BREAK` or `CONT` instructions.

Storage of Variables

Data used in the program (variables) can be stored in memory and protected by battery backup.

Machine Language

Program can be developed and executed in V25 machine language.

Multitasking

Up to 16 tasks can be processed in parallel by executing separate tasks to perform various arithmetic operations, data input/output from/to peripheral devices, and data transfer with the CPU Unit.

Program Control

Program can be started through key input from a terminal or by the snap switch on the front panel. Also, a program can be automatically started on power application or reset.

Data Transfer

Data can be easily transferred back and forth between the BASIC Unit and the PC's CPU Unit. High-speed data transfer is possible from the BASIC program without any programming in the CPU Unit. You can access data not only in the local CPU Unit, but also in other BASIC Units or in Units located on local or remote networks.

Data transfer can be controlled using one or more of the following methods.

Cyclic: A total of 384 input/output words of data can be transferred when the I/O of the PC is refreshed.

CPU Bus Link: Data can be transferred with the CPU Unit or other CPU Bus Units.

Event: The data in the CPU Unit can be read or data can be written to the CPU Unit by using the instructions of the BASIC Unit even when the program of the CPU Unit is not being executed.

Clock

The BASIC Unit uses the same clock the CPU Unit by transferring the time in the CPU Bus Link Area. The time can be set from the BASIC Unit.

EEPROM

With BASIC Units equipped with EEPROM, the program can be saved to the EEPROM so that the Unit can be operated without a battery (however, variables still require battery backup to be maintained during power interruptions).

16 BASIC Units per PC

Up to 16 BASIC Units can be mounted to the CPU Rack or Expansion CPU Rack. The limit of 16 Units, however, includes all CPU Bus Units mounted to the PC, so fewer BASIC Units will be available if any other CPU Bus Units are used.

The other CPU Bus Units are the SYSMAC LINK Unit, SYSMAC NET Link Unit, and SYSMAC BUS/2 Remote I/O Master Unit.

Network Communications

`PC READ` and `PC WRITE` can be used to transfer data to/from other PCs on the same or interconnected networks; `PRINT` and `INPUT`, to transfer data to/from BASIC Units on other PCs on the same or interconnected networks. The BASIC Unit also supports automatic processing for certain FINS commands transmitted via PC networks.

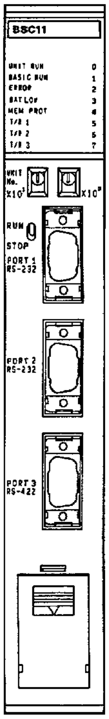
⚠ **Caution** The BASIC Unit is equipped with a hardware test program that is used for inspection and maintenance. When this program is executed, the entire program area will be initialized. This program is not intended for customer use. Do not set the unit number to 99, turn ON pin 2 of the front-panel DIP switch, and restart the BASIC Unit or turn power on.

1-2 System Configuration

Models

Models with three different sets of interfaces are available, each of which is available with or without EEPROM, making a total of six models of BASIC Units. The appearance of these is shown below.

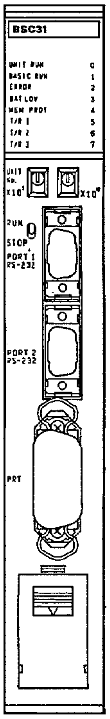
RS-232C (two) and RS-422 Interfaces



CV500-BSC11
(without EEPROM)

CV500-BSC21
(with EEPROM)

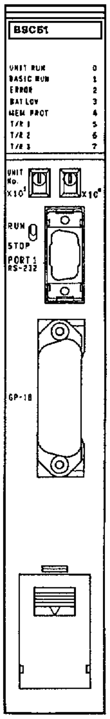
RS-232C (two) and Centronics Interfaces



CV500-BSC31
(without EEPROM)

CV500-BSC41
(with EEPROM)

RS-232C (one) and GP-IB Interfaces



CV500-BSC51
(without EEPROM)

CV500-BSC61
(with EEPROM)

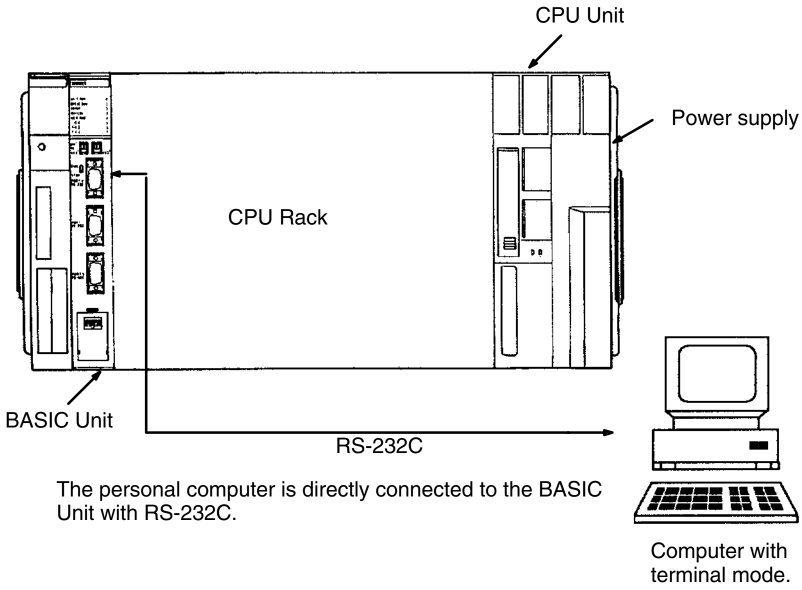
Peripheral Devices

The following peripheral devices can be connected to the BASIC Unit. Note that the peripheral device model that can be connected to the BASIC Unit depends on the BASIC Unit Model.

Interface	BSC11/BSC21	BSC31/BSC41	BSC51/BSC61
Port 1 (RS-232C)	Computer (with terminal mode), display terminal, printer, display	Computer (with terminal mode), display terminal, printer, display	Computer (with terminal mode), display terminal, printer, display
Port 2 (RS-232C)			NA
Port 3 (RS-422)	Host Link Unit (C500-LK203, C500-LK201-V1, C200H-LK202, and C120-LK202-V1) E5AX-A□ Temperature Controller	NA	
Centronics	NA	Printer, display	
GP-IB		NA	Intelligent Signal Processor

Simple System Configuration

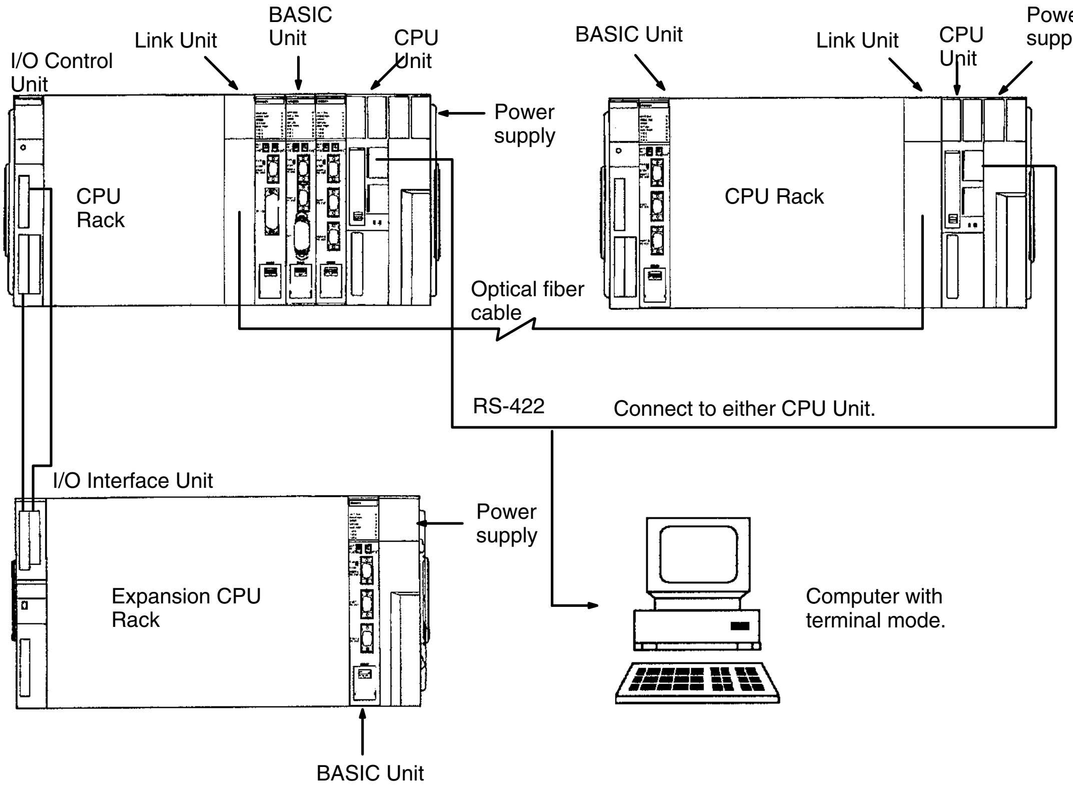
Following is an example of a simple system configuration where only one BASIC Unit is mounted to the CPU Rack.



The personal computer is directly connected to the BASIC Unit with RS-232C.

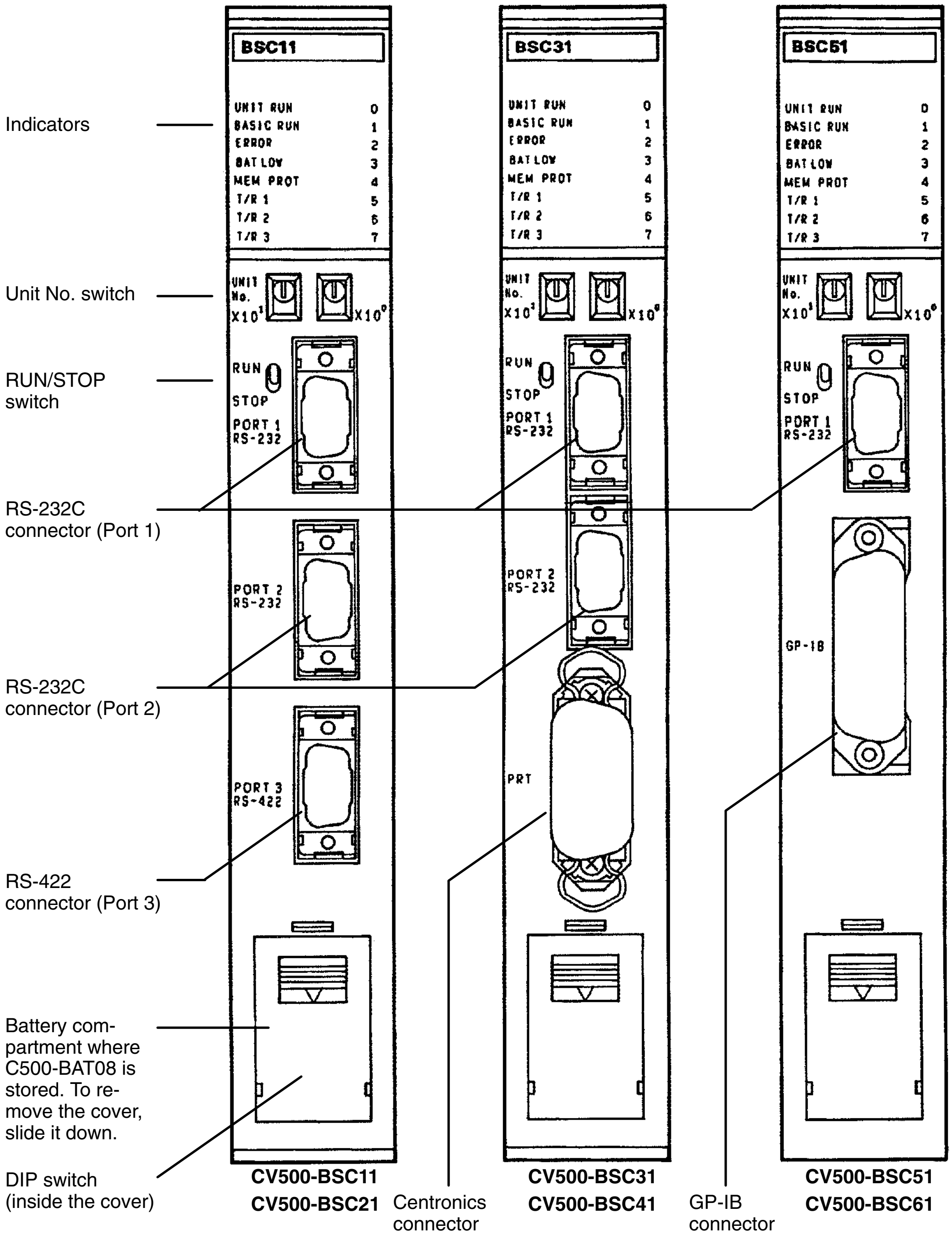
Expanded System Configuration

The system can be expanded by using Link Units to create a network, thus allowing the BASIC Unit to communicate not only with local BASIC Units and the local PC, but also with remote BASIC Units and PCs. The following is an example of such an expanded system. In this system, the computer can be connected to either CPU Unit to access any of the BASIC Units via the optical link between the Link Units and/or the CPU Bus connection to the Expansion CPU Rack.



1-3 Nomenclature and Functions

Front



Ports

RS-232C

Connects a terminal for programming or a display, printer, and bar code reader. The line length is 15 m max.

RS-422

Connects a terminal or peripheral device at a greater distance than for the RS-232C. The total line length is 500 m max.

Centronics

Connects a printer or display.

GP-IB

Connects a GP-IB device, such as an Intelligent Signal Processor.

Indicators

<table>
<tr><th colspan="3">Indicator</th><th colspan="2" rowspan="2">Meaning</th></tr>
<tr><th>Name</th><th>Color</th><th>State</th></tr>
<tr><td rowspan="2">UNIT RUN</td><td rowspan="2">Green</td><td>ON</td><td colspan="2">Lit after the Unit has been initialized.</td></tr>
<tr><td>OFF</td><td colspan="2">Lit when the Unit has been reset by the PC during a power interruption, or when an error has occurred in the Unit (when the watchdog timer operates).</td></tr>
<tr><td rowspan="3">BASIC RUN</td><td rowspan="3">Green</td><td>ON</td><td colspan="2">Lit while the program is executed.</td></tr>
<tr><td>Flashing</td><td colspan="2">Flashes slowly while the program is stopped and can be edited; flashes quickly while the program is executed or while the Unit is waiting for input from a port.</td></tr>
<tr><td>OFF</td><td colspan="2">Goes off when the program is stopped.</td></tr>
<tr><td rowspan="2">ERROR</td><td rowspan="2">Red</td><td>ON</td><td colspan="2">Lit if a significant error (such as user memory check error, area overflow, or executable intermediate code generation error) has occurred while the program is developed or executed.</td></tr>
<tr><td>OFF</td><td colspan="2">Not lit when no error has occurred.</td></tr>
<tr><td rowspan="2">BAT LOW</td><td rowspan="2">Red</td><td>ON</td><td colspan="2">Lit if the supply voltage of the battery has dropped below a specific level.</td></tr>
<tr><td>OFF</td><td colspan="2">Not lit when the battery voltage is at the normal level.</td></tr>
<tr><td rowspan="2">MEM PROT</td><td rowspan="2">Orange</td><td>ON</td><td colspan="2">Lit when the user program area is write-protected.</td></tr>
<tr><td>OFF</td><td colspan="2">Not lit when the user program area is not write-protected.</td></tr>
<tr><td rowspan="2">T/R 1
T/R 2
T/R 3</td><td rowspan="2">Orange</td><td>Flashing</td><td>Flashes while the corresponding port (port 1 to 3) transfers or receives data.</td><td>T/R 3 indicator of the BSC31 and BSC41 does not flash.</td></tr>
<tr><td>OFF</td><td>Not lit when the corresponding port is not exchanging or receiving data.</td><td>T/R 2 and T/R 3 of the BSC51 and BSC61 do not flash.</td></tr>
<tr><td>0 to 7</td><td>Orange</td><td>---</td><td colspan="2">These indicators are turned ON/OFF by the user with system calls.</td></tr>
</table>

UNIT No. Setting Switch

Sets the unit number of the BASIC Unit. Refer to *1-3-1 Switch Settings* for details.

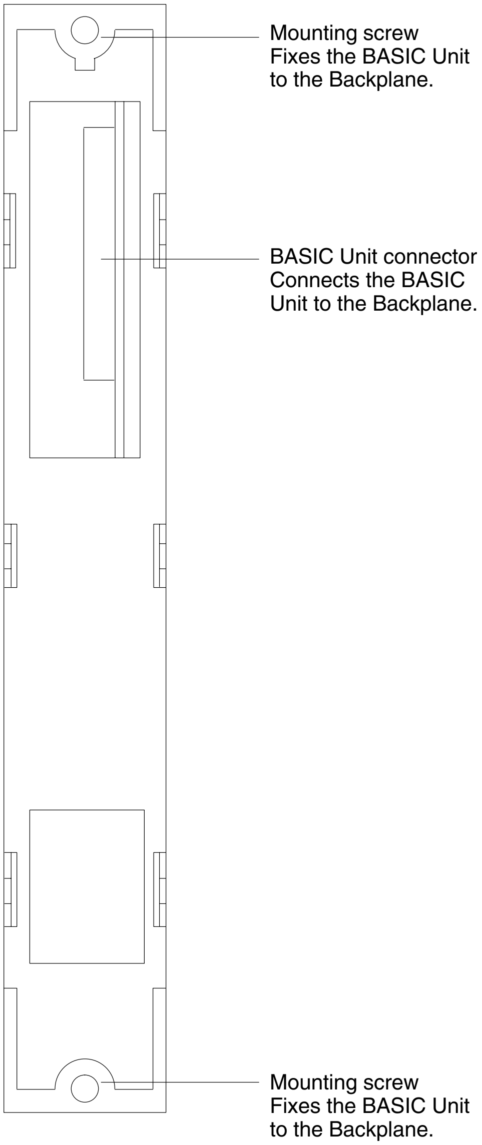
RUN/STOP Switch

Executes or stops the user program. This switch is used in combination with a memory switch set for the BASIC Unit. Refer to *1-3-1 Switch Settings* for details.

DIP Switch

This switch specifies whether the user program memory is write-protected, whether the memory switches are enabled, and whether the termination resistance for RS-422 communications is connected. Refer to *1-3-1 Switch Settings* for details.

Rear View



1-3-1 Switch Settings

The BASIC Unit is provided with three switches: unit number, run/stop, and DIP switches.

Unit Number Switch

This switch specifies the unit number of the BASIC Unit. Set this switch to anywhere between 00 and 15 using a small flat-blade screwdriver. Do not specify a unit number that has already been set for another CPU Bus Unit, i.e., other BASIC Units, SYSMAC LINK Units, SYSMAC NET Link Units, and SYSMAC BUS/2 Remote I/O Master Units.

Run/Stop Switch

Starts or stops the program of the BASIC Unit. This switch is used in combination with a memory switch shown below. The memory switches are contained in the PC and are used to set operating parameters for the BASIC Unit. Refer to *2-2 Memory Switches* for details.

State		Function
RUN/STOP switch	Memory switch	
RUN	Manual start	In this state, the BASIC Unit waits for input of a command after power application or a restart. To start the program, enter RUN from the terminal.
	Automatic start	In this state, the program execution is automatically started when power is turned on or the Unit is restarted.
STOP	Manual start	In this state, the program is not executed even when RUN has been input from the terminal. To execute the program, set the switch to the RUN position, and then input RUN from the terminal.
	Automatic start	The program is not executed in this state. To execute the program, set the switch to the RUN position.

DIP Switch

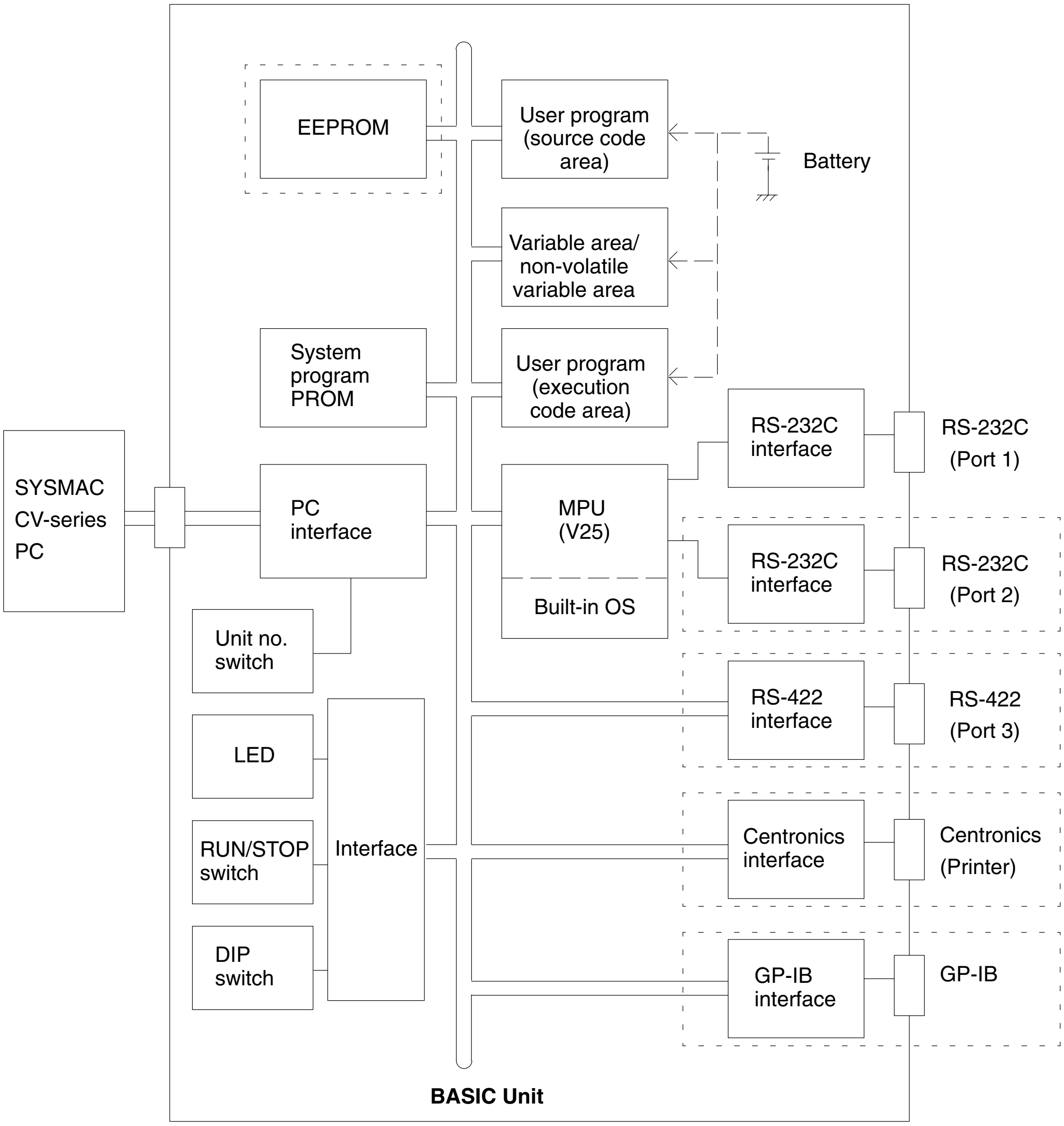
The DIP switch is used as follows:

Pin	Function	State	Operation
1	Memory protect	OFF	Enables the user program area to be written. Set this state when developing, editing, and loading the program.
		ON	Disables writing to the user program area.
2	Memory switch enable	OFF	Enables the current memory switch settings.
		ON	Uses the default memory switch settings regardless of the current memory switch settings. The state of the memory switches, however, can still be changed.* Used when a terminal cannot be connected because of incorrect memory switch settings.
3	---	---	Not used
4	Termination resistance	OFF	Disconnects the termination resistance of RS-422.
		ON	Connects the termination resistance of RS-422. Turn this pin ON when the BASIC Unit is connected as the last devices in a RS-422 communications line.

⚠ **Caution** *Pin 2 of the DIP switch is also used to start the hardware test program, which is used for inspection before shipment. When setting this pin to the ON position, make sure that a correct Unit No. (00 to 15) has been set on the unit umber switches. If the hardware test program is executed, the user program may be erased.

1-3-2 Hardware Configuration

Block Diagram



Note Sections in dotted boxes depend on the model of the BASIC Unit as shown in the following table.

Model	EEPROM	Port 1	Port 2	Port 3	Centronics	GP-IB
CV500-BSC11	---	Yes	Yes	Yes	---	---
CV500-BSC21	Yes	Yes	Yes	Yes	---	---
CV500-BSC31	---	Yes	Yes	---	Yes	---
CV500-BSC41	Yes	Yes	Yes	---	Yes	---
CV500-BSC51	---	Yes	---	---	---	Yes
CV500-BSC61	Yes	Yes	---	---	---	Yes

1-3-3 Memory Configuration

The user memory area of the BASIC Unit consists of the following areas:

User Program Source Code Area

This area stores the source code of the user program. The machine language program is also stored in this area.

The user program source code area can be divided into three areas in each of which can be stored an independent program. It is not possible to move between these areas during program execution; if moving between programs is necessary, you must write them all in one program area as a single program.

Each program area is given a program number to control which area is active. A memory switch controls which program number is active when power is turned on. The active area can be displayed or changed using the `PGEN` command.

`ROMSAVE`, `ROMLOAD`, `ROMVERIFY`, `W`, and `R` commands are preformed for all program areas. `LOAD`, `SAVE`, and `MERGE` are performed only for the current program area.

Non-volatile Variable and Variable Areas

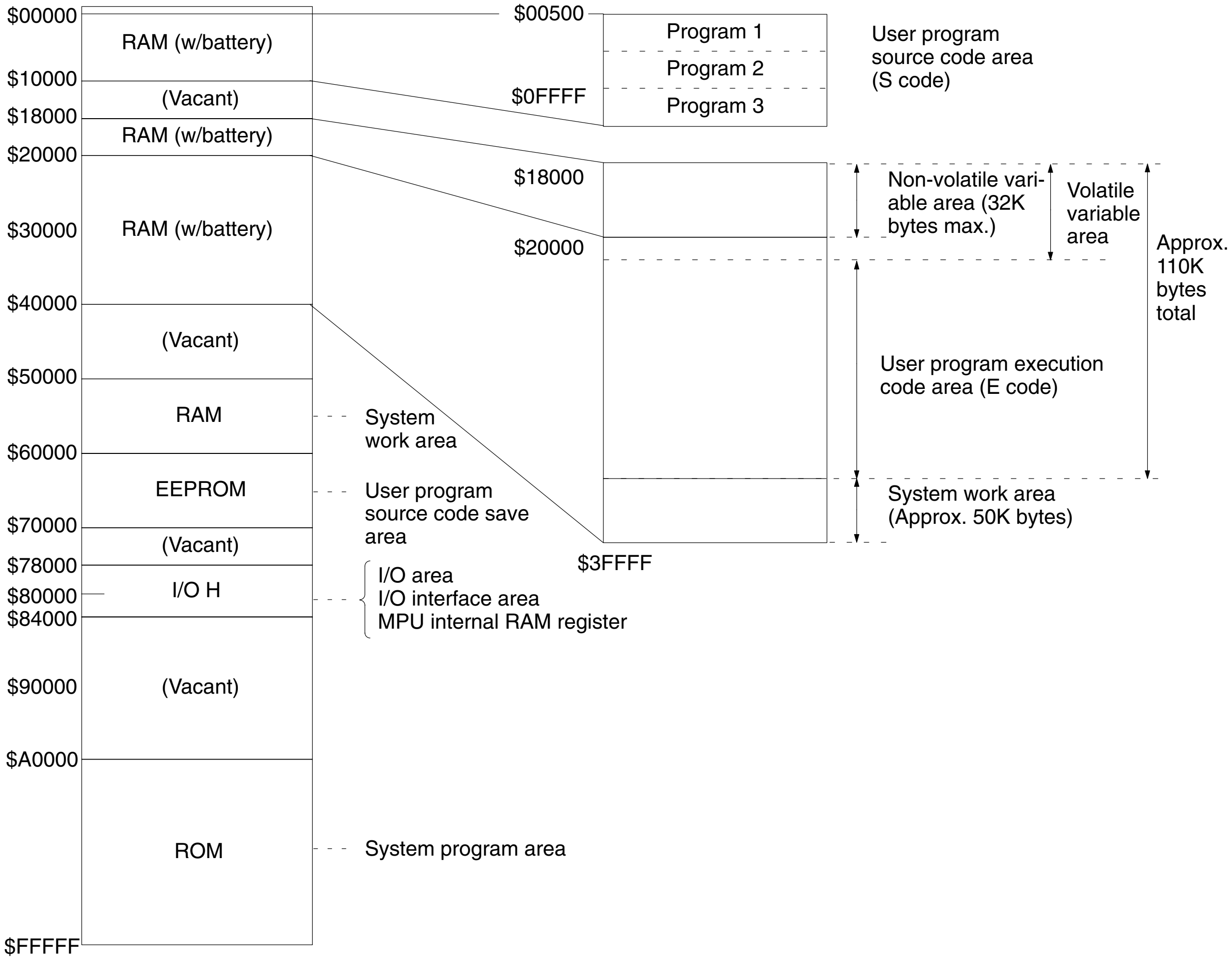
These areas store the variables used in the user program. The variable area and the executable code area are approximately 110K bytes in total. The non-volatile variable area must be within 32K bytes.

Non-volatile variable are preserved when the BASIC Unit is turned on or program execution is restarted. They can be cleared using `OPTION ERASE` or by starting execution with `RUN, ERASE`.

User Program Executable Code Area

When the user program is executed, executable codes are created in this area from the source code and executed.

The memory map of the BASIC Unit is shown below.



1-4 Precautions

Terminals

A terminal or personal computer can be connected to the BASIC Unit and run either in terminal mode (TERM) or via communications software. Terminals must be VT-52, VT-100, or equivalents.

Programming

- Both insert and overwrite programming are available. The writing mode can be set in the memory switches; the default is overwrite.
- Memory cards mounted in the PC's CPU Unit can be treated as files to save BASIC programs and data.
- Programs can be created and edited on any MS-DOS platform and then read into the BASIC Unit. Program files must have a `.BAS` extension.
- The `MERGE` command can be used to join multiple programs into one, but line numbers must be unique.

Program Areas

- Up to three independent programs can be stored in the program areas (S-code areas), but only one of these programs can be executed at a time. You cannot jump between the program areas.
- The current program number is designated in the memory switches and effective when program execution is begun. The `PGEN` command can be used to change the current program number, and the `PINF` command can be used to display it at the left of the monitor screen.
- All three programs areas are saved to, read from, or compared to EEPROM when `ROMSAVE`, `ROMLOAD`, or `ROMVERIFY` is executed. Reads/writes can also be performed to all three program areas regardless of the current program designation.
- Only the current program area is loaded, saved, or merged when `LOAD`, `SAVE`, or `MERGE` are executed for memory cards.

Memory Switches

- Memory switch settings are saved in the PC's CPU Unit in an area separate from the normal PC memory map. The BASIC Unit reads these settings from the PC when started and stores them in a work area for operation. All memory switches are set to all-zeros when the Unit is shipped and must be changed unless the default settings are desired.
- Memory switch settings can be changed in the BASIC Unit's work area in machine language (`MON`). Memory switch settings can be changed in the PC via the `ESW-W` command, or they can be changed via a Programming Device (e.g., CV-series GPC or CVSS) connected to the PC.
- Memory cards can be used to copy memory switch settings from one PC to another.
- The DIP switch on the front of the PC's CPU Unit can be used to return memory switches to their default settings. This can be used if the memory switch settings are unknown to enable connecting a terminal using the default communications parameters.

PC Interface

- PC memory can be accessed from the BASIC Unit even if the PC itself is not programmed.
- Event, cyclic, and CPU bus link processing are available to interface with the PC. Of these, event processing is the most commonly used.
- Event processing allows specific memory areas in the PC to be read or written when necessary.

- Cyclic processing allows specific portions of PC memory to be automatically transferred between the PC and the BASIC Unit. A memory switch is also available to disable cyclic processing to minimize time spent servicing CPU Bus Units.
- CPU bus link processing provides data links between the PC and CPU Bus Units in the CPU Bus Link Area. These links can be used to synchronize processing between CPU Bus Units and the PC. Data link processing does, however, place a load on the PC and is not the only way to synchronize processing. Unless data links are specifically desired, they should be disabled in the PC Setup of the PC.
- The `PC READ` and `PC WRITE` commands can be used to transfer consecutive words to and from the PC. Processing time can be used more effectively by transferring more words with each command rather than splitting the same number of words over multiple commands.
- Only one CPU Bus Unit is serviced each cycle by the PC even if more than one Unit has sent a write request. This can produce delays in executing `PC WRITE`.
- Data can be transferred to and from PCs and BASIC Units on local or remote networks. Transfers to PCs are performed with `PC READ` and `PC WRITE`. Transfers to other BASIC Units are performed with `OPEN` followed by `PRINT` and `INPUT`.

Programming

- Programs are manipulated in S-code (source code) when editing at the terminal or when saving to or loading from EEPROM or memory cards. S-code must be compiled into E-code (execution code) via `RUN` to be executed. Code is compiled automatically when `RUN` is executed and can produce a delay for large programs. If the program is not changed, however, code is complied only once, i.e., the first time `RUN` is executed, increasing execution speed for subsequent `RUN`s.
- Actual execution starts when `RUN` is input, when the RUN/STOP switch is set to RUN, or automatically when the BASIC Unit is turned on and the memory switches are set for automatic program execution. Refer to page 21 for details.
- Memory switches can be set to automatically load, compile, and run a program from a memory card or EEPROM when the BASIC Unit is turned on. Be sure to allow for compiling time when using this method, which also eliminates the need for a backup battery.

Execution

- Character variable length is fixed to 18 characters by default. Garbage collection is not performed. Any changes to variable length must be declared before `PARACT 0` using `OPTION LENGTH`. Errors are not generated when substituting to character variables even if the fixed length is exceeded.
- Non-volatile variables are supported and are backed up by a battery. Data is such variables is maintained during power interruptions and between program executions. Non-volatile variables are cleared when `OPTION ERASE` is executed or when the program is started with `RUN, ERASE`.
- `TRON` and `TROFF` by default display only the status of the current task. Use `TRON ALL` to display the status of all tasks.
- The communications error flags in word n+2 of the cyclic area will be turned ON if a parity, overrun, or framing error occurs during serial data reception. Depending on the type of error, all data up to the character when the error occurred will be lost.

- Interrupts from input commands that are awaiting completion will not return to the input command, but to the line following the input command, i.e., the input command will not be completed. Input command variable substitution will not be performed and data may be left in the input buffer. To see if an input command has not been completed, check `INTRL` (a variable containing the line number of the interrupted command) on the line following the input instruction to see if it contains the line number of the input instruction.
- The send and receive buffers at the RS-232/422 port are 512 bytes respectively.

Multitasking

- Tasks are switched after each command, even for compound lines. Tasks are switched in order of task number to the next task that is ready. Tasks that are busy (e.g., awaiting I/O) are skipped.
- `PARACT N` and `END PARACT` are required to separate tasks. Use `PARACT 0` and `END PARACT` for a single-task program.

Other

- The BASIC Unit does not support a clock, but the clock (RTC) in the PC can be accessed or set from the BASIC Unit.
- The BASIC Unit contains a hardware test program that is used for inspection and maintenance. Executing this program will clear the entire memory area. The hardware test program is executed by setting the unit number to 99, turning ON pin 2 of the front-panel DIP switch, and turning on the power or resetting. This program is not designed for user execution; never executed this program without consulting with qualified service personnel.

SECTION 2
Getting Started

This section provides the basic steps to install a BASIC Unit and initiate operation for the first time. It also explains the methods that can be used to start and stop program execution in the BASIC Unit.

2-1 Installation

This section describes the minimal preparations necessary to set up a BASIC Unit for programming. Refer to *Appendix C Hardware Interfaces* for information on connecting other types of computers or peripheral devices. Refer to the *CV-series PC Installation Guide* for details on general PC installation.

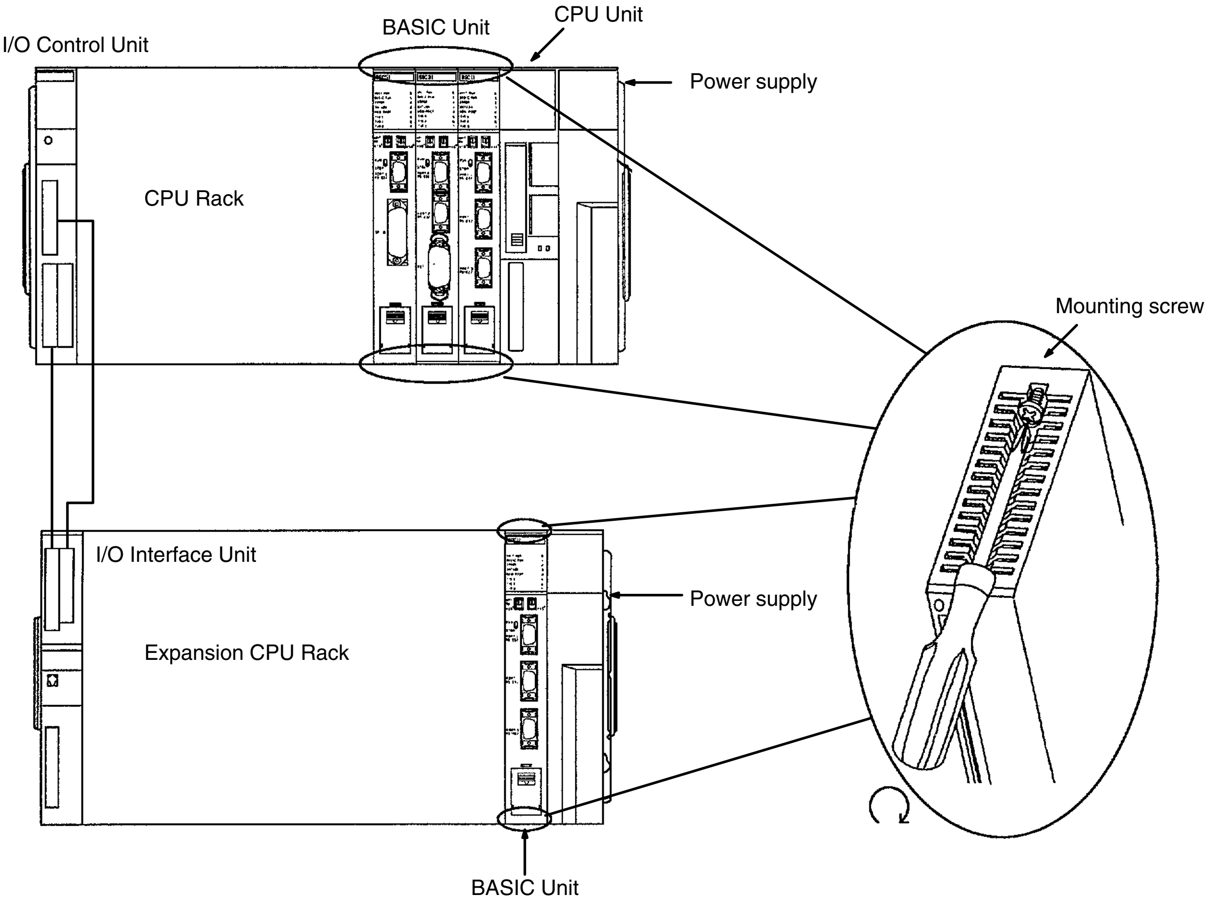
2-1-1 Mounting BASIC Units

A BASIC Unit can be mounted to a CV-series CPU Rack or Expansion CPU Rack. It cannot be mounted to an Expansion I/O Rack.

Up to 16 BASIC Units can be mounted to the CPU Rack and CPU Expansion Rack as long as no other CPU Bus Units are mounted.

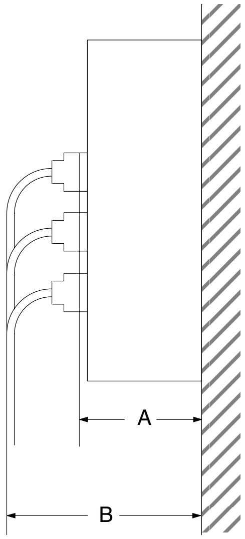
The Unit must be mounted to any of the rightmost 6 slots if the CVM1-BC103 CPU Backplane is used; the rightmost 3 slots if the CVM1-BC053 is used.

Be sure to securely tighten the mounting screws of the BASIC Unit.

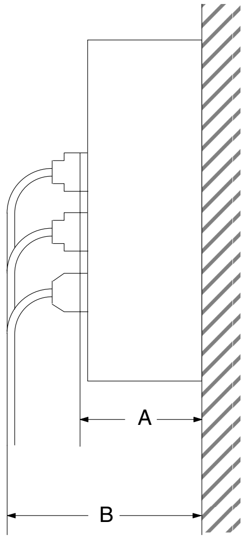


2-1-2 Mounting Dimensions

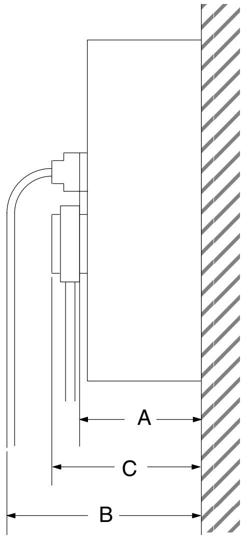
When installing the BASIC Unit in a control box, determine the depth of the control box giving consideration to the connectors to be connected and the height of the cables.



CV500-BSC11/BSC21



CV500-BSC31/BSC41



CV500-BSC51/BSC61

Height (mm)	BSC11/BSC21	BSC31/BSC41	BSC51/BSC61
A: Height from surface of connector to base	103		
B: Connector cover + cable	160 to 180		
C: GP-IB connector cover	---		125

2-2 Switch Settings

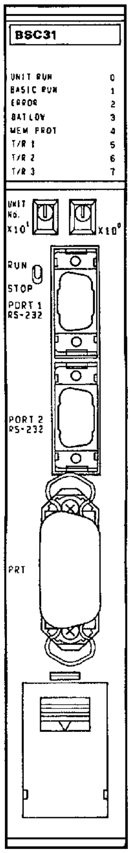
Set the following switches on the BASIC Unit as described below. Details on switch setting are provided in *Section 1 Introduction*.

1, 2, 3...

1. Set a Unit number in the range of 0 to 15. Do not set the same Unit Number as those of the other CPU Bus Units.

2. Set the RUN/STOP switch to the STOP position.

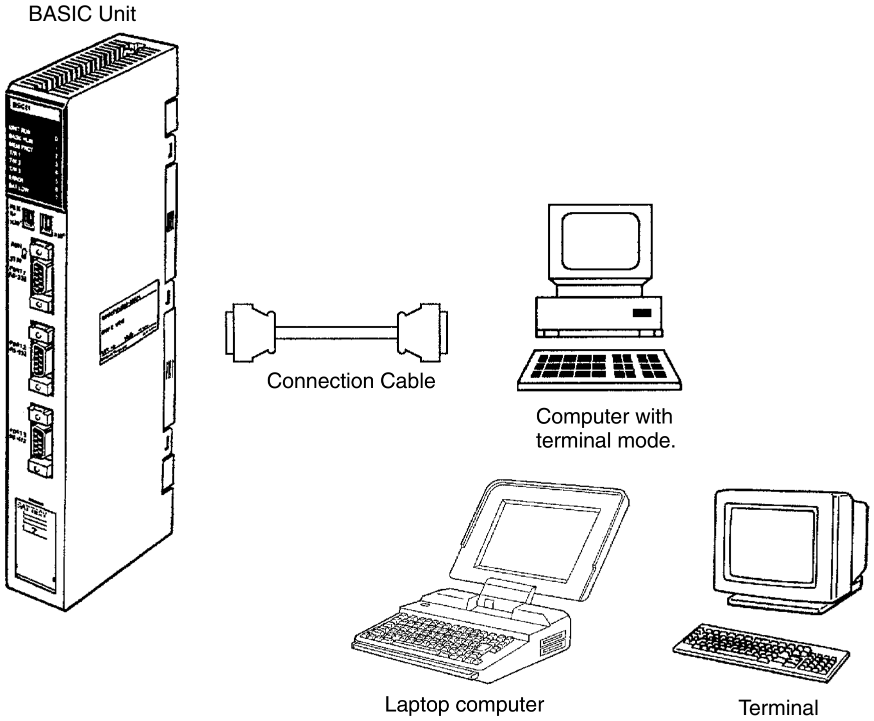
3. Open the battery compartment and set all the switch pins of the DIP switch to the OFF position to get the following settings:



Pin no.	DIP switch setting
1	Memory write-protected (OFF)
2	Memory switches disabled (OFF)
3	Not used (OFF)
4	Termination resistance (OFF)

2-3 Getting the Terminal Ready

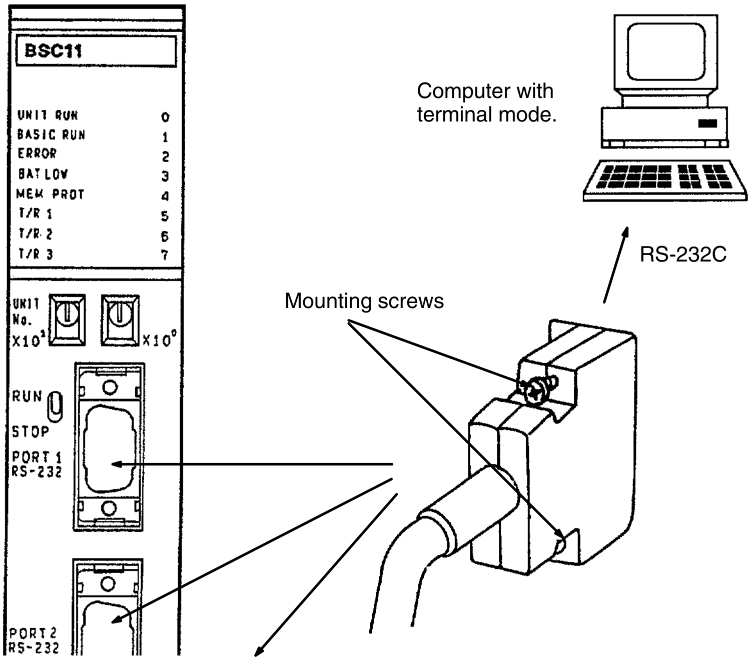
To use the BASIC Unit, the CPU Rack and a terminal for developing programs are necessary. The terminal can be any of those illustrated below. A cable that connects the BASIC Unit and the terminal is also necessary. Use CV500-CN228 as the cable connecting the computer (with terminal mode) and BASIC Unit.



2-4 Connecting the Terminal

Connect the terminal connecting cable to port 1 on the BASIC Unit, and securely tighten the screws of the cable.

The selection of communication ports 1 through 3 used to connect the terminal is specified by the memory switches in the CPU Unit. The default setting is port 1. The terminal is therefore usually connected to port 1. To change the port, refer to *3-3 Memory Switches*.



2-5 Terminal Preparation

First, turn on the power to the terminal. If the power to the PC is turned ON first, the terminal may malfunction.

The defaults of ports 1 through 3 of the BASIC Unit are as follows. Set the communications parameters of the terminal to match these.

Baud rate	9,600 bps
Data length	8 bits
Parity	None
Stop bit	1 bit (or 2 bits)
Others	Full duplex, no echo, XON/XOFF control, no automatic carriage return

2-6 Memory Switches

After setting the terminal, turn ON the power to the PC and start the BASIC Unit. If necessary, change the settings of the memory switches. The memory switches are described in *3-2 Memory Switches*.

Default Settings

If the default values are suited to the application, the memory switch settings do not need to be changed. The default values are as follows:

System Parameters

Manual start	Starts when `RUN` is entered from the terminal
Automatic program transfer	Program is not automatically read from EEPROM or memory card
Program selection 1	Executes program 1
English error messages	Displays error messages in English
Printer selection	Does not print Kanji characters

Communications control using RTS/DTR signals is not possible for the ports set as the terminal and printer ports. To perform communications control using RTS/DTR signals, change the ports set as the terminal and printer ports to ports other than the ones for which RTS/DTR control is to be used. This is done using memory switch 3.

Terminal and Printer Ports The terminal and printer can be connected to the following ports:

BASIC Unit	Terminal	Printer
BSC11/BSC21	Port 1	Port 2
BSC31/BSC41	Port 1	PRT (Centronics)
BSC51/BSC61	Port 1	---

Terminal Specifications

Baud rate	9,600 bps
Number of lines on terminal screen	24 lines
Terminal type	Display Terminal or commercially available terminals with terminal mode
Editing mode	Overwrite mode

Changing Memory Switch Settings

The memory switches can be changed if necessary. After changing the memory switches, power to the PC must be turned OFF once and then back ON again to enable the new settings.

The memory switches can be set from a terminal connected to the BASIC Unit or from a Peripheral Device connected to the CPU Unit. Refer to *3-3 Setting Memory Switches* for details.

2-7 Starting/Stopping Programs

Once a program has been written to the BASIC Unit, it can be started/stopped in any of the following three ways:

- From Terminal

This method is mainly used while the program is being debugged, and the program is started or stopped by the key input from the terminal connected to the BASIC Unit.

- RUN/STOP Switch

This switch is used to debug the program in a system configuration where the terminal is not connected.

- Automatic Starting

This is to automatically start the program on power application or restarting, and is used to start the program after debugging has been completed.

Method	Preparation	Start	Stop
From terminal	Connect terminal. Set RUN/STOP switch to RUN. Set manual start mode in memory switches.	Input `RUN` and carriage return from terminal.	Input `CTRL+X` or `CTRL+C` from terminal.
RUN/STOP switch	Set RUN/STOP switch to STOP. Set automatic start mode in memory switches, and supply power or restart.	Set RUN/STOP switch to RUN.	Set RUN/STOP switch to STOP. Input `CTRL+X` or `CTRL+C` if terminal is connected.
Automatic starting	Set RUN/STOP switch to RUN. Set automatic start mode in memory switches and supply power or restart.	Use terminal mode.	Set RUN/STOP switch to STOP. Input `CTRL+X` or `CTRL+C` if terminal is connected.

Note Execution can be stopped from the keyboard by inputting CTRL-X or CTRL-C. When CTRL-X is input, all execution, including I/O processing, will be aborted immediately and “Quit in ...” will be displayed. `STEP` and `CONT` cannot be used after aborting execution with CTRL-X. When CTRL-C is input, execution is stopped as soon as the current instruction has been executed. If “Break in ...” is displayed, `STEP` and `CONT` can be used. If “Quit in ...” is displayed, `STEP` and `CONT` cannot be used.

SECTION 3
Memory Areas and Operations

This section provides information relating to the memory areas of the BASIC Unit. The memory switch settings and specifications are also provided for the proper operation of the Unit.

3-1 Memory Areas

3-1-1 Cyclic Transfer Areas

Cyclic transfers allow data transfers between the PC's CPU Unit and BASIC Unit to be synchronized with the cyclic servicing of the CPU Unit. The memory words in the CPU Unit that can be allocated for cyclic transfer include those in I/O Memory, the DM Area, and the EM Area.

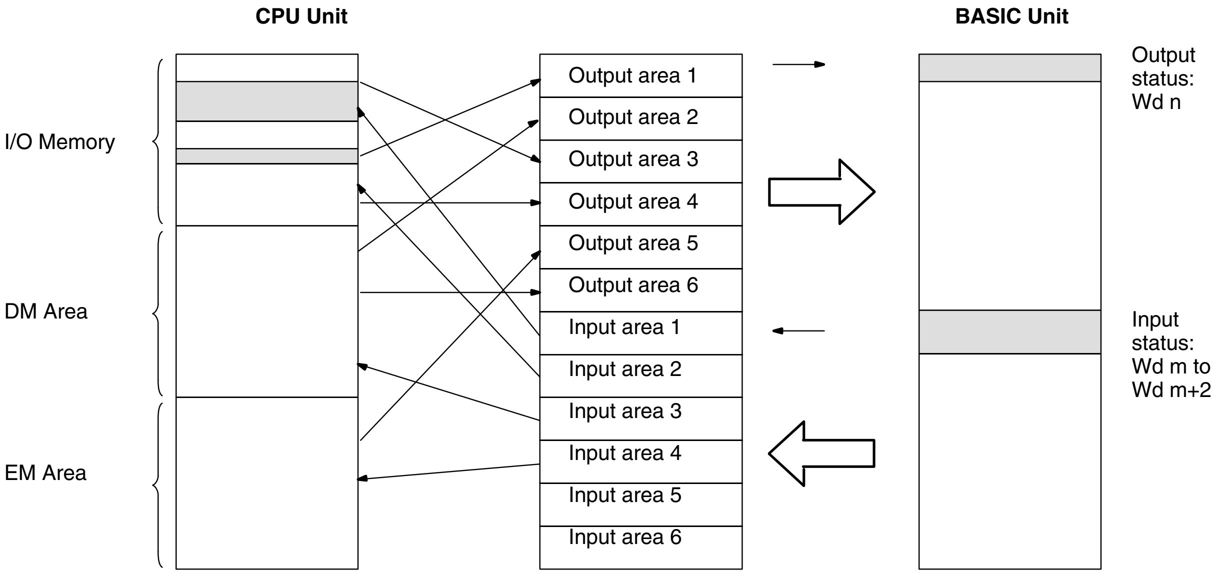
Up to six output areas (CPU Unit to BASIC Unit) and up to six input areas (BASIC Unit to CPU Unit) can be designated. The combined total number of I/O words must be 384 or less in all 12 areas.

Cyclic transfers are set either by default or by using the software switches in the memory of the CPU Unit. For details, refer to *3-3-6 Cyclic Area Settings*. Any words in the I/O Memory (words without prefixes), the DM Area, and EM Area can be set for cyclic transfer. (The EM Area is an option and is available only for the CV1000 and CV2000.)

Data transferred by cyclic transfers to and from the CPU Unit are read and written in the BASIC program using the `PC READ` and `PC WRITE` commands.

The first word of the first output area contains status output from the CPU Unit to the BASIC Unit. This word is designated as word "n." The first 3 words of the first input area contain status input from the BASIC Unit to the CPU Unit. The first of these three words is designated as word "m." The remainder of the first I/O areas and the remaining areas are for user applications.

Example



- When the memory switch is not used to set a specific cyclic area, the following type of allocations are used to receive cyclic data.
  N = 1500 + unit number x 25

Direction	Word	Bit	Name	Remarks
CPU Unit to BASIC Unit	N		System Setup	Data written from the CPU Unit to these words can be read to the BASIC Unit using PC READ ”@SQ....”
	N+1 to N+14		User area	
BASIC Unit to CPU Unit	N+15	00 to 15	Task status display	
	N+16	00	Memory overflow	
		01	Compilation error (error code 000 to 255)	
		02	Compilation error (error code 256 or higher)	
		03	E code error	
		15	Battery error	
	N+17	00 to 07	Error code	
		08	Execution error	
		09	Port 1 error	
		10	Port 2 error	
		11	Port 3 error	
		15	BASIC RUN	
	N+18 to N+24		User area	Data written using PC WRITE ”SI...” from the BASIC Unit is output here.

The memory switch can be used to change the cyclic area allocations.

3-1-2 Reading/Writing to the Cyclic Area Using PC READ/PC WRITE

The method for reading from or writing to the cyclic area in the CPU Unit using the PC READ and PC WRITE instructions in the BASIC program of the BASIC Unit is described here.
Input the following codes as the subcommands to specify the cyclic area using the PC READ and PC WRITE instructions in the BASIC program. (Refer to the *SYSMAC BASIC Units Reference Manual* (W207–E1–2) for information on the format of the PC READ and PC WRITE instructions.)

Instruction	Subcommand	Area	First transfer word	Number of transfer words
PC READ	@SQ	Cyclic output area (Direction: CPU Unit to BASIC Unit)	0 to (maximum number of words –1)	1 to maximum number of words
PC WRITE	@SI	Cyclic input area (Direction: BASIC Unit to CPU Unit)	3 to (maximum number of words –1)	1 to maximum number of words

Example

This example is for a cyclic area as follows:
Output area (CPU Unit to BASIC Unit): DM 12000 to DM 12009
Input area (CPU Unit to BASIC Unit): DM 12010 to DM 12019

- The memory switch setting to make the above areas cyclic areas is as follows:

ESW6-1 = 0082–2000–0001–0010
ESW6-7 = 0082–2010–0001–0010

No. of words

Upper address

Lower address

Area type (0082: DM Area)

Note All the values are set in decimal.

- The first transfer word for the cyclic area is specified as follows:

First transfer word using the PC READ instruction	Address in CPU Unit		First transfer word using the PC WRITE instruction	Address in CPU Unit
0	DM12000		0	DM12010
1	DM12001		1	DM12011
2	DM12002		2	DM12012
3	DM12003		3	DM12013
4	DM12004		4	DM12014
5	DM12005		5	DM12015
6	DM12006		6	DM12016
7	DM12007		7	DM12017
8	DM12008		8	DM12018
9	DM12009		9	DM12019

The shaded areas indicate addresses that are used to display status information, thus not allowing them to be used for user data.

Example

This example shows reading the contents of 10 words from DM 12001 to DM 12009 in the CPU Unit to the BASIC Unit and storing in the variables H, I, J, K, L, M, N, O and P.

```
PC READ  ”@SQ,1,9,9H4“;H,I,J,K,L,M,N,O,P
```

9 words

First word (DM 12001) onwards

Example

This example shows writing values from the variables A, B, C, D, E, F, and G in the BASIC Unit to 7 words from DM 12013 to DM 12019 in the CPU Unit.

```
PC WRITE ”@SI,3,7,7H4“;A,B,C,D,E,F,G
```

7 words

Third word (DM 12013) onwards

Output Status Word (CPU Unit to BASIC Unit)

Word n is the first word of the first output area allocated to the BASIC Unit.

m = 1500 + unit number x 25

Word	Bit	Name	Function
n	00 to 14	---	The contents of the first memory switch word set in the CPU Unit.
	15	System reserved bit	Cannot be used by user

The words from word m+1 onwards are for the user.

Input Status Words (BASIC Unit to CPU Unit)

Word m is the first word of the first input area allocated to the BASIC Unit.

n = 1515 + unit number x 25

Word	Bit	Name	Function
m	00	Task 0 Status Flag	Each flag of this area is turned ON when the corresponding task is executed. When the task is over, the flag turns OFF. To check whether the BASIC Unit is operating, check these flags.
	01	Task 1 Status Flag	
	02	Task 2 Status Flag	
	03	Task 3 Status Flag	
	04	Task 4 Status Flag	
	05	Task 5 Status Flag	
	06	Task 6 Status Flag	
	07	Task 7 Status Flag	
	08	Task 8 Status Flag	
	09	Task 9 Status Flag	
	10	Task 10 Status Flag	
	11	Task 11 Status Flag	
	12	Task 12 Status Flag	
	13	Task 13 Status Flag	
	14	Task 14 Status Flag	
	15	Task 15 Status Flag	
m + 1	00	Memory Overflow Flag	This flag turns ON when the user program executable code area or variable area is exceeded. (See Note 2.)
	01	Compile Error Flag	This flag turns ON when an error whose error code is 255 or lower has occurred. (See Note 2.)
	02	Compile Error Flag	This flag turns ON when an error whose error code is 256 or higher has occurred. (See Note 2.)
	03	E Code Error Flag	This flag turns ON when execution is specified from the E code, or if the E code is abnormal.
	04 to 14	---	Vacant (These bits are undefined.)
	15	Battery Error Flag	This flag turns ON when the supply voltage of the battery has dropped below a specific level. This flag turns OFF when the program is edited and executed after the battery voltage returns to normal.

Word	Bit	Name	Function
m + 2	00 to 07	Error Code	These bits indicate the contents of the system variable ERR in hexadecimal between 00 and FF. The Error Code is reset to 00 when the program is executed again.
	08	Fatal Error Flag	This flag turns ON when an error that causes the BASIC Unit to stop has occurred while the program is executed. This flag is turned OFF when the program is executed again.
	09	Port 1 Error Flag	These flags turn ON when an error has occurred in the corresponding ports. The possible causes that turn ON these flags are incorrect usage of the port, parity errors, overrun errors, and framing errors. These flags turn OFF when the program is executed again.
	10	Port 2 Error Flag	
	11	Port 3 Error Flag	
	12 to 14	---	Vacant (These bits are undefined.)
	15	BASIC Unit Execution Flag	This flag is turned ON when the BASIC Unit is executing a program. It is also turned ON when executable codes are being created or while a command is executed.

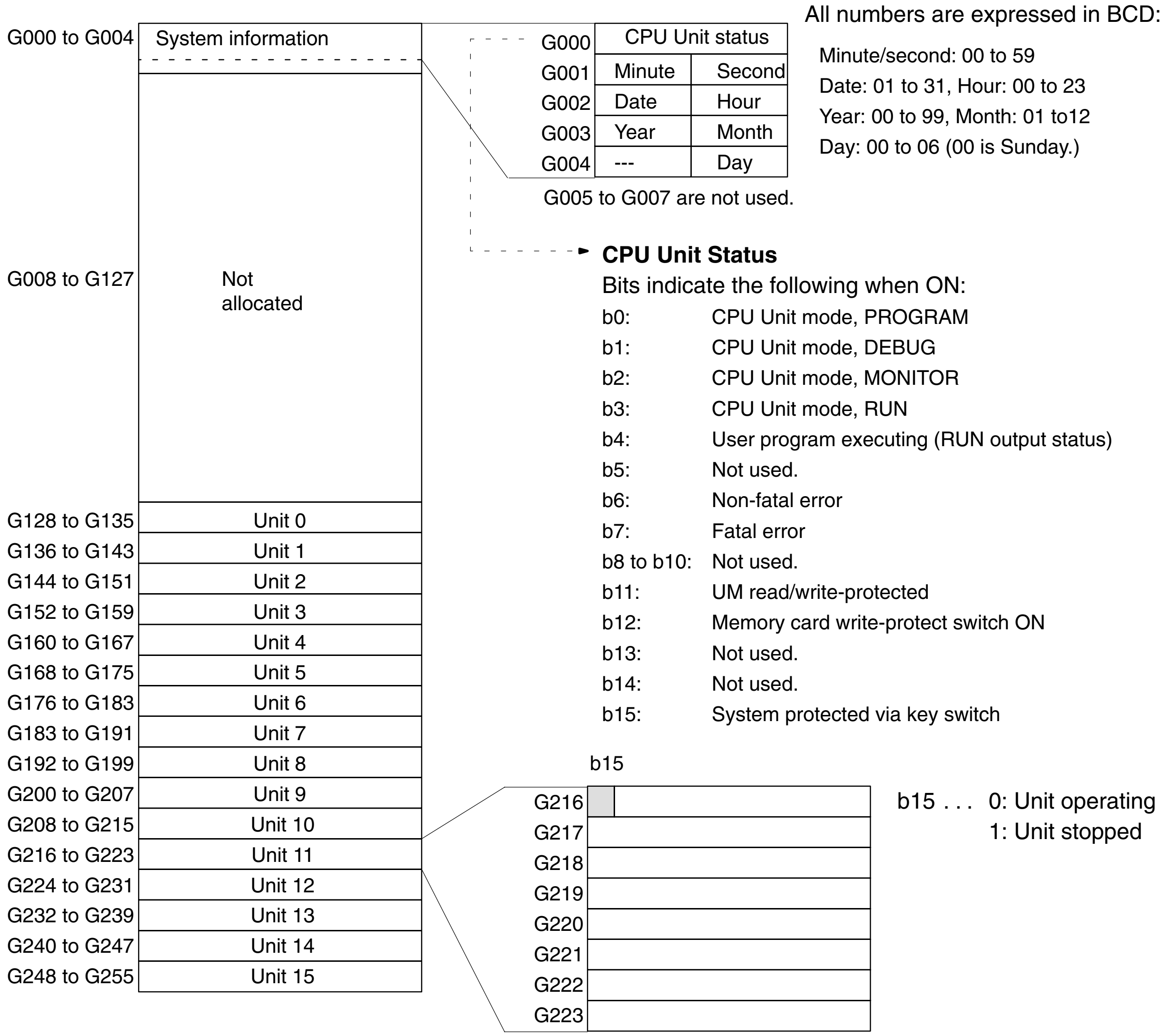
Words from word n+3 onwards are for the user.

Note
1. The error contents are the same as those displayed on the terminal connected to the BASIC Unit. For details on error codes, refer to *8-1 Troubleshooting*.
2. The Memory Overflow and Compiler Error Flags indicate the cause of errors when commands are input or when program execution is not possible. These flags can be turned OFF from the terminal with `TROFF`.

3-1-3 CPU Bus Link Area

The CPU Bus Link Area in the CPU Unit is used to automatically pass data back and forth between the BASIC Unit and the CPU Unit or between the BASIC Unit and another CPU Bus Unit. The default setting is for no CPU bus links. To use CPU bus links, specify them using the computer with terminal mode.

- The CPU Bus Link Area is refreshed in the CPU Unit at 10-ms intervals.
- Words in the CPU Bus Link Area are allocated by the CPU Unit according to the unit numbers of the CPU Bus Units.
- Data can be read from or written to this area by using the `PC READ` or `PC WRITE` commands.



Words	Contents
G000 to G004	System information
G008 to G127	Not allocated
G128 to G135	Unit 0
G136 to G143	Unit 1
G144 to G151	Unit 2
G152 to G159	Unit 3
G160 to G167	Unit 4
G168 to G175	Unit 5
G176 to G183	Unit 6
G183 to G191	Unit 7
G192 to G199	Unit 8
G200 to G207	Unit 9
G208 to G215	Unit 10
G216 to G223	Unit 11
G224 to G231	Unit 12
G232 to G239	Unit 13
G240 to G247	Unit 14
G248 to G255	Unit 15

G000	CPU Unit status	
G001	Minute	Second
G002	Date	Hour
G003	Year	Month
G004	---	Day

G005 to G007 are not used.

All numbers are expressed in BCD:

Minute/second: 00 to 59
Date: 01 to 31, Hour: 00 to 23
Year: 00 to 99, Month: 01 to12
Day: 00 to 06 (00 is Sunday.)

CPU Unit Status

Bits indicate the following when ON:

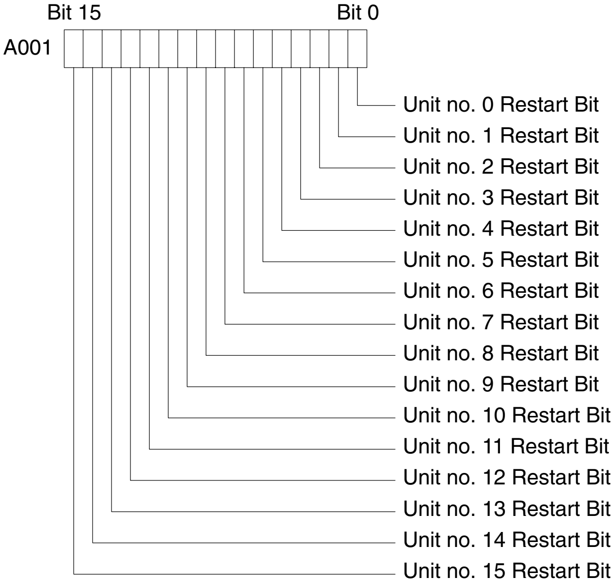
b0:	CPU Unit mode, PROGRAM
b1:	CPU Unit mode, DEBUG
b2:	CPU Unit mode, MONITOR
b3:	CPU Unit mode, RUN
b4:	User program executing (RUN output status)
b5:	Not used.
b6:	Non-fatal error
b7:	Fatal error
b8 to b10:	Not used.
b11:	UM read/write-protected
b12:	Memory card write-protect switch ON
b13:	Not used.
b14:	Not used.
b15:	System protected via key switch

b15 . . . 0: Unit operating
1: Unit stopped

Note
1. All Units can read any CPU bus link words.
2. The words that are not allocated (G008 to G127) can be used for any purpose by the CPU Unit program.
3. Words and bits specified as “Not used.” cannot be used for any purpose.
4. Bit 15 of the first word allocated to Units 0 through 15 is the Stop Flag for that Unit and indicates whether the Unit is operating or not. All other bits and words allocated to each Unit can be used as required by the user.
5. The system information (G000 to G004) can be read at any time.

3-1-4 Restart Bits

A Restart Bit is turned ON to restart a BASIC Unit. A001 contains Restart Bits for the CPU Bus Units. To restart a BASIC Unit, turn the corresponding bit of this area ON, and then back OFF again. These bits can be manipulated using the SET(016) ladder-diagram instruction or from a Programming Device. The bit number within this word corresponds to the unit number as shown below.



Note Unit numbers or memory switch setting cannot be changed by restarting a BASIC Unit using its Restart Bit. To change the unit number of memory switches, restart the Unit by resetting the CPU Unit.

⚠ **Caution** When routing tables are transferred to the CPU Unit, the corresponding Restart Bit will turn ON and the BASIC Unit will stop.

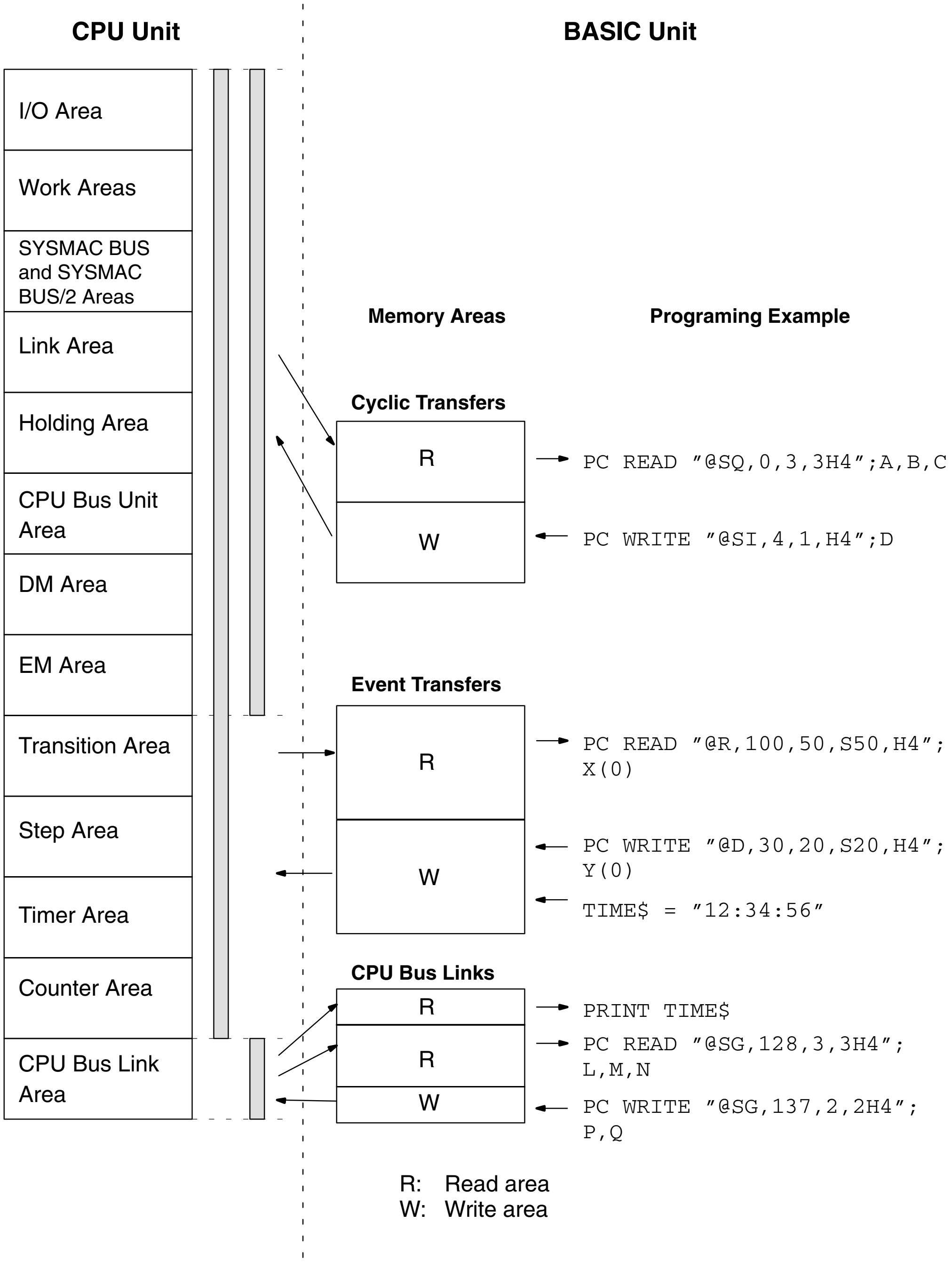
3-2 Data Transfer with the CPU Unit

To transfer data between the BASIC Unit and CPU Unit, the following three methods are available. With each method, data is read and written using the `PC READ` and `PC WRITE` commands from the BASIC Unit. Programming the CPU Unit is not necessary. When desired, programming is also possible from the CPU Unit.

Data transfer	Application
Cyclic	Specified words in the CPU Unit, set in advance using the software switches in the CPU Unit, are read or written during cyclic servicing. Since different areas can be simultaneously read and written, this method is used to transfer data when the same data needs to be transferred repeatedly. The output status from the CPU Unit to the BASIC Unit and the input status from the BASIC Unit to the CPU Unit is transferred or received using cyclic transfer. Software switches can be set to disable cyclic transfers.
Event	Specified data is read from or written to the CPU Unit when required. This method is most frequently used to transfer data.
CPU bus links	CPU bus links can be used to transfer small quantities of data with another BASIC Unit or the CPU Unit at high speeds. This method is used to operate the BASIC Unit in synchronization with another BASIC Unit or the CPU Unit, or to broadcast data to all other Units and the CPU Unit. CPU bus links are disabled in the default settings, but time information in the CPU Bus Link Area can be accessed. To specify CPU bus links, use the computer with terminal mode.

Data Flow

The following figure illustrates the areas to/from which data can be written/read by the three data transfer methods described previously, and examples of the BASIC commands used for the transfer. The data transfer method is determined by the suboperand of the `PC READ` or `PC WRITE` command.



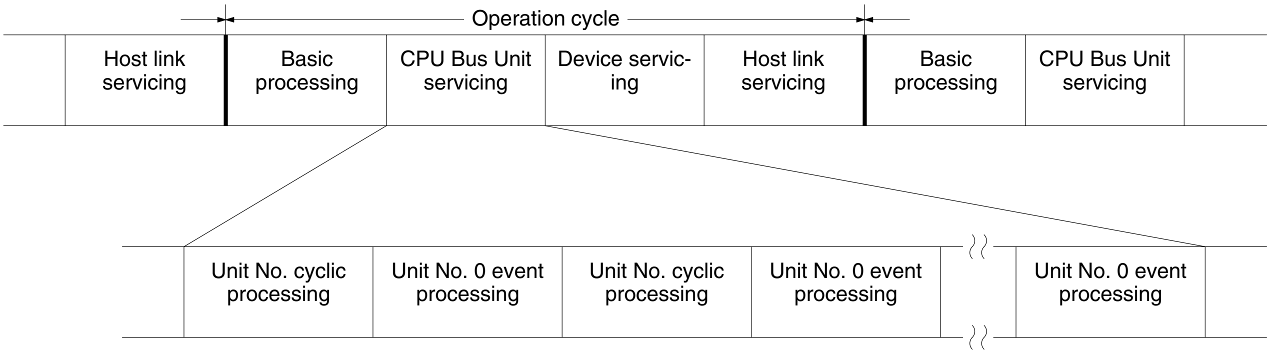
```
PC READ ”@SQ,0,3,3H4”;A,B,C
PC WRITE ”@SI,4,1,H4”;D
PC READ ”@R,100,50,S50,H4”;
X(0)
PC WRITE ”@D,30,20,S20,H4”;
Y(0)
TIME$ = ”12:34:56”
PRINT TIME$
PC READ ”@SG,128,3,3H4”;
L,M,N
PC WRITE ”@SG,137,2,2H4”;
P,Q
```

R: Read area
W: Write area

Data Transfer/Reception Timing

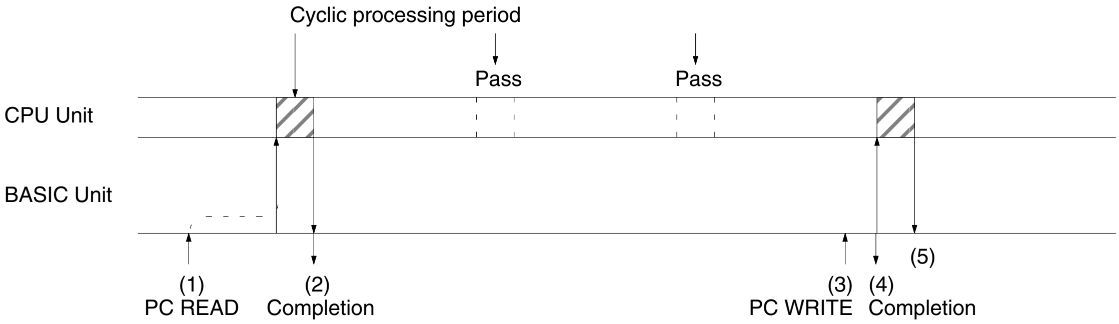
Data is transferred/received during the CPU Bus Unit service interval of the CPU Unit for both the cyclic and event transfer methods. The cycle at which this servicing is executed differs depending on whether the CPU Unit is operating synchronously or asynchronously. For details, refer to the *CV-series PC Operation Manual: Ladder Diagrams*.

CPU Unit Operation



CPU bus links are refreshed via interrupts every 10 ms.

Cyclic Transfers



Timing (1) and (2)

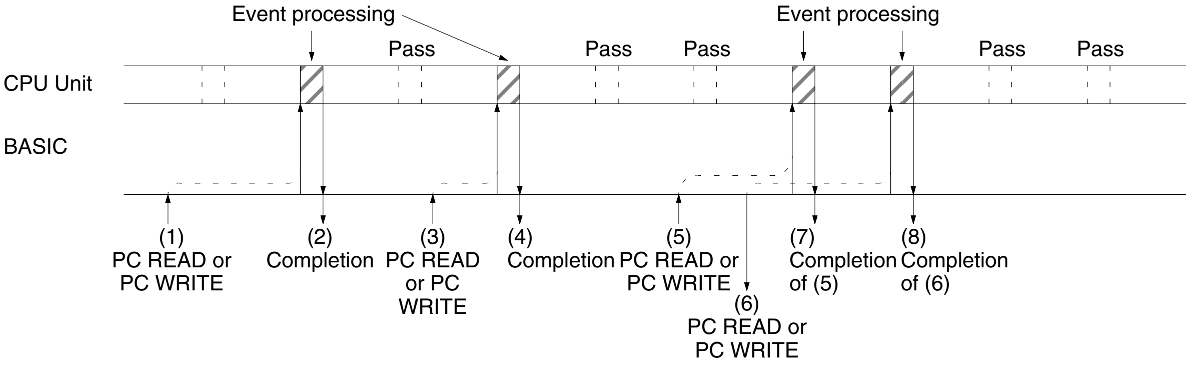
If the BASIC Unit has executed the `PC READ` instruction when the cyclic processing period arrives, the CPU Unit will process data transfer/reception.

Timing (3), (4), and (5)

The `PC WRITE` instruction writes data to the internal area of the BASIC Unit and then ends immediately. Data transfer to the CPU Unit is executed during the next cyclic processing period.

Note If neither the `PC READ` nor the `PC WRITE` instruction is executed, output status from the CPU Unit to the BASIC Unit and input status from the BASIC Unit to the CPU Unit will be transferred every 100 ms.

Event Transfers



Timing (1) to (2) and (3) to (4)

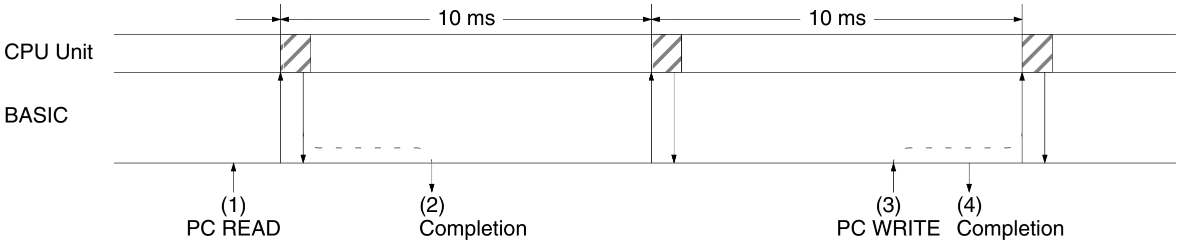
If the `PC READ` or `PC WRITE` instruction is executed by the BASIC Unit immediately before the event processing period, the CPU Unit transfers/receives the data immediately.

Timing (5) to (7) and (6) to (8)

If more than one `PC READ` or `PC WRITE` instruction is executed before the processing of one event, any subsequent instructions are kept pending until the next event processing.

CPU Bus Link Transfers

For CPU bus links, the CPU Unit reads data from each CPU Bus Unit each 10 ms, and then writes the entire CPU Bus Link Area to all the Units.



Timing (1) to (2) and (3) to (4)

When `PC READ` is executed, data written from the CPU Unit is read when the next CPU bus link servicing is performed. When `PC WRITE` is executed, data is read into the CPU Unit and other CPU Bus Units (such as other BASIC Units) when the next CPU bus link servicing is performed.

3-3 Memory Switches

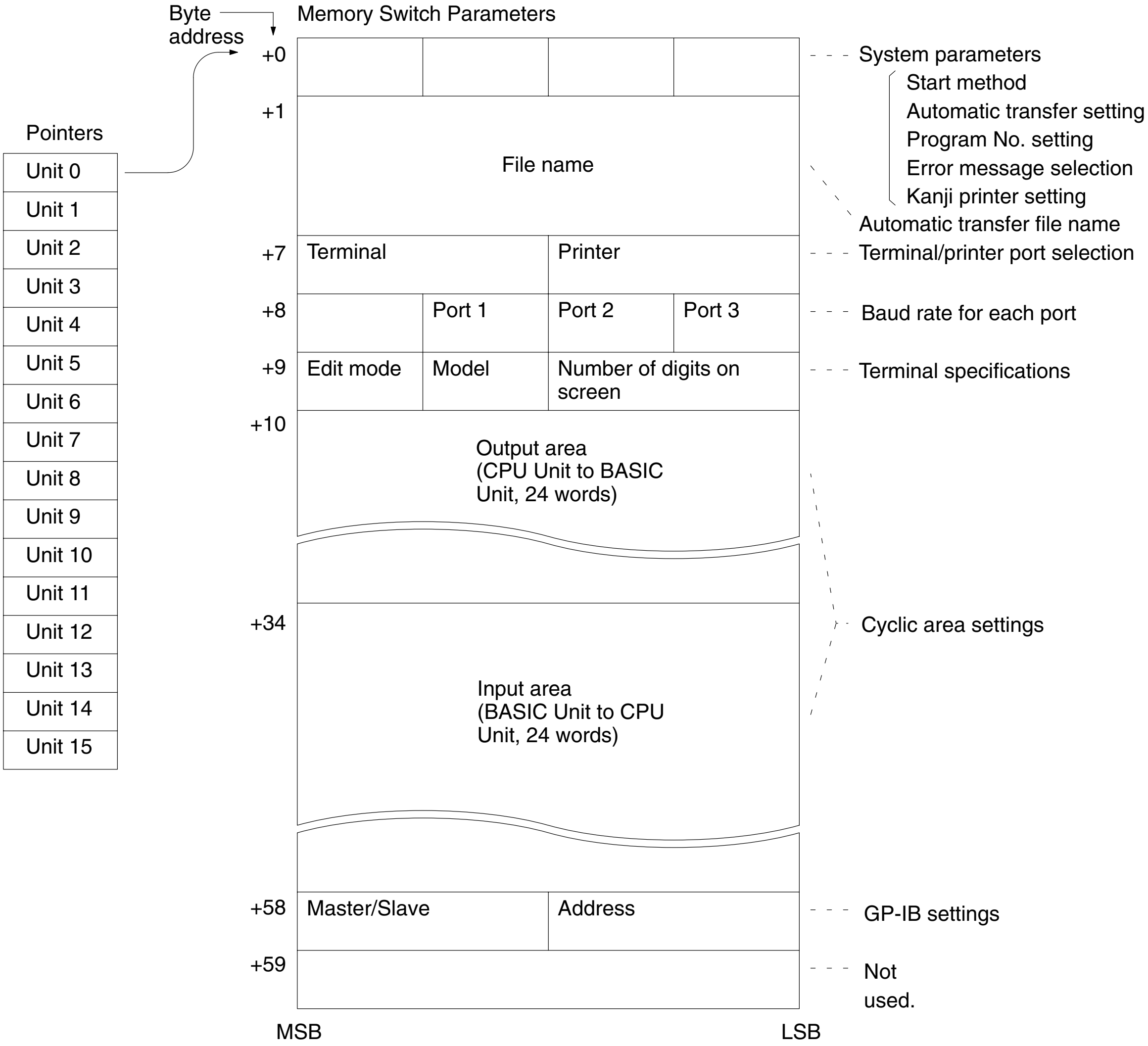
Memory switches are software switches containing operating parameters that control BASIC Unit operation. These parameters are kept in the CPU Unit and are transferred to the BASIC Unit whenever the system is turned ON or restarted. Each BASIC Unit has its own memory switches. (The memory switches are collectively called the CPU Bus Unit System Setup.)

The memory switches for each BASIC Unit consist of a pointer to the memory switches for the Unit and the settings of the memory switches. The default setting can be changed to alter BASIC Unit operating parameters. To write data to the memory switches, use a terminal connected to the BASIC Unit or a Graphic Programming Console with a CV-series Memory Cassette connected to the CPU Unit.

The memory switch settings can be momentarily returned to their default settings without changing the actual settings by turning ON pin 2 on the DIP switch on the front of the BASIC Unit. This is useful if a terminal cannot be connected because of unknown memory switch settings.

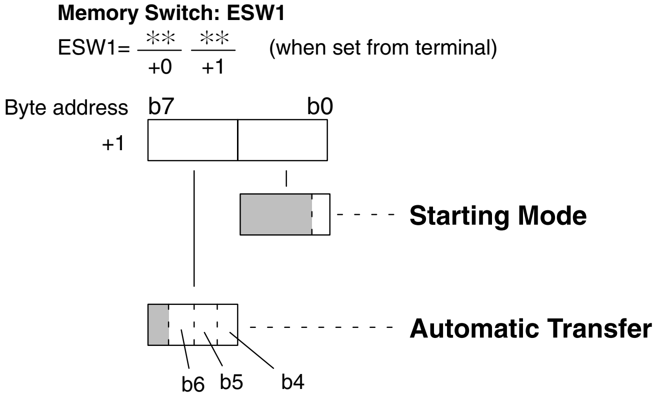
The memory switches consist of the following parameters. The area for each BASIC Unit occupies 60 words. Each parameter is described in detail in the following sections.

Note The Extended PC Setup in the CPU Unit, which includes BASIC Unit memory switch settings, can be transferred to and from Memory Cards. Refer to memory card operations in the *CVSS: Online Operation Manual* for details.



3-3-1 System Parameters

The system parameters of the memory switch set the basic items related to the operation of the BASIC Unit. The following figure illustrates the bit configuration of the system parameters. Set the bits shaded in this figure to `0`.



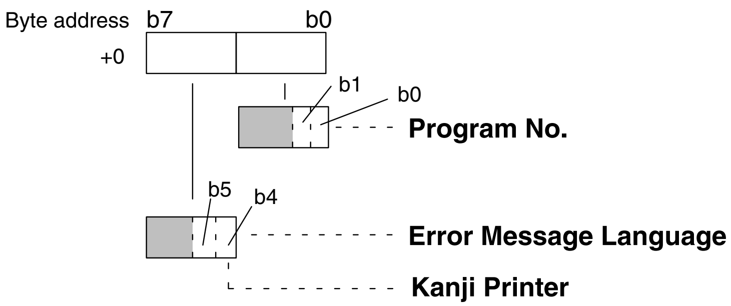
This system parameters are initially set to `0000`, i.e., manual start, manual transfer, program 1, English, and no Kanji printer.

b0: Starting Mode

Setting		Function
`0`	Manual start	The user program is started when `RUN` is input from the terminal after the version has been displayed by inputting CTRL-X. The RUN/STOP switch must be set to RUN to manually start program execution.
`1`	Automatic start	The user program is automatically started on power application or restarted with the RUN/STOP switch set to the RUN position. If the RUN/STOP switch is set to the STOP position, the program is started when the RUN/STOP switch is set to the RUN position.

b6, b5, b4: Automatic Transfer

Setting		Function
`000`	Manual transfer	Automatic transfer is not executed.
`100`	EEPROM automatic transfer	The user program is automatically transferred from the EEPROM to the source code area on power application or restarting (only models with EEPROM). Write the necessary program to the EEPROM in advance by using `ROMSAVE`.
`101`	File automatic transfer 1	The user program is automatically transferred from the memory card in the CPU Unit to the source code area on power application or restarting. The file name is specified by the following words in the memory switches. A memory card must be mounted to the CPU Unit.



b1, b0: Program No.

Setting		Function
00	Program 1	Sets program 1 as the user program to be edited on power application or resetting.
01		
10	Program 2	Sets program 2.
11	Program 3	Sets program 3.

b4: Error Message Language

Setting		Function
0	English	Error messages are displayed in English.
1	Japanese	Error messages are displayed in Japanese.

b5: Kanji Printer

Setting		Function
0	Not used	Not compatible with Kanji printer.
1	Used	Specifies KI/KO processing. (K1 = 1B4B, K0 = 1B48)

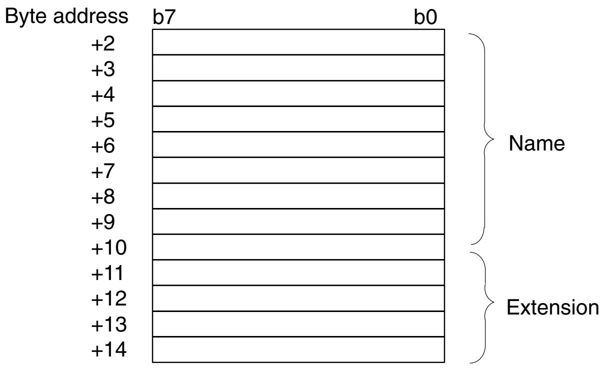
3-3-2 Automatic Transfer File Name

When automatic program transfer is specified in the automatic transfer setting of the system parameters, the name of the file to be transferred must be specified. If the file is specified to be manually transferred, the file name does not need to be specified.

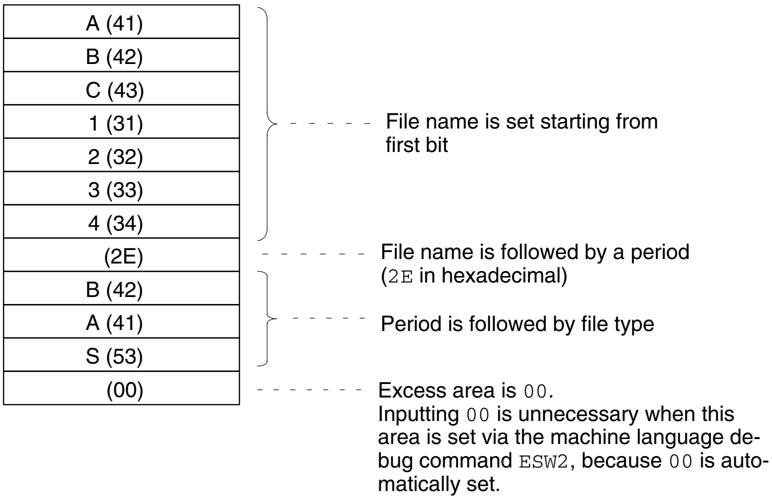
The file name may consist of up to 8 characters of ASCII followed by a file type (extension) delimited by a period from the file name. The file name must consist of alphanumeric characters starting with an alphabetic character. The file extension is BAS.

Memory Switch: ESW2

```
ESW2=✻ ✻ ✻ ✻ ✻ ✻ ✻ ✻ ✻ ✻ ✻ ✻    (when set from terminal)
     0 1 2 3 4 5 6 7 8 9 10 11
```



Example: File Name ABC1234.BAS



File name is set starting from first bit

File name is followed by a period (2E in hexadecimal)

Period is followed by file type

Excess area is 00. Inputting 00 is unnecessary when this area is set via the machine language debug command ESW2, because 00 is automatically set.

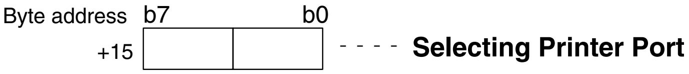
Note Any file can be read and used as a user program by using the automatic file transfer function. However, if automatic transfer is specified, a certain amount of time is required to read the program from the file and create executable codes before the program is actually started.

3-3-3 Terminal and Printer Ports

This area of the memory switch specifies the ports to which the terminal and printer are connected. Set a BCD number from 01 to 04 in this area, by referring to the following illustration:

Memory Switch: ESW3

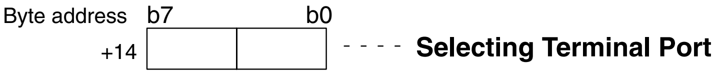
ESW3= $\underset{+14}{\underline{**}}\ \underset{+15}{\underline{**}}$ (when set from terminal)



Selecting Printer Port

Selecting Printer Port

Setting		Function
01	RS-232C 1	Selects top RS-232C (port 1) port as printer port.
02	RS-232C 2	Selects bottom RS-232C (port 2) port as printer port.
04	Centronics	Selects Centronics port as printer port. With the BSC11/21, this sets the Unit as having no printer port.
FF	None	No printer port set. **Note:** This setting is only possible with system ROM versions 1.23 or higher.



Selecting Terminal Port

Selecting Terminal Port

Setting		Function
01	RS-232C 1	Selects top RS-232C (port 1) port as terminal port
02	RS-232C 2	Selects bottom RS-232C (port 2) port as terminal port
03	RS-422	Selects RS-422 (port 3) port as terminal port
FF	None	No terminal port set. If it becomes necessary to use a terminal, disable the memory switch settings using pin 2 of the DIP switch. **Note:** This setting is only possible with system ROM versions 1.23 or higher.

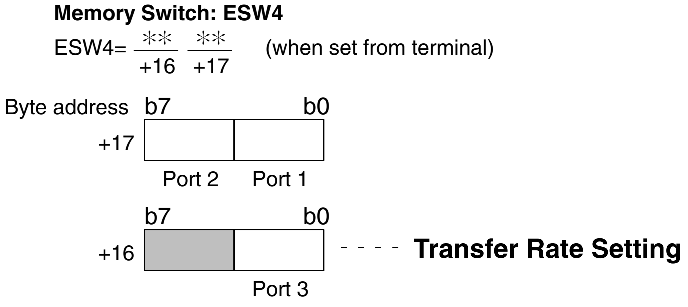
Note The default is `0000`. Consequently, the following printer and terminal ports are selected:

BSC11/BSC21: `0102` (port 1 as terminal port and port 2 as printer port)
BSC31/BSC41: `0104` (port 1 as terminal port and Centronics as printer port)
BSC51/BSC61: `0100` (port 1 as terminal port and no printer port)

- The system ROM version is displayed on the BASIC initial screen on the terminal.
- Communications control using RTS/DTR signals is not possible for the ports set as the terminal and printer ports. To perform communications control using RTS/DTR signals, change the ports set as the terminal and printer ports to ports other than the ones for which RTS/DTR control is to be used. This is done using memory switch 3. Also, if the system ROM version is 1.23 or higher, it is possible to not set a printer port and terminal port by setting the relevant bytes to FF.

3-3-4 Baud Rates

This area sets the baud rates of RS-232C ports (ports 1 and 2) and RS-422 port (port 3). Set a BCD number from `0` to `7` to the area corresponding to each port by referring to the following illustration:



Transfer Rate Setting

Setting	Function
`0`	Sets the baud rate to 9,600 bps (default).
`1`	Sets the baud rate to 300 bps.
`2`	Sets the baud rate to 600 bps.
`3`	Sets the baud rate to 1,200 bps.
`4`	Sets the baud rate to 2,400 bps.
`5`	Sets the baud rate to 4,800 bps.
`6`	Sets the baud rate to 9,600 bps.
`7`	Sets the baud rate to 19,200 bps.

Note

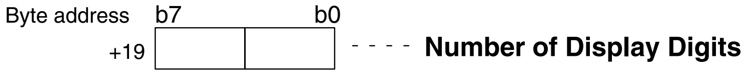
1. The default is `0000`, i.e., the transfer rate of all the ports is 9,600 bps.
2. Be sure to clear the bits shaded in the previous figure to `0`.
3. The RUN echoback will overlap with the port initialization display if program execution is started from a terminal connected to a port set to 300 bps. Always set the port connected to the terminal to 600 bps or greater if you are going to use the terminal to start program execution.

3-3-5 Terminal Specifications

This memory switch sets the model of the terminal and the number of display digits for the terminal connected to the BASIC Unit.

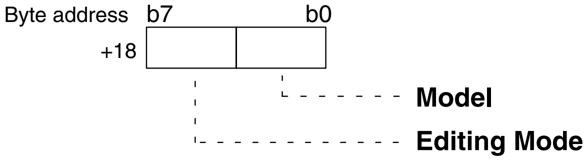
Memory Switch: ESW5

ESW5= $\frac{**}{+18}\ \frac{**}{+19}$ (when set from terminal)



Number of Display Digits

This byte sets the number of display digits of the terminal in 2 BCD digits. When this byte is set to `00`, 24 digits, which is the default value, is assumed.



Model

Setting		Function
0	Terminal mode	Specifies terminal mode.
1	VT-52 (VT-52 mode)	Specifies VT-52 or equivalent.
2	VT-100 (ANSI mode)	Specifies VT-100 or equivalent.

Editing Mode

Setting		Function
0	Overwrite	Sets overwrite mode for program editing
1	Insert	Sets insert mode for program editing

Note The default value is `0000`. Consequently, terminal mode is selected with the number of display digits set to 24 and the overwrite mode already set.

3-3-6 Cyclic Area Settings

This area of the memory switches sets the area of the CPU Unit with which the BASIC Unit will cyclically (periodically) transfer data. Up to six output areas (CPU Unit to BASIC Unit) and up to six input areas (BASIC Unit to CPU Unit) can be set. Up to 384 words can be set for all areas combined.

If this area is not set, the following defaults are used. These are in the CPU Bus Unit Area.

Area:	I/O memory area
Address:	Output: 15 (first 15 words)
	Input: 10 (last 10 words)
Number of areas:	1 for both output and input

Caution Keep the first word address and number of words to within the range of each area. If an improper word address is set, all the settings of the input and output areas following the improper word address will be invalid. A range check is not performed for this setting. Check your settings and input values carefully.

Each setting area consists of 4 words. For unused areas, set `0000` as the area setting. If `0000` is set as the area type setting for all the areas, cyclic data transfer is not executed.

A minimum of 3 words is required in the input area to refresh BASIC Unit information.

Memory Switch: ESW6

ESW6–1= $\underset{+20}{\underline{**}}\ \underset{+21}{\underline{**}} - \underset{+22}{\underline{**}}\ \underset{+23}{\underline{**}} - \underset{+24}{\underline{**}}\ \underset{+25}{\underline{**}} - \underset{+26}{\underline{**}}\ \underset{+27}{\underline{**}}$ (when set from terminal)

Byte address	b7 b0
+20	Output area 1
+28	Output area 2
+36	Output area 3
+44	Output area 4
+52	Output area 5
+60	Output area 6
+68	Input area 1
+76	Input area 2
+84	Input area 3
+92	Input area 4
+100	Input area 5
+108	Input area 6

8 words ⇨

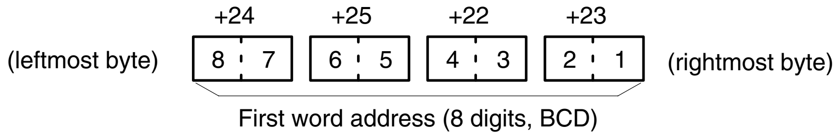
Byte address	b7	b0
+20	0	0
+21	Area setting (See the following table.)	
+22	First word address (See the following table)	
+23		
+24		
+25		
+26	Number of words (See the following table)	
+27		

Area Setting

Setting	Function
0080	I/O Memory Area
0082	Data Memory Area
0090	Expansion Data Memory Area, bank 0
0091	Expansion Data Memory Area, bank 1
0092	Expansion Data Memory Area, bank 2
0093	Expansion Data Memory Area, bank 3
0094	Expansion Data Memory Area, bank 4
0095	Expansion Data Memory Area, bank 5
0096	Expansion Data Memory Area, bank 6
0097	Expansion Data Memory Area, bank 7
0000	None

First Word Address

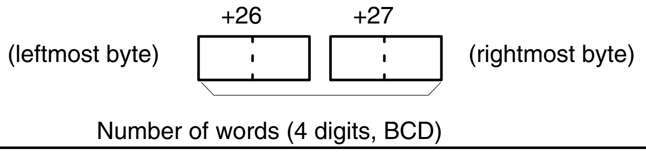
Function
Specifies the first word address of the specified area in 8 digits BCD.



Note: The order of the byte address when setting in 8 digits, BCD is 4,3→2,1→8,7→6,5 (where the numbers indicate the number of the digit).

Number of Words

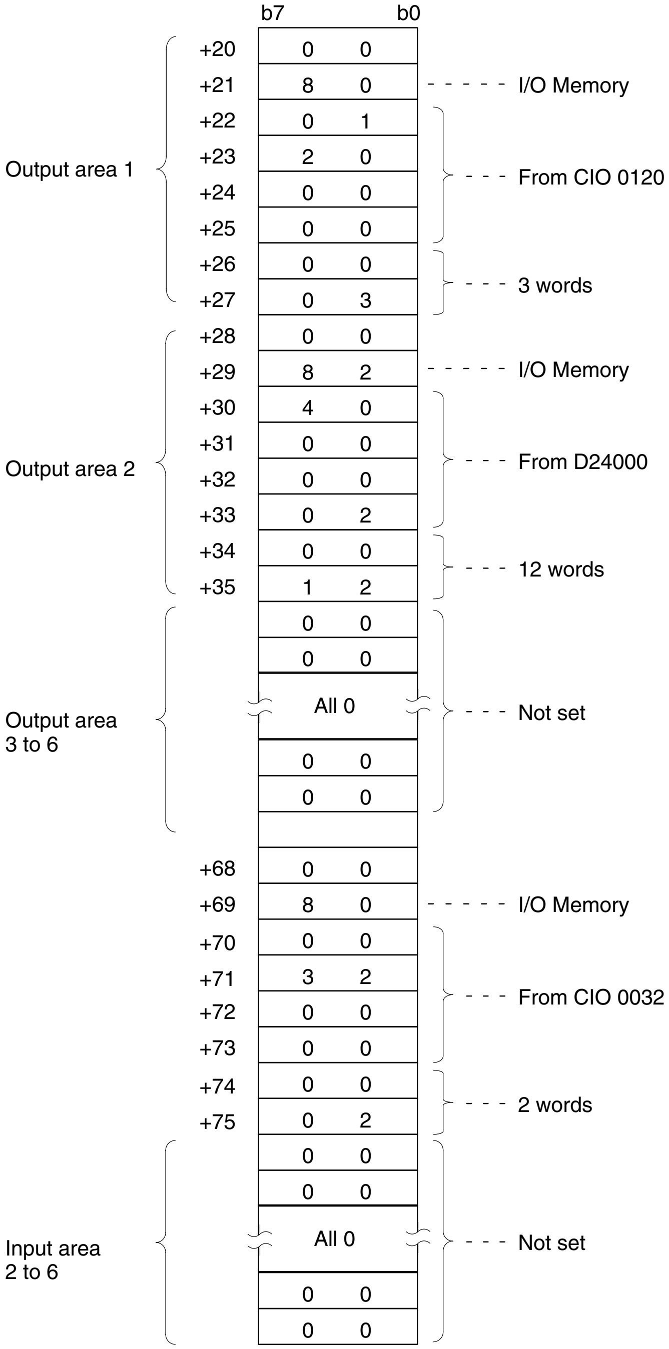
Function
Specifies the number of words in the specified area in 4 digits BCD.



Example

2 Output Areas: 3 words from CIO 0120 of I/O Memory.
12 words from D24000 of DM Area.

1 Input Area: 2 words from CIO 0032 of I/O Memory.



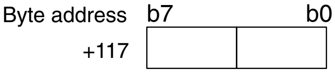
		b7	b0	
Output area 1	+20	0	0	
	+21	8	0	I/O Memory
	+22	0	1	From CIO 0120
	+23	2	0	
	+24	0	0	
	+25	0	0	
	+26	0	0	3 words
	+27	0	3	
Output area 2	+28	0	0	
	+29	8	2	I/O Memory
	+30	4	0	From D24000
	+31	0	0	
	+32	0	0	
	+33	0	2	
	+34	0	0	12 words
	+35	1	2	
Output area 3 to 6		0	0	Not set
		0	0	
		All 0		
		0	0	
		0	0	
	+68	0	0	
	+69	8	0	I/O Memory
	+70	0	0	From CIO 0032
	+71	3	2	
	+72	0	0	
	+73	0	0	
	+74	0	0	2 words
	+75	0	2	
Input area 2 to 6		0	0	Not set
		0	0	
		All 0		
		0	0	
		0	0	

3-3-7 GP-IB Setting

This parameter sets the operation of the GP-IB interface. The parameter is necessary only for the CV500-BSC51 and CV500-BSC61.

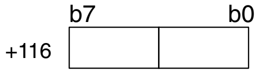
Memory Switch: ESW7

ESW7= $\underset{+116}{\underline{**}}\ \underset{+117}{\underline{**}}$ (when set from terminal)



---- **Address of Talker and Listener**

Sets addresses of talker and listener in BCD (00 to 30).



---- **Master/Slave Setting**

Master/Slave Setting

Setting		Function
00	Master	Sets BASIC Unit as master.
01	Slave	Sets BASIC Unit as slave.

3-4 Setting Memory Switches

The memory switches can be set from a Graphic Programming Console with a CV-series Memory Cassette connected to the CPU Unit or by a terminal connected to the BASIC Unit. The CVSS is not currently equipped with this feature.

Memory switch settings can be transferred from one CPU Unit to another using the CVSS and copying the Extended PC Setup onto a Memory Card. Refer to the Memory Card operations in the *CV Support Software: Online Operation Manual* for details.

The following procedures will explain how to set the memory switch using a terminal connected to the BASIC Unit. When the following procedure is completed, new software memory settings will exist in both the BASIC Unit and the CPU Unit.

Refer to *Appendix J* for details on setting methods using Support Software.

1, 2, 3...

1. First, set the BASIC Unit in the machine language monitor mode. When the message `OK` is displayed, or while the terminal is in the command input wait status, input `MON` followed by a carriage return.
2. A prompt (*) will be displayed and the BASIC Unit will be set in the machine language mode. Input as follows to set each memory switch. Input upper-case characters.

   ```
   ESW1=0300
   ```

   Here, `1` is the memory switch and `0300` is the setting (hexadecimal).

   For the settings, refer to *3-3 Memory Switches*. The memory switch areas are as follows:

   1: System parameters (ESW1)
   2: Automatic transfer file name (ESW2)
   3: Terminal/printer ports (ESW3)
   4: Baud rates (ESW4)
   5: Terminal specifications (ESW5)
   6: Cyclic area setting area (ESW6)
   7: GP-IB setting (ESW7)
3. Set the cyclic areas as follows:

   ```
   ESW6–1=0080–0100–0000–0008
   ```

   Here, `1` is the output/input area no., `0080` is the area type no., `0100` are the rightmost bytes of the first word address (BCD), `0000` are the rightmost bytes of the first word address (BCD), and `0008` is the number of words (BCD). This setting sets 8 words beginning from word 100 in the I/O memory area as output area 1.

Output/Input Area Numbers

Output area 1 to 6	`1` to `6`
Input area 1 to 6	`7` to `12`

Area Specifications

I/O Memory Area	`0080`
Data Memory Area	`0082`
Expansion Data Memory Area, bank 0 through bank 7	`0090` to `0097`

4. After setting all the memory switches, input `ESW-W` followed by a carriage return to write the data to the CPU Unit.

SECTION 4
Programming Overview

This section provides an overview of BASIC programming and is not meant to provide a comprehensive explanation of BASIC programming.

4-1 BASIC Syntax and Operations

4-1-1 Syntax

To develop a program in BASIC, an understanding of the syntax and description of BASIC is essential. This section describes some fundamentals of the BASIC syntax and programming techniques. For the details of the BASIC syntax, refer to the *BASIC Unit Reference Manual (W207-E1)*.

Line Numbers and Labels

Line Numbers

A program consists of lines. Each line consists of a line number, executable statement, a comment statement, and/or a non-executable statement.

```
10 PRINT ″BASIC UNIT″ . ........ Executable statement
20 REM *** BASIC UNIT*** . .... Comment statement
30 DIM A(10) . ................. Non-executable statement

60 IF A$ = ″ ″ THEN GOTO 40  . Executable statement

90 END . ........................ Executable statement

                                   Line number
```

Line numbers are integers from 1 to 65529 and are arranged in ascending order. The program is executed in the order of the line numbers. The line numbers are sometimes used to specify the destination to where the program execution is branched with the GOTO and GOSUB commands.

Labels

A label is a name assigned to a line number to specify the branch destination of such commands as GOTO and GOSUB. With the BASIC Unit, a label must start with an asterisk (*) and followed by an alphabetic character.

```
50 GOSUB *LABEL . .......... Calling by label
60 GOSUB 80 . ............... Calling by line number
70 END
80 *LABEL . ................. Label
90 RETURN
```

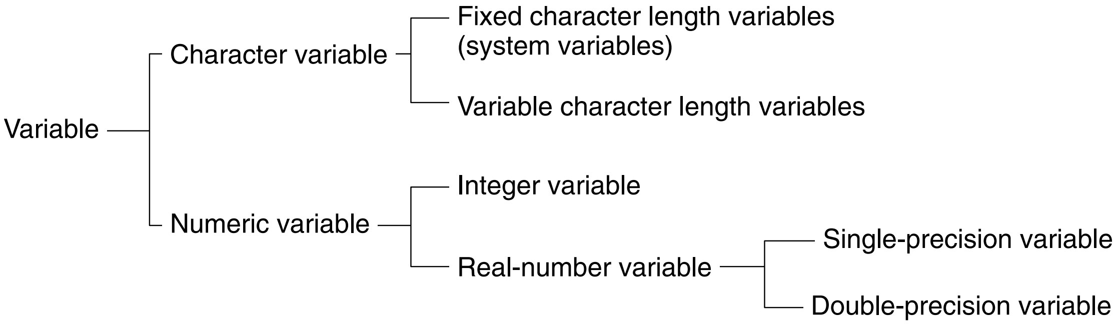
If a line number is specified as a branch destination, and if the line number changes when the program is modified, an error will occur. However, if a label is used, the label will remain the same even when the program is modified.

Variables and Constants

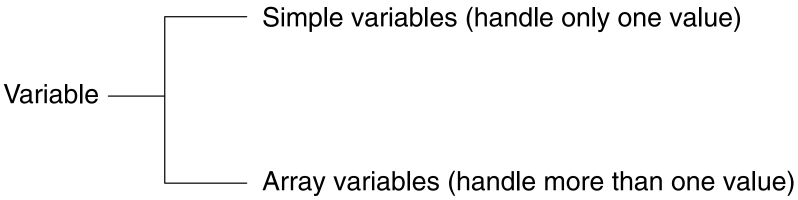
Variables

A computer handles various types of data such as characters and numeric values. In a computer language such as BASIC, areas called variables in which data is temporarily stored are used so that a program can be easily developed. A variable is given a variable name and is assigned a value after substitution or after an operation has been executed.

Data is classified into character data and numeric data. This also applies to variables, which can be classified into character variables in which character data is stored and numeric variables in which numeric data is stored. Numeric variables are further classified into integer variables and real-number variables. Real-number variables are then further classified into single-precision variables and double-precision variables. These relationships are shown as follows.


Variable
Character variable
Fixed character length variables (system variables)
Variable character length variables
Numeric variable
Integer variable
Real-number variable
Single-precision variable
Double-precision variable

In addition to the above classifications, variables are also classified into simple variables which handle only one piece of data, and array variables which handle more than one piece of data.


Variable
Simple variables (handle only one value)
Array variables (handle more than one value)

The variable name given to a variable is specified by using alphanumeric characters, a period (.), and a declarator. The length of a variable can be up to 40 characters including the declarator. The declarator specifies the type of the variable, as follows:

`$` ... Character

`%` ... Integer

`!` ... Single-precision real number (this type is assumed if no type declarator is specified)

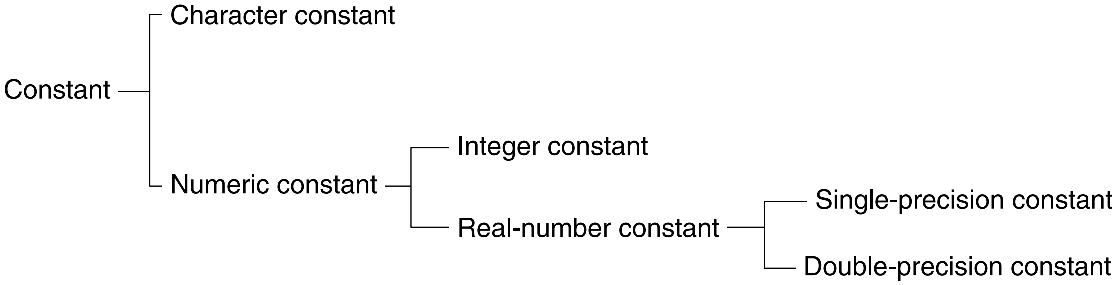
`#` ... Double-precision real number

For example, `A%`, `A`!, `A#`, and `A$` all indicate different variables. If the type declarator is omitted, a single-precision real-number type is assumed, and consequently, `A` and `A!` indicate the same variable.

Note The default length for character variables is fixed at 18 characters. Because of this, garbage collection is not performed. If character variable length needs to be changed, use the `OPTION LENGTH` instruction before `PARACT 0`. Length checks are not performed for substitution into character variables.

Constants

The contents (data) of a variable are changed by a substitution or operation. In contrast, a constant, which indicates a value by itself, is used where data does not need to be changed. Like variables, constants are classified into character constants and numeric constants, which are further classified into integer constants and real-number constants. The real-number constants are further divided into single-precision constants and double-precision constants. These relationships are shown below.



A character constant usually consists of a character string of 255 characters or less enclosed by a pair of double quotation marks. These characters can be in alphanumeric characters and/or symbols.

Examples `"12345"`
`"BASIC UNIT"`

Numeric constants are expressed as a positive or negative value, or as 0, and are specified in decimal, octal, hexadecimal, or exponential format.

Decimal (–32768 to 32767):

```
9200 . .......................
123%
```

Octal (&0 to &77777):

```
&123 . .......................
&O200
```

Hexadecimal (&H0 to &HFFFF):

```
&H123 . ......................
&H2B3F
```

Exponential (single precision) (–3.4E + 38 to 3.4E + 38):

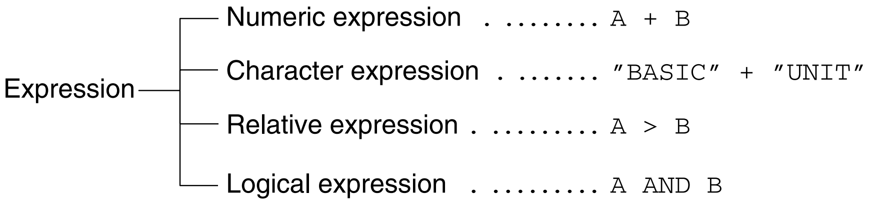
```
-1.23E + 4 . ................
345.2!
```

Exponential (double precision) (–1.701411834604692D + 307 to 1.701411834604692D + 307):

```
-1.23D - 12 . ...............
345.2#
```

Types of Expressions

Expressions are classified into numeric, character, relative, and logical expressions depending on the type of the value handled in the expressions.


Expression
Numeric expression . ... A + B
Character expression . ... "BASIC" + "UNIT"
Relative expression . ... A > B
Logical expression . ... A AND B

Numeric Expressions

A numeric expression returns a numeric value and consists of numeric variables and numeric constants coupled with arithmetic operators. The arithmetic operators shown in the following table can be used.

Arithmetic operator	Operation	Example
+	Addition	A + B
–	Subtraction	A – B
*	Multiplication	A * B
/	Real-number division	A / B
¥ or \	Integer division	A¥B, A \ B
^	Exponent calculation	A ^ B
MOD	Remainder calculation	A MOD B

Note ¥ or \ depend on the terminal used.

Character Expressions

A character expression returns a character string and consists of character variables and character constants coupled with an arithmetic operator (+).

Example

```
”OMRON” + ”Corporation”
```

Relative Expressions

A relative expression consists of numeric expressions coupled with a relative operator. The relative operators shown in the following table can be used.

Relative operator	Operation	Example
=	Equal	A = B
<>, ><	Not equal	A <> B, A >< B
<	Less than	A < B
>	Greater than	A > B
<=, =<	Less than or equal to	A <= B, A =< B
>=, =>	Greater than or equal to	A >= B, A => B

Logic Expressions

A logic expression is used to execute logic operations, manipulate bits, or check conditions of IF statements. A logic operator is used to form a logic expression. The logic operators shown in the following table can be used.

Logic operator	Operation	Example
NOT	Negation	NOT A
AND	Logical product	A AND B
OR	Logical sum	A OR B
XOR	Logical exclusive sum	A XOR B
IMP	Implication	A IMP B
EQV	Equivalence	A EQV B

Result of Operations by Logic Operator

NOT

A	NOT A
0	1
1	0

AND

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

XOR (Exclusive OR)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

IMP (Implication)

A	B	A IMP B
0	0	1
0	1	1
1	0	0
1	1	1

EQV (Equivalence)

A	B	A EQV B
0	0	1
0	1	0
1	0	0
1	1	1

BASIC Functions

The BASIC Unit supports many functions in addition to ordinary BASIC functions. A function is used to perform a predetermined operation on a given argument. Some functions return numeric values, while others return character strings. These are explained in more detail later in this manual and in the *BASIC Unit Reference manual*.

Functions Returning Numeric Values

Function	Operation
ABS	Gives absolute value
ACOS	Gives arc cosine
ASC	Gives character code
ASIN	Gives arc sine
ATN	Gives arc tangent
CDBL	Converts integer value or single-precision value into double-precision value
CINT	Converts real-number value into integer value
COS	Gives cosine
CSNG	Converts integer value or double-precision value into single-precision value
CVI/CVS/CVD	Converts character string into numeric value
EOF	Gives end code of file
ERL/ERR	Gives line in which error occurs and error code
EXP	Gives value of exponential function
FIX	Gives integer
FRE	Gives size of unused memory area
INSTR	Searches characters string and gives position of character
INT	Gives integer value truncated at decimal point
INTRB/INTRL/INTRR	Gives destination line, generation line, and type of interrupt
LEN	Gives total number of characters of character string
LOC	Gives present value in FILE

Function	Operation
`LOF`	Gives size of `FILE`
`LOG`	Gives natural logarithm
`PEEK`	Returns contents of specified address
`RND`	Gives random number
`SEARCH`	Searches element of array variable and gives position of character
`SGN`	Checks sign
`SIN`	Gives sine
`SPC`	Outputs blank
`SQR`	Gives square root
`TAB`	Sets column position of screen or printer
`TAN`	Gives tangent
`USR`	Calls machine language function on memory
`VAL`	Converts character string into numeric value
`VARPTR`	Gives storage address of variable

Functions Returning Character String

Function	Operation
`CHR$`	Gives character having specified character code
`DATE$`	Gives date
`HEX$`	Converts into hexadecimal number
`INKEY$`	Inputs only one character
`INPUT$`	Inputs only specified number of characters
`LEFT$`	Gets character string (from leftmost position)
`MID$`	Gets character string
`MKI$/MKS$/MKD$`	Converts numeric code into character code
`OCT$`	Converts into octal number
`RIGHT$`	Gets character string (from rightmost position)
`SPACE$`	Gives blank character string
`STR$`	Converts numeric value into character string
`STRING$`	Creates character string of specified characters
`TIME$`	Gives time

4-1-2 BASIC Operations

This section introduces examples of programming for fundamental operations of the BASIC Unit.

Displaying Data

To display data, program as follows by using the `PRINT` and `PRINT USING` commands:

To Display “BASIC UNIT” and Contents of Variable X

```
10 PARACT 0
20 X = 10
30 PRINT ”BASIC UNIT”
40 PRINT ”X = ”;X
50 PRINT ”X = ”,X
60 END
70 END PARACT
```

⇩ Result of execution

```
BASIC UNIT
X = 10
X =                        10
```

If a character (in this example, X) is delimited by “;”, it is displayed immediately after the character displayed immediately before. If it is delimited by “,”, the character is displayed from the beginning of the next area (one area consists of 14 characters). In addition, TAB specification that displays the current position of the cursor as character X coordinate = 0 can also be made.

Difference Between PRINT and WRITE Commands

The WRITE command has a similar function to the PRINT command. The WRITE command is also used to output data to the screen. With the WRITE command, the variables and expressions to be displayed are delimited by only commas when they are specified. They are also delimited by commas when they are displayed. To display a character string, it automatically encloses a pair of double quotation (") marks. To display a numeric value, unlike the PRINT command, no blank is placed before and after the numeric value.

Consequently, if the sample program shown previously is written by using the WRITE command instead of the PRINT command, the display will be as follows:

```
10 PARACT 0
20 X = 10
30 WRITE "BASIC UNIT"
40 WRITE "X = ";X
50 WRITE X, 20
60 END
70 END PARACT
```

⇩ Result of execution

```
"BASIC UNIT"
"X = ", 10
10, 20
```

To Specify Display Format

Sometimes, the data displayed by the PRINT command is hard to see. The PRINT USING command is used to specify the format in which the data are displayed, so that the data is easy to see.

```
10 PARACT 0
20 X = 1234.56
30 PRINT USING "#####.###";X
40 PRINT USING "+#####.###";X
50 PRINT USING "X = #####.##";X
60 PRINT USING "###.#";1234.5
70 END
80 END PARACT
```

⇩ Result of execution

```
  1234.560
  + 1234.560
X = 1234.56
%1234.5
```

The number of digits of a numeral, including that of the sign, is specified by the number of “#” marks. If the number of digits of the data is less than the specified number of “#” marks, the data is right-justified for display. If the number of digits is greater, “%” is prefixed to the extra digits of the data.

To Output Data to Printer

To output data to the printer, use the LPRINT or LPRINT USING command.

```
10 PARACT 0
20 LPRINT ”BASIC UNIT”
30 END
40 END PARACT
```

⇓ Result of execution

```
BASIC UNIT
```

END and STOP Commands Ending Program

Write the END command at the end of the program. This command closes all open files and terminates the execution of the program. However, sometimes it is necessary to stop the program under execution. For example, if a wrong key has been pressed, or if a certain condition is satisfied, it may be necessary to stop the program. In this case, the STOP command is used. When this command is executed, a message is displayed and the program execution is stopped.

```
10 PARACT 0
20 FOR I = 1 TO 100
30  IF 5 – I = 0 THEN STOP
40  PRINT I
50 NEXT I
60 END
70 END PARACT
```

⇓ Result of execution

```
 1
 2
 3
 4
Stop in 30
```

Inputting Data From Keyboard

To input data to the variables in the program from the keyboard, program as follows by using the INPUT or LINE INPUT command:

To Input Numeric Data

```
10 PARACT 0
20 INPUT A
30 INPUT B
40 PRINT A, B
50 END PARACT
```

Numeric data input

⇓ Result of execution

```
?  100
?  200
 100                         200
```

When the INPUT command is executed, ? is displayed, indicating that the program is waiting for the input of data. The program is stopped until data has been input. Then input a desired numeric value from the keyboard and press the carriage return.

To Input Character Data

If an attempt is made to input character data in the above example, an error occurs. To input a character, `$` must be suffixed to a variable name. This means that for the variable name specified by the `INPUT` command, the data type of the variable must be specified by `$`, depending on the type of the data to be input.

```
10 PARACT 0
20 INPUT A$
30 INPUT B$
40 PRINT A$; B$
50 END PARACT
```

Character data input

⇩ Result of execution

```
?  BASIC
?  UNIT
BASIC UNIT
```

For example, to input integer type numeric value in the above program, `%` must be suffixed to the numeric value, like `A%` and `B%`. To input a numeric value of double-precision real-number type, `#` must be suffixed.

To Display Message While Data is Input

The `INPUT` command is used to input data to a variable while the program is executed. However, it may be unclear which data is to be input if only "`?`" is displayed when the `INPUT` command has been executed. To clarify which data should be input, therefore, a message can be displayed before "`?`".

```
10 PARACT 0
20 INPUT ″NAME″;A$
30 INPUT ″TEL ″;B$
40 PRINT ″NAME ″;A$,″TEL ″;B$
50 END PARACT
```

⇩ Result of execution

```
NAME? OMRON
TEL ? 123-4567
NAME OMRON                  TEL 123-4567
```

As shown above, if a character string specified is enclosed in a pair of double quotation marks (`″`) before a variable name, the specified character string can be displayed when data is input. Note that the character string must be delimited by a semicolon (`;`) from the variable name.

Variable Name and Reserved Word

As described earlier, any name can be given to a variable. However, the names used for commands and functions must not be used as the names of variables. For example, `PRINT$` and `INPUT%` must not be used as variable names because these names are command names. The names that must not be used by the user are generically called reserved words or keywords. A list of reserved words are presented in *Appendix G*.

Operations

To process data through operations, program as follows by using operators and arithmetic functions:

To Perform Arithmetic Operation

To perform an operation, use arithmetic, relative, and logic operators described earlier.

```
10 PARACT 0
20 PRINT 10/3
30 PRINT 10%¥3%
40 PRINT 10%/3#
50 END PARACT
```

⇩ Result of execution

```
3.33333
3
3.333333333333333
```

The above program is to execute a division and display the result. The result differs depending on the data type (such as integer, single-precision real-number, and double-precision real-number).

On line 20, the operation is performed with single-precision real-numbers, and the result is rounded at the sixth digit. Therefore, five or less digits are displayed as the result.

On line 30, the operation is performed with integer values. Therefore, the data is truncated at the decimal point.

On line 40, a single-precision real-number variable and double-precision real-number variable are processed. If the precision of a variables differ, the higher precision takes precedence. In this case, therefore, the double precision takes precedence. Consequently, the data is rounded at the 16th digit, and displayed in 15 digits or less.

Priority of Operators

Each operator is assigned priority as shown in the following table. Relative operators are not assigned priority in respect to each other, and are executed in sequence starting from the left.

Priority	Operator	Operation	Classification
1	`( )`	Gives priority to ( )	Expression in ( )
2	Numeric function	Returns numeric value	Function
	Character function	Returns character string	
3	`^`	Exponential operation	Arithmetic operator
4	`–`	Negative sign	
5	`*, /`	Multiplication, division of real number	
6	`¥ or \`	Division of integer	
7	`MOD`	Remainder	
8	`+, –`	Addition, subtraction	
9	`=`	Equal to	Relative operator
	`<>, ><`	Not equal to	
	`<, >`	Less than, greater than	
	`<=, =<`	Less than or equal to	
	`>=, =>`	Greater than or equal to	
10	`NOT`	Negation	Logic operator
11	`AND`	Logical product	
12	`OR`	Logical sum	
13	`XOR`	Logical exclusive sum	
14	`IMP`	Implication	
15	`EQV`	Equivalence	
16	`=`	Substitutes right member into left member	Substitution

Character Operations

The only operation available for character variables and character constants is adding (coupling).

```
10 PARACT 0
20 A$ = ”BASIC”
30 B$ = ”UNIT”
40 PRINT A$ + B$
50 END PARACT
```

⇩ Result of execution

```
BASIC UNIT
```

Changing Program Flow

It may be necessary to change the flow of the program execution according to the result of an operation or conditions. The BASIC Unit can change the flow of program execution by using the following program control commands:

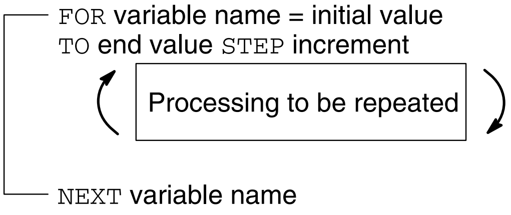
Instruction	Operation
`FOR TO STEP NEXT`	Repeatedly executes program enclosed by `FOR` and `NEXT` commands the specified number of times
`GOSUB RETURN`	Calls subroutine and returns from subroutine
`GOTO`	Unconditionally jumps to specified line number

`IF THEN ELSE/`
`IF GOTO ELSE`
	Selects line to be execution in accordance with result of relative or logic expression

`ON GOSUB/ON GOTO`	Branches to specified line
`WHILE WEND`	Repeatedly executes a series of commands until condition is satisfied

To Repeat the Same Process

Repeating the same processing is called a loop. Loop processing can be implemented by using the `FOR TO STEP NEXT` command. This command repeatedly executes the processing enclosed between `FOR` and `NEXT`.



```
FOR variable name = initial value
TO end value STEP increment
    Processing to be repeated
NEXT variable name
```

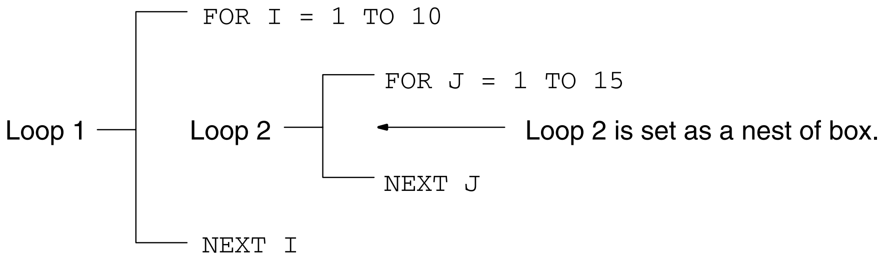
Loop processing can also be performed by using the `GOTO` command. However, if the number of times the processing to be repeated is fixed, the `FOR TO STEP NEXT` command is used. A sample program using this command is shown below.

```
90 'Calculate even sum and odd sum from 0 through 100.
100 PARACT 0
110 A% = 0
120 B% = 0
130 FOR I%=0 TO 100 . ...... Sum of even numbers and odd numbers
                             from 0 to 100
140     J%=I% MOD 2
150     IF J%=0 THEN A%=A% + I% ELSE B%=B% + I%
160 NEXT I%
170 PRINT "The even number sum 0 through 100?";A%
180 PRINT "The odd number sum 0 through 100?";B%
190 PRINT
200 END
210 END PARACT
```

⇩ Result of execution

```
The even number sum 0 through 100? 2550
The odd number sum 0 through 100? 2500
```

The `FOR TO STEP NEXT` command can also nest loops as follows:

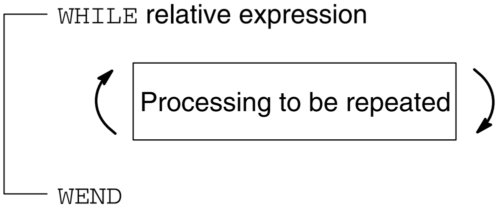


```
FOR I = 1 TO 10
    FOR J = 1 TO 15
    NEXT J
NEXT I
```

The variable name of `NEXT` can be omitted.

To Specify Conditions for Repetition

Instead of specifying the number of times for the `FOR TO STEP NEXT` command, it may be necessary to specify a condition under which repetition should be executed, for example, when the number of times the execution is to be repeated is not known such as when the processing is to be executed until `X = 0`. In this case, the `WHILE WEND` command is used as follows:



```
WHILE relative expression
    Processing to be repeated
WEND
```

Indefinite loop where relative expression is 1

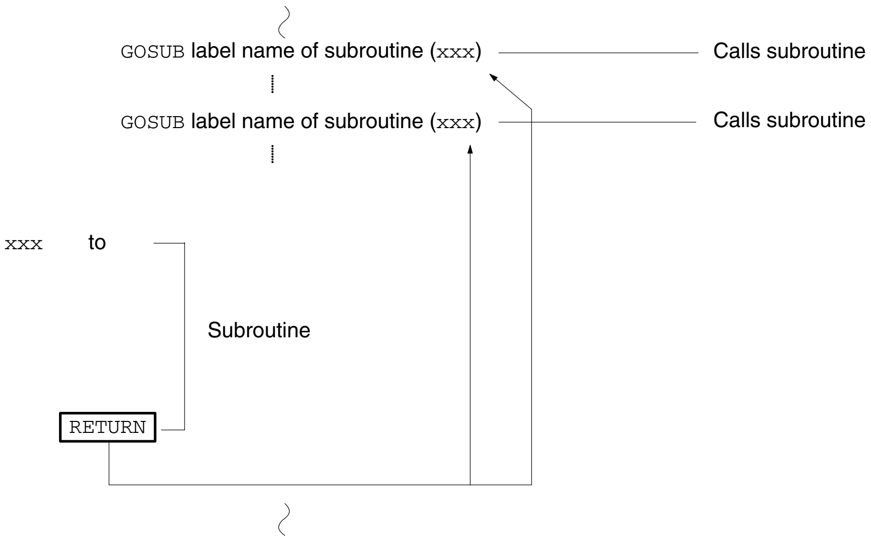
Example:

`WHILE 1` to `WEND`

The `WHILE  WEND` command executes the processing enclosed between `WHILE` and `WEND` until the condition specified by the relative expression is not satisfied (i.e., becomes false (`0`)).

To Execute the Same Processing at Different Locations

The `FOR TO STEP NEXT` command is used to repeat the same processing at the same location. However, it may be necessary to repeat the same processing at different locations, depending on the program. For example, if the same processing should be executed to various measured data, and if the same program is described for each measured data, the program becomes redundant. In this case, a subroutine is created and called as required by using the `GOSUB` and `RETURN` commands.


GOSUB label name of subroutine (xxx)
Calls subroutine
GOSUB label name of subroutine (xxx)
Calls subroutine
xxx to
Subroutine
RETURN

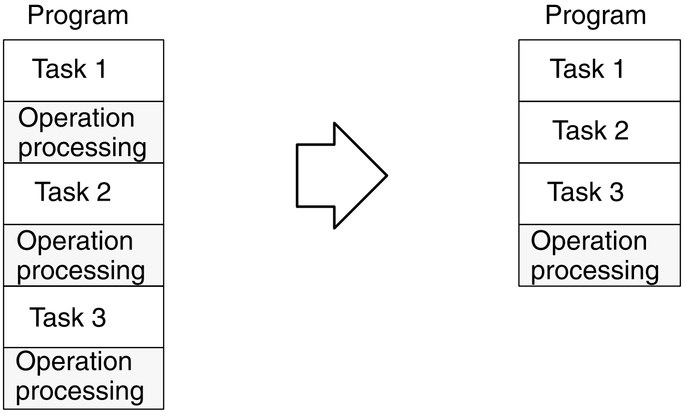
The following is a sample program using the `GOSUB RETURN` command.

```
Calling and returning from subroutine
100 PARACT 0
110 *START
120 PRINT ″Program calculating area of circle″
130 INPUT ″Input radius (to end, radius = 0)″;R%
140 IF R%=0 THEN END
150 GOSUB *CAL
160 PRINT ″Area of radius ″;R%;″ is ″;S!;″.″
170 GOTO *START
180 `
190 *CAL . .................. Subroutine calculating area of circle
200 S! = 3.14*R%*R%
210 RETURN . ................ End of subroutine by RETURN
220 END
230 END PARACT
```

⇩ Result of execution

```
Program calculating area of circle
Input radius (to end, radius = 0)?5
Area of radius 5 is 78.5
Program calculating area of circle
Input radius (to end, radius = 0)0
Ok
```

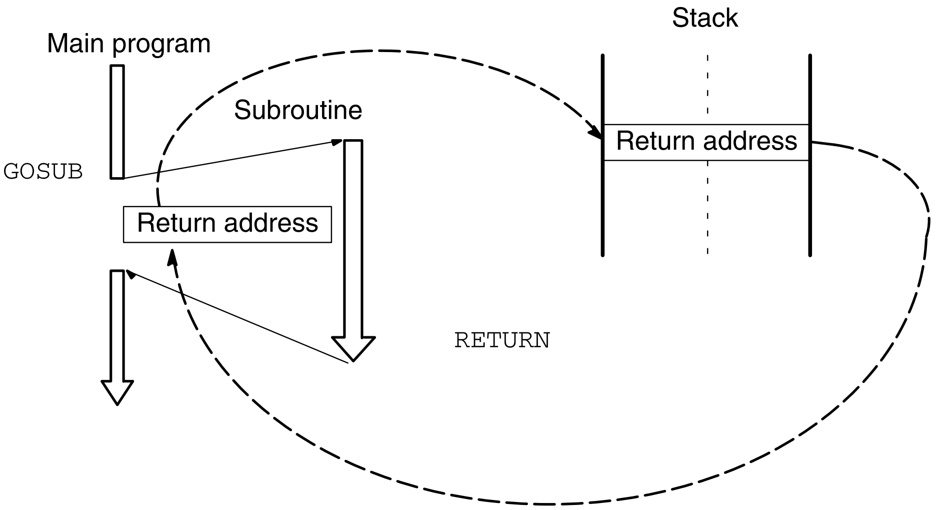
As shown above, by using subroutines the program can be divided into several modules so that it can be easy to see and develop and so that the same process can be executed from different locations.



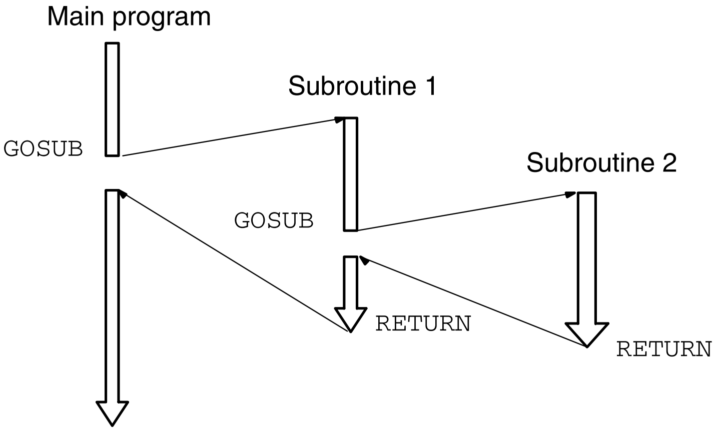
RETURN Command Ending Subroutine

When a subroutine is called, a return address is stored in a memory area so that the program execution can be returned to the main routine after the subroutine has been executed. This memory area is called a stack. To return the execution from a subroutine to the main routine, the return address is restored from the stack by the `RETURN` command.

Calling Subroutines



Only one level can be restored by the `RETURN` command. This means that to call another subroutine (`2`) from one subroutine (`1`) as shown below, the `RETURN` command is necessary for each subroutine.



Changing Processing According to Conditions

To select and execute processing according to the result of a relative expression, the `IF THEN ELSE` or `IF GOTO ELSE` command is used.

Example

$$\texttt{IF}\ \textit{relative expression}\ \texttt{THEN}\left\{\begin{matrix}\textit{line no.}\\ \textit{string}\\ \textit{label}\end{matrix}\right\}\ \texttt{ELSE}\left\{\begin{matrix}\textit{line no.}\\ \textit{string}\\ \textit{label}\end{matrix}\right\}$$

The following is a sample program using the `IF THEN ELSE` and `IF GOTO` commands.

```
Conditional branch operation
100 PARACT 0
110 *START
120 PRINT ”0: End 1: Sum 2: Difference 3: Product”
130 INPUT ”Select from menu”;I%
140 IF I%=0 THEN END . ..... When the input value is 0
150 IF I%>3 OR I%<0 THEN GOTO *EPROCESS
160 INPUT ”A”;A#
170 INPUT ”B”;B#
180 IF I%=1 THEN PRINT A#;”+”;B#;”=”;A#+B# ELSE *NEXT1
                          ....... When the input value is 1
190 GOTO *START
200 *NEXT1
210 IF I%=2 THEN PRINT A#;”–”;B#;”=”;A#–B# ELSE *NEXT2
                          ....... When the input value is 2
220 GOTO *START
230 *NEXT2
240 IF I%=3 THEN PRINT A#;”*”;B#;”=”;A#*B#
                                  When the input value is 3
250 GOTO *START
260 *EPROCESS . ............ When the input value is other than
                                  above
270 PRINT”     ***INPUT ERROR***”
280 GOTO *START
290 END
300 END PARACT
```

⇓ Result of execution

```
0: End 1: Sum 2: Difference 3: Product
Select from menu? 1
A ? 42
B ? 39
42 + 39 = 81
0: End 1: Sum 2: Difference 3: Product
Select from menu? 3
A ? 81
B ? 27
81 * 27 = 2187
0: End 1: Sum 2: Difference 3: Product
Select from menu? 0
```

Changing Processing According to Value of an Expression

To select a line number to which the execution is to branch according to the value of an expression, the `ON GOSUB` or `ON GOTO` command is used.

$$\texttt{ON}\ \textit{expression}\left\{\begin{matrix}\texttt{GOSUB}\\ \texttt{GOTO}\end{matrix}\right\}\left\{\begin{matrix}\textit{line no}\\ \textit{label}\end{matrix}\right\}\left\{\begin{matrix}\textit{line no}\\ \textit{label}\end{matrix}\right\}\left\{\begin{matrix}\textit{line no....}\\ \textit{label}\end{matrix}\right\}$$

Example

```
ON ABC GOSUB 1000, 2000, 3000, *SUB3, 5000
ON X1% GOTO *LAB1, 1500, *LAB3, *LAB4
```

If the value specified by the numeric expression is `1`, the execution branches to a line number specified first. If the value is `2`, the execution branches to a line number specified second. If the value is `3`, the execution branches to a line number specified third. A sample program using the `ON GOSUB` and `ON GOTO` commands is shown below.

```
Expression value branch
100 PARACT 0
110 *PRCS
120 PRINT ”(1: Sum 2: Difference 3: Product 4: End) ”;
130 INPUT ”Select number”;A%
140 IF A%<1 OR A%>4 THEN PRINT ”INPUT ERROR!!”: GOTO
*PRCS
150 IF A%=4 GOTO *E
160 PRINT ”Input 2 integers”
170 INPUT S1%
180 INPUT S2%
190 ON A% GOSUB *PLUS, *MINUS, *MULT
200 GOTO *PRCS
210 *E . ..................... When A% is 4
220 END
230 ’
240 *PLUS . ................. When A% is 1
250 PRINT S1%;”+”;S2%”=”;S1%+S2%
260 RETURN
270 ’
280 *MINUS . ................ When A% is 2
290 PRINT S1%;”–”;S2%”=”;S1%–S2%
300 RETURN
310 ’
320 *MULT . ................. When A% is 3
330 PRINT S1%;”*”;S2%”=”;S1%*S2%
340 RETURN
350 END PARACT
```

⇩ Result of execution

```
(1: Sum 2: Difference 3: Product 4: End) Select number?
                           1
Input 2 integers
? 12
? 23
12 + 23 = 35
(1: Sum 2: Difference 3: Product 4: End) Select number?
                           3
Input 2 integers
? 31
? 23
12 * 23 = 713
(1: Sum 2: Difference 3: Product 4: End) Select number?
                           4
```

`ON GOSUB` and `ON GOTO` functions are similar to each other. When `ON GOTO` is used, the destination will not be the same subroutine as `ON GOSUB`.

4-2 Writing and Entering Programs

4-2-1 Preparations

When developing or editing program, the uppercase and lowercase characters are not distinguished.

The uppercase and lowercase characters are also not distinguished in describing variable names, constant names, and array names. However, they are distinguished in character strings and comments.

When the program is displayed by the `LIST` command, it is displayed in uppercase characters.

Enable writing with the memory protect switch.

4-2-2 Program Storage Locations

When programs are input from a terminal, they are created in the user program source program area. Commands that read the program to the terminal, such as `LIST,` handle the program as source code.

When programs are executed they are automatically compiled into execution code and moved into the program execution area, requiring a certain amount of processing time. If the same program is executed a second time without alteration, this processing time is eliminated.

When programs are written to or read from EEPROM, the entire program area is copied as source code. Because the entire area is always copied, the size of the program does not affect the processing time.

When programs are written to or read from a Memory Card, only the program with the designated program number is transferred.

If the Memory Switches are set to specify automatic program transfer or automatic starting, the source code is loaded and recompiled each time the BASIC Unit is started. The Memory Switches can be set to transfer the program from a Memory Card or from EEPROM.

4-2-3 Allocating a Program Area

1, 2, 3...

1. Allocate areas to develop and store the program. Three areas are available, each of which separate programs can be developed and stored.

   `PGEN_2↵` . ... `2` is the program no. (`1` to `3`)
2. Confirm that the program area has been allocated.

   `PINF ↵`
3. The following information is displayed:

```
     No.     |     PNAME     |    S-CODE    |    E-CODE    |    GLOBAL    |    LOCAL
     1            TEST            41
    *2                             4
     3                             4
    FREE                         64207         112276
```

`*` on the left of No. indicates the area currently used.

4-2-4 Clearing Program Area

If a program previously developed or used remains in the allocated program area, clear the area. If the program is given a name, first delete the name by using the `PNAME` command, and clear the program area with the `NEW` command.

`PNAME␣""↵` . ... Deletes program name
`NEW↵` . ... Clears program area

If the program is not given a name, the program can be cleared only with the `NEW` command.

4-2-5 Generating Line Numbers

Generate line numbers automatically by using the AUTO command.

AUTO˽100,5↵ . ... 100 is the start line no. and 5 is the increment.

In this case, the program starts from line 100, and the line number is incremented by 5.

The specification of increment can be omitted, in which case, the program line number is incremented by 10.

AUTO˽100↵ . ... 100 is the start line no.

Both the start line number and increment can be omitted, in which case, the program with line number 10 is incremented by 10.

```
AUTO↵
```

In this case, the following messages are displayed. Input the program below these messages.

```
AUTO
Ok
10
```

To end generation of the line numbers, input CTRL+X, CTRL+C, or press carriage return after the line numbers have been displayed.

Line numbers can also be manually input one at a time without using the AUTO command.

4-2-6 Writing a Program

Input and write the program along with line numbers. Each line must end with a carriage return. A new line number will automatically be displayed. Continue inputting the program.

As an example, input the following program:

Key Input

```
PARACT˽0↵
A=3:B=4↵
FOR˽I=1˽TO˽3↵
A=A+B↵
PRINT˽A↵
NEXT˽I↵
END↵
END˽PARACT↵
```

Program

```
10 PARACT 0
20 A = 3 : B = 4
30 FOR I = 1 TO 3
40 A = A + B
50 PRINT A
60 NEXT I
70 END
80 END PARACT
```

Input CTRL+X, CTRL+C, or press carriage return to end generation of the line numbers.

Note The BASIC Unit is provided with a multitasking function by which more than one task (program) can be processed in parallel. The programs in the BASIC Unit should be written in units of tasks. PARACT 0 on line 10 in the above example program is a command indicating the beginning of a task. A task can be numbered 0 to 15. END PARACT on line 80 indicates the end of a task. For further information, refer to *6-2 Multitasking*.

4-2-7 Editing Programs

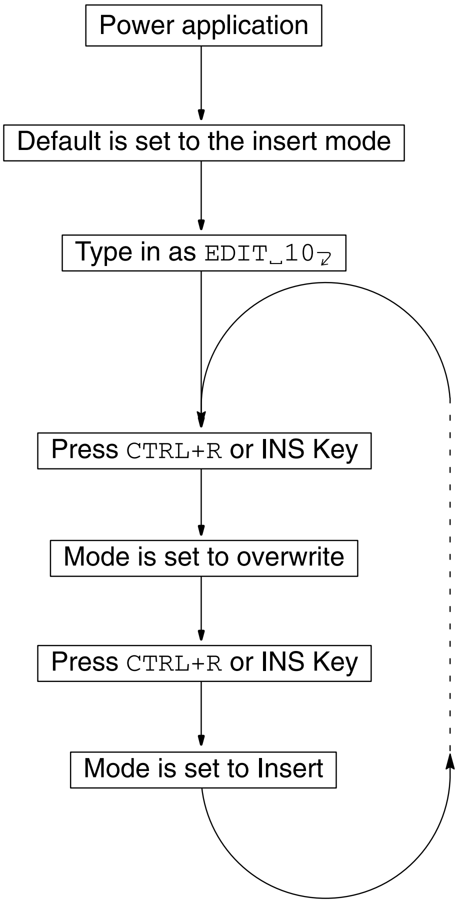
To edit a program, use the `EDIT` command. With this command, read and edit one line of the developed program at a time.

Changing Overwrite/Insert Mode

To edit programs, it is necessary to write characters over existing characters (overwrite mode), or insert new characters between existing characters (insert mode).

With BASIC, the mode is changed between the overwrite and insert modes with the memory switch (refer to *3-3-5 Memory Switch/Terminal Specification Setting Area*).

To change the mode, read the program with the `EDIT` command, and then input `CTRL+R` or INS Key. The mode is alternately changed each time `CTRL+R` or INS Key has been input. However, after one line has been edited, the setting of the memory switch is assumed.



1, 2, 3...

1. Turn the Power ON.
2. Insert mode by memory switch.
3. Type as `EDIT˽10↵`
4. Press `CTRL+R` or INS Key to change the mode to overwrite mode.
   or
   Press `CTRL+R` or INS Key again to change the mode to insert mode.

Editing Program in Overwrite Mode

The following procedure changes `I=1` on line `30` into `I=2`.

1, 2, 3...

1. Read the program.
   `EDIT˽30↵` . ... `30` is line no. to edit

2. The program of line 30 is displayed as follows. Move the cursor to the position of 1 by using the Left Key.

```
EDIT 30
Ok
30 FOR I = 1 TO 3
```

3. Input 2 followed by carriage return. This has edited the program.

```
EDIT 30
Ok
30 FOR I = 2 TO 3
```

Inserting Characters

The following procedure inserts I before PRINT A on line 50.

1, 2, 3...

1. Type EDIT␣50↵
2. The line to be edit is displayed as follows, then move the cursor to the position A with the Left Key.

```
EDIT 50
Ok
50 PRINT A
```

3. Change the mode from the overwrite to the insert mode by pressing CTRL + R or INS Key.
4. Insert I followed by carriage return.

```
EDIT 50
Ok
50 PRINT I,A
```

This has inserted I and edited the program.

4-2-8 Deleting in Programs

Deleting Characters

The following procedure deletes A+ of A=A+B on line 40 of the following program.

1, 2, 3...

1. Type EDIT␣40↵
2. The line to be deleted is displayed as follows, then move the cursor to the position B with the Left Key.

```
EDIT 40
Ok
40 A = A + B
```

[BS][BS][ ] or [DEL][DEL][ ]

Note The BS Key of the CVSS has the same function as the DEL Key. However, depending on the terminal, the character at the cursor position is deleted by the DEL Key.

```
EDIT 40
Ok
40 A = B
```

3. The program is edited when the carriage return is pressed.

Deleting Line

The following procedure deletes line 40 in the above sample program.

To do this, only input the line number or use the DELETE command.

40↵ or DELETE␣40↵

More than one line can also be specified at a time by specifying a range as follows:

DELETE␣120-150↵ . ... 120 is the beginning line no. which requires editing and 150 is the end line no. which requires editing

4-2-9 Copying in Programs

The following procedure copies program line 50 of the sample program below to line 55.

1, 2, 3...

1. Type EDIT_50↵

2. The line to be copied is displayed as follows, then move the cursor to the position `0` with the Left Key.

```
EDIT 50
Ok
50 PRINT I,A
```

3. Input the number of the line to which line `55` is to be copied.

```
Type 5↵
```

4. This has copied the contents of line `50` to line `55`.

```
EDIT 50
Ok
55 PRINT I,A
```

5. Next, `A` of line `50` is changed to `B`.
6. Move the cursor to the position of `A`. Input `B` and then carriage return.

```
EDIT 50
Ok
55 PRINT I,B
```

Any part of the program can be copied and edited. In addition to the above method, the line to be copied can be displayed by inputting, say `EDIT 50`, and a new line can be created by changing the program and line number at the same time and then pressing carriage return.

4-2-10 Merging Programs

The `MERGE` command can be used to add another program to the existing program. Be sure that the line numbers in the two programs do not overlap.

4-2-11 Changing Line Numbers

To put line numbers in order and assign new line numbers, the `RENUM` command is used.

```
RENUM↵
```

In the following example, the program is changed so that the first program line starts with `100` and the program lines are incremented by 10:

`RENUM␣100,10,10↵` . ... `100` is the new first line, left `10` is the old first line, and the right `10` is the increment

In the above example, the line numbers of the existing program are changed, so that the program starts with line `100`, instead of `10`, and the line numbers are incremented by 10. The program lines less than `10` are left untouched.

The line numbers specified for `GOTO` and `GOSUB` commands are automatically changed by the `RENUM` command. Therefore, it takes some time to complete the processing. Wait until the message `OK` is displayed.

4-2-12 Naming Programs

To identify the contents of a program, a program name is given to each program area by the `PNAME` command. When a program name has been given, the program cannot be erased by the `NEW` command.

`PNAME␣"SAMPLE"↵` . ... `SAMPLE` is the program name

When the `PNAME` command is executed without specifying a program name, the existing program name is deleted.

```
PNAME␣""↵
```

After this, a new program name can be given by another `PNAME` command.

4-2-13 Keys Operations in Editing

The following tables shows the keys that can be used in editing operations.

Key	Operation
Left Key	Moves the cursor to the left. This key is invalid while the cursor is at the beginning of a line.
Right Key	Moves the cursor to the right. This key is invalid while the cursor is at the end position of a line + 1 column.
Up Key	Moves the cursor up. If this key is pressed while the cursor is at the top line, the cursor moves to the leftmost position. If the cursor is at the leftmost position of the top line, this key is invalid.
Down Key	Moves the cursor down. If this key is pressed while the cursor is at the bottom line, the cursor moves to the bottom position of the + 1 column. With the cursor at this position, further pressing of this key is invalid.
Return	Executes editing functions and rewrites the program. After that, line feed and carriage return are performed.
SHIFT+HOME/CLR	Moves the cursor to the first position of a line. If the cursor is at the top line, this key is invalid. This function is not provided to the CVSS.
CTRL+H or BS	Deletes the character at the left of the cursor. This key is invalid with the cursor at the leftmost position of a line.
DEL	Deletes the character at the cursor position. This key is invalid while the cursor is at the rightmost position + 1.
CLR or CTRL+L	Clears the entire screen and moves the cursor to the home position (upper left). The processing under execution is canceled.
CTRL+E	Deletes the characters starting from the cursor position to the end of the line.
CTRL+R or INS	Switches between the overwrite mode and insert mode. Either the overwrite or insert mode is assumed according to the setting of the memory switch when editing is started by the `EDIT` command.
CTRL+X or CTRL+C	Terminates the execution of the `AUTO` or `EDIT` command.

Note
1. SHIFT+HOME CLR represents the pressing of the HOME CLR Key while holding down the SHIFT Key.
2. CTRL+L represents the pressing of the L Key while holding down the CTRL Key.
3. The edit function is executed when the carriage return has been pressed, and the program in the BASIC Unit will be rewritten accordingly.
4. The DEL Key and BS Key of CVSS are the same. In addition, HOME CLR and SHIFT+HOME CLR Keys are invalid.

4-3 Program Execution and Debugging

4-3-1 Preparations

The BASIC Unit is provided with commands that execute or debug the program.

To start or stop the program, the following commands are used:

```
RUN, STOP, BREAK
```

To resume program execution, or execute the program on a step-by-step basis, these commands are used:

```
CONT, STEP
```

To display the execution status of the program, these commands are used:

```
TRON, TROFF
```

By using the above commands, the program is debugged. As an example, the following sample program is debugged.

```
10 PARACT 0
20 A = 3 : B = 4
30 FOR I = 1 TO 3
40 A = A + B
50 PRINT A
60 NEXT I
70 END
80 END PARACT
```

Execution can also be stopped from the keyboard by inputting CTRL-X or CTRL-C. When CTRL-X is input, all execution, including I/O processing, will be aborted immediately and “Quit in ...” will be displayed. `STEP` and `CONT` cannot be used after aborting execution with CTRL-X. When CTRL-C is input, execution is stopped as soon as the current instruction has been executed. If “Break in ...” is displayed, `STEP` and `CONT` can be used. If “Quit in ...” is displayed, `STEP` and `CONT` cannot be used.

Note Data received while program execution is stopped may not be retrievable after `CONT` is executed. To avoid this problem, make sure that data has been received and jump to the address defined with `ON PC` before setting the `BREAK` point.

4-3-2 Execution

1, 2, 3...

1. To execute the program, use the `RUN` command.

   ```
   RUN,ERASE↵
   ```
2. Specify `ERASE` to clear the non-volatile variables.
3. Display and confirm the sample program. (`LIST↵`)
4. Execute the program. `ERASE` can be omitted.
5. Type `RUN` and press `ENTER` to execute the program.

   ```
   LIST
   10 PARACT 0
   20 A = 3 : B = 4
   30 FOR I = 1 TO 3
   40 A = A + B
   50 PRINT A
   60 NEXT I
   70 END
   80 END PARACT
   Ok
   RUN
    7
    11
    15
   Ok
   ```
6. If an error is found on a line of the program while the program is executed, the execution is stopped at that point, and the line number and an error message identifying the nature of the error are displayed. To correct the error, display the line by using the `EDIT` command, input the correct command, and press the carriage return. Then input again and execute the program by the `RUN` command. If another error message is displayed, correct the program in the same manner and execute it again.

Displaying Execution Result Commands can be input or directly executed from the terminal without assigning line numbers. The values of variables immediately after the program has been executed can be displayed and checked by the `PRINT` command.

Type `PRINT␣A,I↵`

```
PRINT A, I
 15                        4
Ok
```

4-3-3 Stopping and Resuming Execution

STOP Command

The STOP command is inserted in the program in advance. When the program is executed and the STOP command is reached, the program is stopped. In the following example, the STOP command is placed at line 55.

55˽STOP↵ .... 55 is the line number into which STOP command is inserted.

1, 2, 3...

1. Execute the program.

   ```
   RUN↵
   ```

2. The program is stopped at line 55 by the STOP command and the line number (55) is displayed.

   ```
   55 STOP
   RUN
    7
   Stop in 55
   ```

CONT Command

To resume the program stopped by the STOP command, the CONT command is used.

```
CONT↵
```

Erase line 55. (55↵)

```
CONT
 11
Stop in 55
CONT
 15
Stop in 55
CONT
Ok
55
```

BREAK Command

The program execution can also be stopped by the BREAK command. With this command, the line where execution is to be stopped is specified. This method stops the execution without modifying the program. Up to 10 lines, where the execution is to be stopped, can be specified.

BREAK˽20,70↵ .... 20 and 70 are the line nos. where the program is required to break

1, 2, 3...

1. First specify a break line execute the program. (RUN↵)
2. Displays a message and stops the program before executing line 20. Then resumes execution. (CONT↵)
3. Displays a message and stops the program before executing line 70. Then resumes execution. (CONT↵)

   ```
   BREAK 20, 70
   Ok
   RUN
   Break in 20
   CONT
    7
    11
    15
   Break in 70
   CONT
   Ok
   ```

To cancel the effect of the BREAK command, use the BREAK DELETE command.

BREAK˽DELETE˽20↵ .... 20 is the break line no

BREAK˽DELETE˽ALL↵ .... ALL means all break lines are deleted

BREAK↵ .... BREAK means the set break line is displayed

4-3-4 Step Execution

After stopping the execution of the program, the program can be executed one line at a time by the `STEP` command.

```
STEP↵
```

1, 2, 3...

1. First, specify a break point and execute the program.
   ```
   BREAK 20
   RUN
   ```
2. The program execution is stopped at line `20`. Then four steps of the program are executed on a step-by-step basis. (`STEP↵`)
3. `7` is displayed.
4. Display the contents of `A` and `B` by the `PRINT` command.
   ```
   PRINT␣A,B↵
   ```
5. Resume execution by the `CONT` command.
   ```
   CONT↵
   ```
6. Clear the break point. (`BREAK␣DELETE␣ALL↵`)

```
BREAK 20
Ok
RUN
Break in 20
STEP
STEP
STEP
STEP
STEP
 7
PRINT A, B
 7                       4
CONT
 11
 15
Ok
BREAK DELETE ALL
Ok
```

Note Since line `20` is a multi-statement, the `STEP` command must be executed two times to execute this line. Also, because the execution code (E code) is an intermediate code, sometimes one `STEP` command will execute two consecutive lines and sometimes two `STEP` commands will be required to execute one line.

4-3-5 Tracing Program Execution

The line numbers of the program under execution can be displayed in the order of execution by the `TRON` command. This function is canceled by the `TROFF` command.

```
TRON␣1↵ . ....................  1 is the task no.
TROFF␣ALL↵ . ................  ALL is the task no.
```

If `ALL` is specified instead of the task number, the line numbers of all the tasks are traced (or tracing is canceled). If neither a task number nor `ALL` is specified, the current task is traced.

1, 2, 3...

1. Starts tracing. (`TRON↵`)
2. Run the program (`RUN↵`). Then displays line number under execution.
3. Cancels tracing (`TROFF↵`). Then starts tracing task 1.

```
TRON
Ok
RUN
[10][20][30][40][50] 7
[60][40][50] 11
```

```
[60][40][50] 15
[60][70] Ok
TROFF
Ok
TRON 1
Ok
RUN
 7
 11
 15
Ok
```

When `@` is input, the number of the task under execution can be checked.

4-4 Saving and Loading Programs

The program can be saved to/loaded from the following three devices:

EEPROM If the BASIC Unit is provided with EEPROM, all the three programs in the source code area can be saved to or loaded from the EEPROM.

Floppy Disk Each program can be saved to or loaded from a floppy disk by the save/load function of the computer with terminal mode connected to the BASIC Unit.

Memory Cards If the CPU Unit is equipped with a memory card, the program can be saved to or loaded from the memory card.

Note To load the program, set the memory protect pin of the DIP switch to the OFF position. Otherwise, an error will occur.

Pin no.	DIP switch setting
1	Terminal resistor (OFF)
2	Not used (OFF)
3	Memory switch (invalid) (OFF)
4	Memory protect (OFF)

4-4-1 EEPROM

With the BASIC Unit with EEPROM (CV500-BSC21/41/61), the program can be saved to or loaded from the EEPROM. In addition, the program in the EEPROM can be verified.

To save the program, use the `ROMSAVE` command.

When this command is executed, the contents of all the user program areas are written to the EEPROM.

```
ROMSAVE↵
```

To read (load) the program written to the EEPROM to the user program areas, use the `ROMLOAD` command.

```
ROMLOAD↵
```

All the contents of the EEPROM are read to all the user program areas. Consequently, the current contents of the user program areas are erased.

To compare the contents of EEPROM and those of the user program areas, use the `ROMVERIFY` command.

```
ROMVERIFY↵
```

This command verifies the contents of the user program areas with those of EEPROM. If a discrepancy is found, the message “`VERIFY ERROR`” is displayed.

Note If the above three commands are executed with the BASIC Unit not equipped with EEPROM, an error occurs.

4-4-2 Memory Cards

The program of the BASIC Unit can be saved to or loaded from the memory card of the CPU Unit. The memory card must be formatted in advance by a Memory Card Writer.

To save the program to the memory card, use the SAVE command.

SAVE␣″0:SAMPLE″↲ . ... 0 is the device name (0: memory card),
and SAMPLE is the file name

The contents of the program area currently used are saved to the memory card under a specified file name in text format (in displayed image). If the specified file name already exists in the memory card, the contents of the existing file are changed.

To load the contents of the memory card to the program area of the BASIC Unit, use the LOAD or MERGE command.

To Load New Program:

LOAD␣″0:SAMPLE″↲ . ... 0 is the device name (0: memory card),
and SAMPLE is the file name

The LOAD command clears the currently used area and loads the program to that area.

To Load Additional Program:

MERGE␣″0:SAMPLE″↲ . ... 0 is the device name (0: memory card),
and SAMPLE is the file name

The MERGE command loads an additional portion of a program to the area currently used.

Only files of text format can be loaded. If a file of specified file name does not exist on the memory card, an error message is displayed. If the line numbers of the program loaded by the MERGE command overlap the line numbers of the existing program, the line numbers of the newly loaded program take precedence.

4-4-3 Saving and Loading via Personal Computers

Personal computer editing operations can be used to create source programs in the BASIC Unit or transferred programs between the personal computer and the BASIC Unit. Connect the personal computer to the terminal port of the BASIC Unit as proceed as described next.

Use the following program to load programs from the personal computer to the BASIC Unit.

1, 2, 3...

1. Use an editor in the personal computer to create a source program consisting of BASIC Unit commands as a file in memory or on a disk.
2. Place the BASIC Unit into BASIC mode so that input from the terminal is enabled.
3. Create a program in the personal computer to do the following.
   a) Send the LOAD command to the BASIC Unit to initialize reception.
   b) Send the program created in step 1. one line at a time to the BASIC Unit.
   c) Send the file end code after the last line of the source program has been sent.

Use the following program to save programs from the BASIC Unit to the personal computer.

1, 2, 3...

1. Place the BASIC Unit into BASIC mode so that input from the terminal is enabled.
2. Create a program in the personal computer to do the following.
   a) Send the LIST command to the BASIC Unit to have the BASIC Unit output one line at a time of a source program.
   b) Store each line of the program being read into the personal computer into a file in memory or on a disk.
   c) Detect "OK" in the transmission from the BASIC Unit to determine the end of the transmission.

The following sample program can be used as reference in program development. If there are problems with loading using this program, increase the time on line 340.

```
100 . ********                                                        ********
110 . ********               CPU Unit-BASIC UNIT UPLOAD/DOWNLOAD       ********
120 . ********                                                        ********
130 . >>This program uploads/downloads programs created on the CPU Unit from/to
140 .   the BASIC Unit.
150 . *SELECT
160 . INPUT "SELECT L(LOAD(Computer->BSC))/S(SAVE(BSC->Computer)) ---";k$
170 . IF K$="L" GOTO *PCBSC
180 . IF K$="K" GOTO *BSCPC
190 . GOTO *SELECT
200 ..
210 .  ---- DOWNLOAD (Computer to Basic Unit) ----
220 .   <<<Caution>>>
230 .    If a program name is registered, use PNAME "" to delete it in advance.
240 . *PCBSC
250 . ON ERROR GOTO *EEE                                Breaks at file end.
260 . OPEN "COM:N83XN" AS #1                            8 bits, no parity, 2 stop bits
270 . OPEN "BSCTEMP.BAS" FOR INPUT AS #2                Opens source file.
280 . B$="LOAD #1,"+CHR$(&H22)+"COMU:"+CHR$(&H22)       Preparing for LOAD
290 . *CMND
300 .  PRINT #1,BS                                      Sends =
310 .  LINE INPUT #1,A$                                 Reads command echoback
320 .  IF A$<>B$ GO TO *CMND                            Checks =
330 . *LOOP
340 .  FOR TT=1 TO 100 : NEXT                           Waits for BASIC load processing
350 .  LINE INPUT #2,A$                                 Reads one line of source program
360 .  PRINT #1,A$+CHR$(13);
370 .  GO TO *LOOP                                      Loops until end of file is reached
380 . *EEE
390 .  PRINT #1,CHR$(&H1A)                              Sends file end code
400 .  LINE INPUT #1,A$                                 Reads "OK" echoback
410 .  CLOSE #1 : CLOSE #2
420 . END
430 .
440 .  ---- UPLOAD (Basic Unit to Computer) ---
450 . *BSCPC
460 . OPEN "COM:N83XN" AS #1                            8 bits, no parity, 2 stop bits
470 . OPEN "BSCTEMP.BAS" FOR OUTPUT AS #2               Opens file to save in
480 . B$="LIST"                                         Preparing for LIST
490 . PRINT #1,B$                                       Sends =
500 . LINE INPUT #1,A$                                  Reads command echoback
510 . *LOOPS
520 .  LINE INPUT #1,A$                                 Reads one line from file
530 .  IF A$="OK" THEN *FINAL                           Checks for end
540 .  PRINT #2,A$                                      Sends one line
550 .  GOTO *LOOPS                                      Loops until end of file is reached
560 . *FINAL
570 .  CLOSE #1 : CLOSE #2                              CLOSE
580 . END
```

SECTION 5
Data and Files

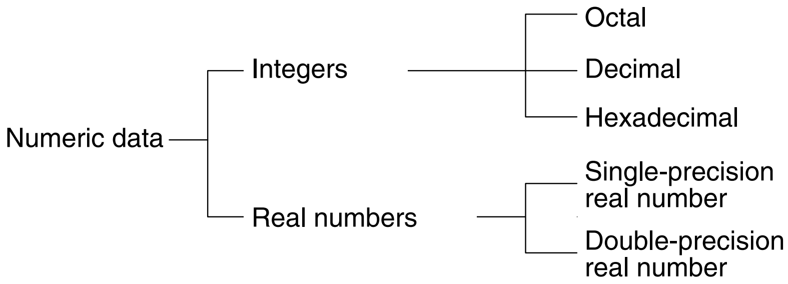
This section provides information on data management and operations for the BASIC Units.

5-1 Data Operations

5-1-1 Handling Numeric Data

Types of Numeric Data

The numeric data the BASIC Unit handles is classified into integers and real numbers, as shown below, and can be expressed in various formats.



Octal Format

In this format, the numeric data is expressed in numerals 0 through 7 with `&O` or `&` prefixed. With the BASIC Unit, up to 5 digits of octal numbers can be expressed (from 0 to 77777).

Examples:

```
&O123
&256
```

Decimal Format

The BASIC Unit can handle decimal integers from –32768 to +32767.

Examples:

```
-256
 123%
```

Hexadecimal Format

In this format, the numeric data is expressed in numerals 0 through 9 and alphabetic characters `A` through `F` with `&H` prefixed. The BASIC Unit can represent hexadecimal numbers from 0 to FFFF.

Examples:

```
&H123
&H2E4F
```

Single-precision Real Numbers

The numeric data of this type is expressed using up to 5 digits with the sixth digit rounded. The range of the numeric data is from –3.4E+38 to 3.4E+38 for variables and –3.40282E+38 to 3.40282E+38 for arithmetic results. The representation format of single-precision real number can be any of the following:

- Number of six digits or less
- Exponential format with `E`
- With `!` suffixed to numeral

Examples:

```
 3.21
-1.23E + 4
345.2!
```

Double-precision Real Numbers

The numeric data of this type is expressed with up to 15 digits with the 16th digit rounded. The range of the numeric data is therefore from –1.701411834604692D+307 to 1.701411834604692D+307. The representation format of double-precision real numbers can be any of the following:

- Number of seven digits or less
- Exponential format with `D`
- With `#` suffixed to numeral

Examples:

```
9876543210
        -1.2345D - 12
        345.21 #
      12345.6789098
```

Exponential Format

When a number with many digits is handled, writing many 0s is cumbersome and can cause errors in the program. Therefore, the BASIC Unit employs an exponential format to handle a number having many 0s. For example, to express number 12300000, it is simpler and easier to read by expressing it using an exponent, as follows:

= 1.23 x 10000000
= 1.23 x 10^7

With the BASIC Unit, this exponent is represented as follows:

`= 1.23E+7` ... (single-precision real numbers)
or,
`= 1.23D+7` ... (double-precision real numbers)

Here, `1.23` is called the mantissa, while `E+7` and `D+7` are called the exponents. This representation method is called exponential format. The relationship among the numbers and units of the exponent are as follows:

Indication	Number	Name	Symbol
`E-3`	0.0001	milli	m
`E-6`	0.0000001	micro	µ
`E-9`	0.0000000001	nano	n
`E-12`	0.0000000000001	pico	p
`E+3`	1000	kilo	K
`E+6`	1000000	mega	M
`E+9`	1000000000	giga	G
`E+12`	1000000000000	tera	T

Number Precision and Type Conversion

Usually, the same type of numbers are operated (for example, an integer is operated with an integer, and a single-precision number is operated with a single-precision number). On some occasions, however, various types such as integer, single-precision real numbers, and double-precision real numbers must be mixed when executing an operation. On these occasions, the type having the highest precision takes precedence and the other types are converted into that type.

```
10 PARACT 0
20 PRINT 10%¥3% . .......... Integers
30 PRINT 10%/3! . .......... Integer and single-precision real num-
                             bers
40 PRINT 10!/3# . .......... Single-precision real-numbers and
                             double-precision real-numbers
50 END PARACT
```

⇩ Result of execution

```
3
3.33333
3.333333333333333
```

To determine the types of variables at the beginning of a program, the `DEFINT`, `SNG`, `DBL`, or `STR` command is used.

Examples:

`DEFINT A`	Specifies variable name starting with `A` as integers
`DEFSNG B`	Specifies variable name starting with `B` as single-precision number
`DEFDBL C`	Specifies variable name starting with `C` as double-precision number
`DEFSTR D`	Specifies variable name starting with `D` as character

To perform batch conversion of variable types from `A` to `D`, the range of type must be specified as follows, by using a hyphen:

Example:

`DEFSNG B – E`	Converts all types of variables with names starting with `B`, `C`, `D`, or `E` into single-precision number.

Numeric Operation Functions

The BASIC Unit supports the following functions to execute arithmetic operations based on numeric data.

List of Functions Executing Arithmetic Operations

Function	Meaning
`ABS`	Gives absolute value
`ACOS`	Gives arc cosine
`ASIN`	Gives arc sine
`ATN`	Gives arc tangent
`CDBL`	Converts into double-precision real number
`CINT`	Converts into integer
`COS`	Gives cosine
`CSNG`	Converts single-precision real number
`CVI/CVS/CVD`	Converts character string into numeric value
`EXP`	Gives value of exponential function
`FIX`	Gives integer
`INT`	Truncates at decimal point
`LOG`	Gives natural logarithm
`RND`	Gives random number
`SGN`	Gives sign
`SIN`	Gives sine
`SQR`	Gives square root
`TAN`	Gives tangent

5-1-2 Handling Character Data

The BASIC Unit also handles character data in addition to numeric data. When characters are handled as data, various commands and functions that manipulate character strings in various manners are necessary. The BASIC Unit therefore supports the following character string manipulation commands and functions.

The length of a character variable is fixed, and the default length is 18 characters unless otherwise specified. If more than 19 characters are specified as a character variable, the excess characters are ignored, but no error occurs. To handle more than 19 characters, the necessary length (i.e., number of characters) must be declared by the `DIM` or `OPTION LENGTH` command. The maximum number of characters in a string is 538.

Functions Retrieving Part of Character String

For functions that retrieve the specified number of characters from a specified location of a character string, or that check the number of characters of a character string, `LEFT$`, `RIGHT$`, `MID$`, and `LEN` are used.

To Check Character String Length

The `LEN` function checks the number of characters of a character string.

Example:

`L = LEN(A$)` . ... Stores the number of characters of string `A$` in `L`

To Retrieve n Characters from Ends of Character String

The `LEFT$` and `RIGHT$` functions retrieve n characters from the left and right ends of a character string, respectively.

Examples:

`A$ = LEFT$(”BASIC UNIT”,2)` Stores the left two characters “`BA`” from “`BASIC UNIT`” in `A$`

`A$ = RIGHT$(”BASIC UNIT”,5)`

Stores the right five characters “`˽UNIT`” from “`BASIC UNIT`” in `A$`

To Retrieve Characters from Character String

The `MID$` function retrieves the specified number of characters from the specified position of a character string.

Example:

`A$ = MID$(”BASIC UNIT”,7,3)`

Stores three characters “`UNI`” from “`BASIC UNIT`” starting from the seventh character position from the left in `A$`

Functions Searching a Character String

To Search and Return from Character String

The `INSTR` function searches a specified character string from a character string and returns its position.

Example:

`X = INSTR(”ABCDEFGH”,”E”)`

Checks the position of “`E`” in “`ABCDEFGH`” and stores the result, 5, into `X`

If the specified character string is not found, `0` is returned. In the above example, even if more than one “`E`” exists, only “`E`” at the leftmost position in the character string can be found because the character string is searched starting from the left. To search a character string at a specified position, therefore, a position from which the search is to be started must be specified.

Example:

`X = INSTR(6,”ABCDEFGHE”,”E”)`

Searches for “`E`” after the `6`th character position in “`ABCDEFGHE`”. If “`E`” is found, its position (in this case, 9th position) is stored into variable `X`

Functions Creating Character String Consisting of Identical Characters

The STRING$ or SPACE$ function is used to arrange identical characters or spaces.

To Arrange Identical Characters

The STRING$ function is used to arrange as many of the identical characters as required.

Example:

A$ = STRING$ (10,"*") . .... Stores character string consisting of 10 *,"**********", into A$

The maximum number of characters that can be arranged by this function is 538. If two or more different characters are specified, only the one specified first is assumed.

To Arrange More Than One Space

The SPACE$ function arranges as many spaces as required.

Example:

A$ = SPACE$(10) . ... Stores 10 spaces into A$

Commands Rewriting Part of Character String

To Change Only Part of Character String

To change only part of a character string, the MID$ command is used. Note that the MID$ command is different from the MID$ function in use.

MID$(A$,X,Y)=B$ . ... A$ is the character string rewritten, X is the position of character to or rewritten, Y is the number of characters rewritten, and B$ is the contents to be replaced (character string)

Y characters from Xth position of A$ are replaced by Y characters of B$.

The number of characters to be rewritten can be omitted. In this case, the number of characters specified in the left member is assumed. As an example, the following program replaces character string "ABCDE" with "OMRON".

```
10 PARACT 0
20 A$ = "ABCDE"
30 PRINT A$
40 B$ = "OMRON"
50 MID$ (A$,1) = B$
60 PRINT A$
70 END PARACT
```

⇩ Result of execution

```
ABCDE
OMRON
```

Functions Converting Numeric Value and Character String

To Convert Numeric Value into Character String

On some occasions, programming is easier if numeric values are handled as characters. For example, numeric values are easier to see if each three digits are delimited by a comma (,) as 123,000, or if 0s are prefixed to unify the number of digits, as 0012, 0123, and 0001. To perform processing of this kind, it is necessary to convert numeric values into character strings. The STR$ function is used for this purpose.

Example:

A$ = STR$(1234) . ... Stores the character string "1234" in A$

To Convert Character String into Numeric Value

To convert a character string into a numeric value, the VAL function is used.

Example:

A = VAL(1234) . ... Stores character data "1234" in A as numeric value 1234

5-1-3 Handling Large Quantities of Data

When handling a large quantity of data in a program, programming is extremely difficult if separate variable is used for each data item. To facilitate programming, therefore, variables called array variables are used. Array variables can specify more than one data item under one variable name, and are classified into one-dimensional array variables and multi-dimensional (two-dimensional, three-dimensional, and so on) variables.

One-dimensional Array and Multi-dimensional Array

An array variable consists of a variable name followed by a numeric value enclosed in ( ). This numeric value is called a subscript. An array variable having only one subscript is called a one-dimensional array. An array having two subscripts is called a two-dimensional array, and the one having three subscripts is called a three-dimensional array. Generally, an array variable having two or more subscripts is called a multi-dimensional array.

Examples:

X = A(5) .... Stores A(5) of one-dimensional array into X

Y = B(3,3) .... Stores B(3,3) of two-dimensional array into Y

For example, (12, 54, 33, 95, 28) can be represented by one array variable A as A(0), A(1), A(2), A(3), and A(4).

Subscript	0	1	2	3	4
Data	12	54	33	95	28

The two-dimensional array is used to represent the data that can be represented by rows and columns. For example, suppose that three parameters, voltage, current, and temperature, are each measured three times. The first measured data set of voltage, current, and temperature, (3, 5, 20), the second data set (2, 4, 21), and third data set (4, 6, 25) can be represented by a two-dimensional array as follows:

Subscript	0 (voltage)	1 (current)	2 (temperature)
0 (first time)	3	5	20
1 (second time)	2	4	21
2 (third time)	4	6	25

Assuming the array variable name to be A, the second measured data of voltage is specified as A(1,0), and the third measured data of temperature is specified as A(2,2).

Use of Array Variables

Declaring Array Variables

When using an array variable with the BASIC Unit, first declare the array variable by using the DIM command. The number of array elements that can be specified by one array variable is not restricted, but limited by the memory capacity of the variable area.

Example:

DIM A(1,3) .... Allocates 2 x 4 = 8 array elements as array variable A (two-dimensional array) of numeric data.

⇩ Result of execution

A(0,0)	A(0,1)	A(0,2)	A(0,3)
A(1,0)	A(1,1)	A(1,2)	A(1,3)

The above array elements are allocated in memory.

Setting Lower-limit Value of Subscript

Usually, the subscript of an array starts from 0. However, it can be specified to start from 1 by using the `OPTION BASE` command.

Example:

```
OPTION BASE 1
DIM A (2,3)
```

⇩ Result of execution

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)

2 x 3 = 6 array elements are allocated in memory. The declaration made by the `OPTION BASE` command cannot be changed once it has been made.

Array of Character Variables

A character array of up to 538 characters can be handled by using a character variable name.

`DIM A$(50) 538` .... Defines one-dimensional character array variable having maximum character storage area of `538` characters

Here, `A$(50)` is the character variable array name, and `538` is the maximum number of characters.

5-1-4 Handling Time Data

The BASIC Unit also supports functions that handle time data such as dates and hours.

To Check Current Time

To check the current time, the `TIME$` function is used.

```
10 PARACT 0
20 T$ = TIME$
30 HH$ = LEFT$ (T$, 2)
40 MM$ = MID$ (T$, 4, 2)
50 SS$ = RIGHT$ (T$, 2)
60 PRINT″Current time is ″;HH$;″:″;MM$;″:″;SS$;″.″
70 END
80 END PARACT
```

⇩ Result of execution

```
Current time is 23:07:26.
```

DATE$ Function

This function is used to check the current date.

```
10 PARACT 0
20 D$ = DATE$
30 YY$ = RIGHT$ (D$, 2)
40 DD$ = MID$ (D$, 4, 2)
50 MM$ = LEFT$ (D$, 2)
60 PRINT″Today is ″;MM$;″–″;DD$;″–″;YY$;″.″
70 END
80 END PARACT
```

⇩ Result of execution

```
Today is 07–26–91.
```

5-1-5 Data Input/Output in Program

To read data by a program, the INPUT command or substitution statement such as A = B is used. However, if a large quantity of data is to be handled or if the input data is known in advance, describing the INPUT command or substitution statement is inefficient and not necessary. To simplify data input/output in the program, the READ and DATA commands are used.

To Simplify Data Input/Output in Program

The DATA command reads data (constants) continuously to the program. These data items are automatically read to specified variables by the READ command. A sample program using the DATA and READ commands is shown below.

```
10 PARACT 0
20 READ A$; B$ . ........... Reads character data from data state-
                             ment on line 70
30 READ C, D, E . .......... Reads numeric data from data state-
                             ment on line 80
40 PRINT A$; B$
50 PRINT C; D; E
60 END
70 DATA ”BASIC”,”UNIT” . ... Character data known in advance
80 DATA 10, 16, 1990 . ..... Numeric data known in advance
90 END PARACT
```

⇩ Result of execution

```
BASIC UNIT
10 16 1990
```

The READ and DATA commands are always used in pairs. The DATA command can be described anywhere in the program because it is a non-executable statement. As many DATA commands as required can be used in one program.

An error occurs if

- The number of character constants of the DATA command is read by the numeric variable of the READ command (the numeric constant of the DATA command can be read as a character string by the character variable of the READ command),
- The data of the DATA command has run out while the READ command is executed, or
- DATA of another task has been read.

To Read DATA Command Using READ Command

If more than one READ and DATA command exists, data is read in the execution sequence of the program. However, it may be necessary for the READ command to read the DATA command on specified line. In this case, the RESTORE command is used. Note, however, that the DATA command of another task must not be specified. A sample program using the RESTORE command is shown below.

```
10 PARACT 0
20 RESTORE 100
30 READ A$, B$ . ........... Reads character data from date state-
                             ment on line 100
40 RESTORE 90
50 READ C, D, E . .......... Reads numeric data from data state-
                             ment on line 90
60 PRINT A$; B$
70 PRINT C; D; E
80 END
90 DATA 10, 16, 1990 . ..... Numeric data known in advance
100 DATA ”BASIC”, UNIT . ... Character data known in advance
110 END PARACT
```

⇩ Result of execution

```
BASIC UNIT
10 16 1990
```

5-2 File Operations

5-2-1 Files

A BASIC Unit file manages a cluster of program information and data. Files are classified by the contents or access mode as seen in the following:

Data File and Program File

Files can be classified by contents into program files and data files.

Program File

A program file is a BASIC source program file created by using the editing commands of the BASIC Unit. This file can be read from or written to the memory card of the CPU Unit by the `SAVE`, `LOAD`, or `VERIFY` commands.

```
10 OPEN...
20 PRINT...
30 IF...THEN...
```

Data File

A data file is a file recording the data used by a program file. This file is opened by the OPEN command, and read or written by the `PRINT`, `WRITE`, `INPUT`, `PUT`, or `GET` commands. It is closed by the `CLOSE` command.

```
100 30 70
 60 11 23
 74 49 86
```

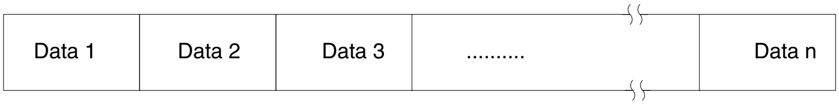
Note The BASIC unit reads or writes the memory card of the CPU Unit as program and data files.

Sequential/Random Access File

Files can be classified by data access mode into two types: sequential file and random access file.

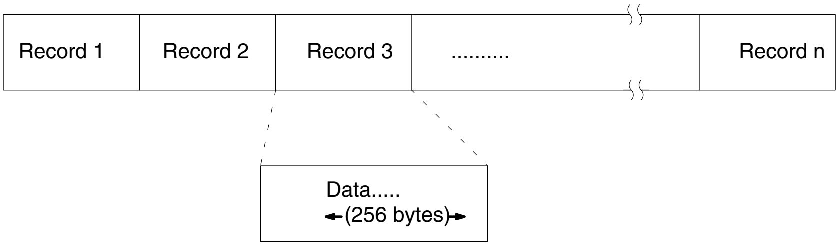
Sequential File Access

A sequential access file is sequentially read or written starting from the beginning of the file and is also known as a consecutive file.



Random Access File

A random access file is read or written in units called records (one record is fixed to 256 bytes with the BASIC Unit). This file can be accessed more quickly than the sequential file.

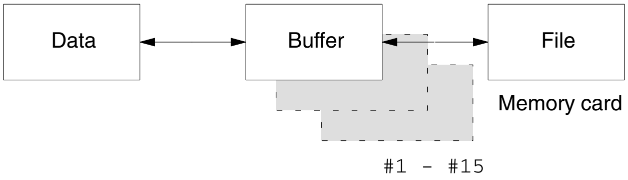


The sequential access file and random access file each have their own features, seen as follows:

Feature	Sequential access file	Random access file
Data access	Can only be read from beginning	Can be read/written starting from any location (in record units)
Data length	Can be changed freely	Fixed
Changing data	Entire file must be updated	Can be changed in record units
Adding data	Data is written at the end of file	Can be written to any position
Data type	Numeric data, character data	Numeric data must be converted into character data

5-2-2 Manipulating Data Files

To input/output a file, a memory area called a buffer is used to temporarily store data. The number assigned to this buffer is called a file number. One buffer corresponds to one file, and therefore, one buffer cannot be used by more than one file. The file numbers that can be used are from `#1` through `#15`. This means that the maximum number of data files that can be simultaneously used is 15.



File Names

With the BASIC Unit, the data file can be read only by the memory card. In this case, a file name must be given to the file. A file name must consist of eight characters or less and start with an alphabetic character. A device name `0:` is prefixed to the file name to access the memory card. In addition, an extension consisting of up to three characters can also be suffixed.

`″0:MFILE.DAT″` . ... `0` is the device (`0:` memory card), `MFILE` is the file name, and `DAT` (preceded by ".") is the extension

If the file name consist of 9 or more characters, or if the extension consists of 4 or more characters, the excess characters are ignored and thus not recognized. A period (`.`) must proceed the extension. A file name can also be specified in character string.

⚠ Caution Although file names in the BASIC Unit can consists of any characters except:, ., and blanks, lowercase letters and the ¥ symbol can cause problems on DOS machines and should be avoided.

Opening/Closing Files

Opening

A file is opened by the `OPEN` command. Once a file has been opened, the file number assigned to that file must not be used by any other files until closed by the `CLOSE` command. The `OPEN` command specifies a file name, mode, and file number. The mode does not need to be specified for a random access file.

Opening Sequential Access File

```
OPEN ″0:DATA2″ FOR OUTPUT AS #1
```

`0:DATA2` is the device and file name, `OUTPUT` is the mode setting, and `#1` is the file number.

Three modes can be specified: `INPUT` (to read data from a file), `OUTPUT` (to write data to the file), and `APPEND` (to add data to the file).

Opening Random Access File

`OPEN ″0:SAMPLE″ AS #1` . .... `0:SAMPLE` is the directory and file name, and `#1` is the file number.

If the mode is omitted, the random access file is assumed.

Closing

To end inputting/outputting of a file, the file number allocated by the OPEN command must be released by using the CLOSE command to close the file. When the CLOSE command is executed, the data remaining in the buffer is written to the file, so that the file number assigned to that file can be used by other files. Therefore, the CLOSE command must be used in conjunction with the OPEN command. When the END or STOP command is executed, the open files are automatically closed.

CLOSE #1,#2 . ... #1 and #2 are the file numbers (several files can be closed simultaneously) and if omitted, all files are closed

Operation of Sequential Access File

Data is sequentially written to a sequential access file starting from the beginning of the file. Any part of data cannot be rewritten, and only new data can be added to the end of the file.

Opening File With OPEN Command

```
OPEN ″0:DATA2″ FOR OUTPUT AS #1
```

0:DATA2 is the directory and file name, OUTPUT is the mode setting, and #1 is the file number

OUTPUT: write

INPUT: read

APPEND: additional write

In the above example, sequential file DATA2 is opened under file name of #1 to output data to the file.

In OUTPUT and APPEND Modes

```
WRITE #1,A$,B$
READ #1,A$,B$
```

#1 is the file no., and A$ and B$ are the variables

Character data given by character variables A$ and B$ are written file #1 in the order of A$ and B$.

In INPUT Mode

INPUT #1,A$,B$ . ... #1 is the file no., and A$ and B$ are the variables

Sequential data is read from file #1 and stored into A$ and B$.

Closing Using the CLOSE or END Command.

CLOSE #1 . ... #1 is the file no.

When a file opened for output is closed, all the data remaining in the buffer is written to the file and then the file is closed.

The following sample program illustrates the above process:

```
10 PARACT 0
20 OPEN ″0: DATA2″ FOR OUTPUT AS #1
30 A$ = ″BASIC″: B$ = ″UNIT″
40 WRITE #1, A$, B$
50 CLOSE #1
60 OPEN ″0: DATA2″ FOR INPUT AS #1
70 INPUT #1, A$, B$
80 PRINT A$, B$
90 CLOSE #1
100 END
110 END PARACT
```

⇩ Result of execution

```
BASIC                       UNIT
```

Program Example of Sequential Access File

Here is an example of operating a sequential access file.

```
'Sequential file
90 PARACT 0
100 DIM F$30
110 OPEN "0: DATA2" FOR OUTPUT AS #1
                           Opened to output new sequential file
120     A$="  OMRON  "
130     B$=" BASIC "
140     C$="UNIT"
150     D$="BASIC UNIT"
160     WRITE #1,A$,B$ . ... Data output to sequential file (data com-
                           pression)
170     PRINT #1, USING "  &   & &      &";C$,D$
                           Data output to sequential file with format
180     GOSUB *WRT
190 CLOSE . ................. Closes opened file
200 OPEN "0: DATA2" FOR INPUT AS #1
                           Opens sequential file for input
210     PRINT "Contents of data file are as follows"
220     LINE INPUT #1, F$  Reads one entire line to character vari-
                           able (F$)
230     PRINT F$
240     LINE INPUT #1, F$
250     PRINT F$
260     GOSUB *RD
270 CLOSE
280 END
290 '
300 *WRT . .................. Processing to output data to sequential
                           file
310 INPUT "Input data (999 to end writing)";E$
320 IF E$="999" THEN RETURN
330 PRINT #1, E$ . ......... Data output to sequential file
340 GOTO *WRT
350 '
360 *RD . .................... Processing to input data from sequential
                           file
370 IF EOF(1) THEN RETURN  Branches if data runs out
380 INPUT #1, G$ . ......... Reads data
390 PRINT G$
400 GOTO *RD
410 END PARACT
```

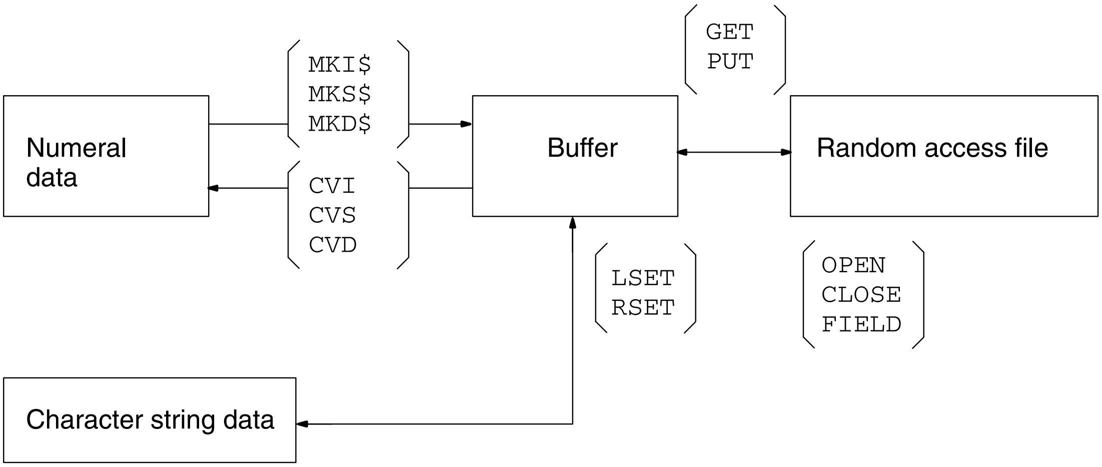
Result of execution

```
Input data (999 to end writing)?1
Input data (999 to end writing)?2
Input data (999 to end writing)?3
Input data (999 to end writing)?999
Contents of data file are as follows
UNIT      BASIC UNIT
"  OMRON  ", " BASIC "
 UNIT        BASIC UNIT
1
2
3
```

Operating Random Access File

The data length of a sequential access file can be set freely. Data of a random access file is read or written in record units, and the data length is fixed in record units. However, the random access file can be accessed more quickly than the sequential access file since data (record) can be read or written in any sequence. Only character data can be used with the random access file. To write numeric data, it must be converted into character data by the `MKI$`, `MKS$`, or `MKD$` functions when it is written. When reading data, it is converted back to numeric data by the `CVI`, `CVS`, or `CVD` functions.

Numeric data for random access files is converted into character data as seen in the following diagram.



Programming Sequence

1, 2, 3...

1. Open a file using the `OPEN` command.

   `OPEN ″0:DATA3″ AS #10 .` `DATA3` is the directory and file name, and `#1` is the file number

   To read/write data from/to random access file `DATA3`, the file is opened under file number `1`.

2. Assign variable areas to the buffer in record units by using the `FIELD` command.

   `FIELD #1, 5 AS A$, 18 AS B$`
   `#1` is the file number, `5` is the field width, and `A$` is the character variable

   A 5-byte variable area is assigned under variable name `A$` and an 18-byte area is assigned under variable name `B$` to the I/O buffer of the random access file opened under file name `#1`. To assign an area of more than 19 bytes to a character variable, allocate a variable area at the beginning of the program by using the `OPTION LENGTH` command. More than one character variable can be specified, but keep the total field width to within 256 bytes.

3. To write data to a file, set the data in the buffer by the `LSET` or `RSET` command, and write the data to a record of the file from the buffer by using the `PUT #` command.

   `LSET A$ = ″BASIC″ . .....` `A$` is the variable name, `BASIC` is the character sting.

   To write, left-justified, character string BASIC to variable area (buffer) of variable name A$.

   `RSET B$ = ″UNIT″ . ......` `B$` is the variable name, `UNIT` is the character string

To write, right-justified, character string `UNIT` to variable area (buffer) of variable name `B$`.

```
PUT #1,8 . ...............  #1 is the file number, 8  is the record
                            number (1 through 32767)
```

The data in the buffer is written to the eighth record of the random access file opened under file number `#1`.

4. Use the `GET #` command to read data from the file.

```
GET #1,8 . ...............  #1 is the file number, 8 is the record
                            number (1 through 32767).
```

The data is read to the buffer from the eighth record of the random access file opened under file number `#1`. This data is stored into a variable defined by the `FIELD` command, and therefore, can be displayed by the `PRINT` command.

5. Close the file by using the `CLOSE` command.

```
CLOSE #1 . ...............  #1 is the file number.
```

The file opened under file number `#1` is closed.

The following sample program illustrates the above procedure.

```
10 OPTION LENGTH 20
20 PARACT 0
30 OPEN ”0: DATA3” AS #1
40 FIELD #1, 15 AS A$, 20 AS B$
50 LSET A$ = ”BASIC”
60 RSET B$ = ”UNIT”
70 PUT #1, 8
80 GET #1,8
90 PRINT A$; B$
100 CLOSE #1
110 END
120 END PARACT
```

⇓ Result of execution

```
BASIC             UNIT
```

Program Example of Random Access File

```
10 ’Random file
      ⋮
90 PARACT 0
100 DIM A$50
110 ON ERROR GOTO *ERPRCS  Setting of error processing routine
120 OPEN ”0:DATA3” AS #1 . Opens random file
130     FIELD #1, 50 AS A$ Assigns variable area
140     PRINT ”Input [W] to write file.”
150     PRINT ”Input [R] to read file.”
160     PRINT ”Input [E] to end.”
170     B$=INPUT$(1) . .....  Conditional input from buffer to charac-
                              ter string
180     IF B$=”w” OR B$=”W” THEN GOSUB *WRT
190     IF B$=”r” OR B$=”R” THEN GOSUB *RD
200     IF B$=”e” OR B$=”E” THEN GOSTO *E
210     GOTO 140
220     *E
230     PRINT ”Data file size is” . .......... ;LOF(1);”
                              Size of file by record number
240 CLOSE #1 . ..............  Closes file
250 END
260 ’
```

```
270 *WRT . .................. Write subroutine
280 INPUT ”Record no. (1–999):”;REC%
290 IF REC%>999 THEN ERROR 1
                              Sets error generation number (ERR =
                              1)
300 IF REC%<1 THEN ERROR 2 Sets error generation number (ERR =
                              2)
310 LINE INPUT ”DATA:”;C$
320 PRINT ”Writes data (Y/[ELSE])”
330 D$=INKEY$ . ............ Inputs 1 character from keyboard
340 IF D$=”” THEN GOTO 330
350 IF D$=< >”Y” AND D$< >”y” THEN RETURN
360 LSET A$=C$ . ........... Sets data in buffer
370 PUT #1, REC% . ......... Writes buffer data
380 RETURN . ................ End of write subroutine
390 ’
400 *RD . .................... Read subroutine
410 INPUT ”Record no. (1–999):”;REC%
420 IF REC%>999 THEN ERROR 1
430 IF REC%<1 THEN ERROR 2
440 GET #1, REC% . ......... Reads data to buffer
450 PRINT A$
460 RETURN . ................ End of read subroutine
470 ’
480 *ERPTCS . ............... Error processing routine
490 IF ERR=1 THEN PRINT ”Record no. is too large.”
500 IF ERR=2 THEN PRINT ”Record no. is too small.”
510 IF ERL=440 THEN PRINT ”The record no. has NO data.”
                              When reading data fails
520 ’
530 RESUME 140
540 END PARACT
```

Result of execution

```
Input [W] to write file.
Input [R] to read file.
Input [E] to end.
Record No. (1–999):? 3
Data: 3
Writes data (Y/[ELSE])
Input [W] to write file.
Input [R] to read file.
Input [E] to end.
Record No. (1–999):? 4
Data: 4
Writes data (Y/[ELSE])
Input [W] to write file.
Input [R] to read file.
Input [E] to end.
Record No. (1–999):? 3
3
Input [W] to write file.
Input [R] to read file.
Input [E] to end.
Data file size is 4.
```

SECTION 6
Advanced Programming

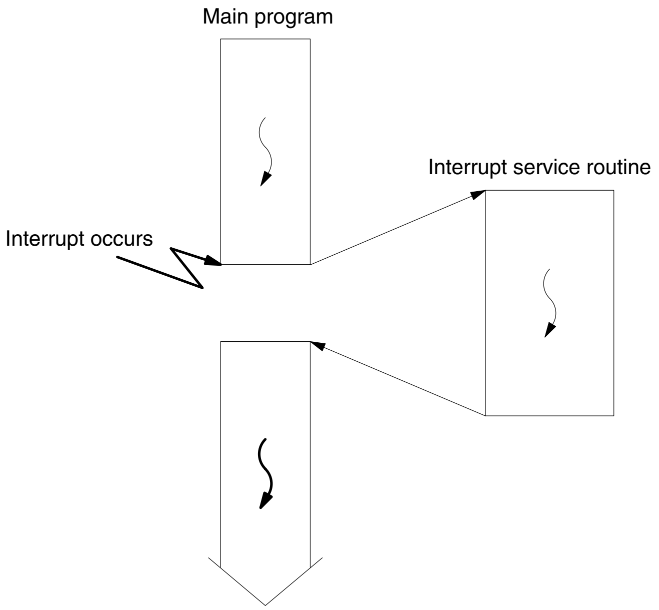
This section advances further into BASIC programming and provides information on interrupts, multitasking, and machine language for the purposes of advanced programming.

6-1 Interrupts

An interrupt is one means by which a device connected to the BASIC Unit can inform the program that some event has occurred and that some action on the part of the program is required immediately. For example, when a character is received by a communications port, the program must stop whatever it is doing and read the character as soon as possible so that the input buffer does not overflow. When an interrupt occurs, the BASIC Unit may stop executing the current task and run an *interrupt service routine* instead. When the service routine is finished, control is returned to the task that was executing before the interrupt.

Interrupts can also be used to restart a task which has been stopped by the `PAUSE` statement.

Interrupt Processing



The BASIC Unit supports several different interrupts which indicate various conditions. The table below lists the interrupt types and related BASIC instructions.

Interrupt Type	Meaning	BASIC Instructions	
`TIME$`	Time	`ON TIME$ GOSUB`	`TIME$ ON/OFF/STOP`
`ALARM`	Elapsed time	`ON ALARM GOSUB`	`ALARM ON/OFF/STOP`
`TIMER`	Time interval	`ON TIMER GOSUB`	`TIMER ON/OFF/STOP`
`KEY(`*key-number*`)`	Numeric key pressed	`ON KEY GOSUB`	`KEY ON/OFF/STOP`
`COM`	Input from communication port	`ON COM GOSUB`	`COM ON/OFF/STOP`
`PC`	Interrupt from CPU Unit	`ON PC GOSUB`	`PC ON/OFF/STOP`
`FINS`	Interrupt from network	`ON FINS GOSUB`	`FINS ON/OFF/STOP`
`SIGNAL` *signal-number*	Signal received from another task	`ON SIGNAL GOSUB`	`SIGNAL ON/OFF/STOP`
`ERROR`	Error occurred	`ON ERROR GOSUB`	`ERROR ON/OFF/STOP`

6-1-1 Defining an Interrupt Service Routine

Before interrupts of a certain type can be processed, the program must define an interrupt service routine to be called when that type of interrupt occurs. The `ON` *interrupt-type* `GOSUB` { *line-number* | *label* } instruction is used for this purpose. The *line-number* or *label* indicates the start of the service routine. Interrupt service routines must end with a `RETURN` statement.

6-1-2 Interrupt-related Instructions

Interrupts usually occur asynchronously; that is, the program cannot know when an interrupt will occur. However, there may be sections of the program which should not be interrupted. For example, if an interrupt occurs while the program is performing a time-critical calculation, the result of the calculation will be delayed and the program may miss its deadline.

Therefore, the BASIC Unit provides the *interrupt-type* `ON`, `OFF`, and `STOP` instructions, which may be used to enable, disable, or temporarily delay interrupts of the specified type.

The *interrupt-type* `ON` instruction enables interrupts of the specified type; after this instruction is executed, the interrupt service routine will be called each time an interrupt is received.

The *interrupt-type* `OFF` instruction disables interrupts of the specified type; after this instruction is executed, the BASIC Unit will ignore those interrupts. Interrupts during `OFF` execution for `COM`, `PC`, and `FINS`, however, are handled the same as those during `STOP` execution, as described next.

The *interrupt-type* `STOP` instruction disables interrupts of the specified type, but any interrupts received while the interrupt is `STOP`ped will be recorded, and the interrupt service routine will be called if the interrupt is later enabled.

Note Interrupts from a source are disabled (turned `OFF`) immediately after an interrupt service routine for that type of interrupt is defined (or re-defined) with `ON` *interrupt-type* `GOSUB`. Furthermore, interrupts are `STOP`ped while the interrupt service routine is being executed.

An interrupt can be accepted while an input instruction is being executed. When an interrupt-type ON instruction occurs while an input instruction is being executed, the input instruction will be interrupted and the interrupt service routine as defined by the interrupt will be executed. If all I/O data has not been processed when the interrupt occurs, the data will be discarded and the instruction ended.

An interrupt will be `STOP`ped if the another interrupt from the same source occurs before interrupt processing is completed. To produce effective interrupts, write multitasking programs so that each interrupt is executed independently (for example: ON COM2 combined with INPUT or ON PC combined with PC READ). If interrupts are combined during single task execution, PC STOP must be executed during `INPUT`.

There is no priority ranking for the interrupts listed above. If an interrupt is received during the execution of any interrupt subroutine, the later one interrupts the earlier one and is executed.

6-1-3 Interrupt Programming

To write a program that makes use of interrupts:

1, 2, 3...

1. Select the type of interrupt to be used and develop an interrupt routine. Be sure to use a `RETURN` statement at the end of the routine.
2. Define the interrupt routine in the main routine using the `ON` *interrupt-type* `GOSUB` statement.
3. Use the *interrupt-type* `ON` instruction to enable interrupts.
4. If an interrupt occurs, the interrupt routine will be executed. Execution continues at the point where the interrupt occurred when the interrupt routine's `RETURN` statement is executed.
5. Use *interrupt-type* `STOP` if necessary to protect sections of the program from interruption. Use *interrupt-type* `OFF` when you are no longer interested in interrupts.

Interrupt Programming Example

```
100 PARACT 0
110 ON KEY(1) GOSUB 700  . . Define interrupt service routine

            Interrupts from numeric
            key 3 are ignored.

200 KEY(1) ON  . ............ Enable interrupts

            If numeric key 3 is pressed
            here, the interrupt service
            routine will be called.

300 KEY(1) OFF  . ........... Disable interrupts

            Interrupts from numeric
            key 3 are ignored.

600 END . .................... End of main routine
700 'Start of KEY 1 interrupt routine

800 RETURN  . ................ End of interrupt service routine
```

6-1-4 Interrupt Types

Timer Interrupts

The BASIC Unit supports three types of timer interrupts. These interrupts occur at a specified time (`TIME$`), at specified time intervals (`TIMER`), or when a specified time has elapsed (`ALARM`).

Interrupt at Specified Time

The `ON TIME$ GOSUB` statement defines an interrupt routine to be executed at a specified time. For example:

```
100 ON TIME$ = ″02:30:10″ GOSUB 1000
110 TIME$ ON
```

The interrupt service routine starting at line 1000 will be called at 2:30:10. The time at which the interrupt occurs is specified as a character string containing hours, minutes, and seconds.

Interval Interrupt

The `ON TIMER GOSUB` statement defines an interrupt service routine to be executed repeatedly at a certain interval. For example:

```
100 ON TIMER 3600 GOSUB 1000
110 TIMER ON
```

The interrupt service routine starting at line 1000 will be executed once every six minutes until `TIMER STOP` or `TIMER OFF` is executed. The time interval is specified in units of 0.1 second, in the range 1 to 864000 (0.1 second to 24 hours).

Elapsed Time Interrupt

The `ON ALARM GOSUB` statement defines an interrupt service routine to be executed once after the specified time has elapsed. For example:

```
100 ON ALARM 10 GOSUB 1000
110 ALARM ON
```

The interrupt service routine starting at line 1000 will be called 1 second later. The time is specified in units of 0.1 second.

Numeric Key Interrupts

The `ON KEY GOSUB` statement defines an interrupt routine to be executed when a certain numeric keypad key is pressed. For example:

```
100 ON KEY(1) GOSUB 1000
110 KEY(1) ON
```

When numeric key 1 is pressed, the interrupt service routine starting at line 1000 will be executed.

The key pressed is read during the interrupt processing and does not remain in the input buffer.

Communications Port Interrupts

The `ON COM GOSUB` statement defines an interrupt routine to be executed when a character is received by a communications port. For example:

```
100 ON COM(2) GOSUB 1000
110 COM(2) ON
```

When a character is received by communications port 2, the interrupt service routine starting at line 1000 will be executed. If the port number is omitted, port 1 is assumed.

Interrupts for communications ports are enabled and disabled using `COM ON` and `COM OFF`. `COM STOP` will operate the same as `COM OFF`.

This table shows the correspondence between port numbers and port types.

Port no.	Port type
1	RS-232C
2	RS-232C
3	RS-422

Network Interrupts

The `ON FINS GOSUB` statement defines an interrupt routine to be executed when data is received from another BASIC Unit on the PC or connected with a network, or an FA computer. For example:

```
100 ON FINS GOSUB 1000
110 FINS ON
```

The interrupt service routine starting at line 1000 will be called when network data is received. (For information about establishing communication between BASIC Units, see *7-1 Peripheral Device Operation*.)

Interrupts from networks are enabled and disabled using `FINS ON` and `FINS OFF`. `FINS STOP` will operate the same as `FINS OFF`.

Signal Interrupts

The `ON SIGNAL GOSUB` statement defines an interrupt routine to be executed when a specified signal is received from another task. For example:

```
100 ON SIGNAL 5 GOSUB 1000
110 SIGNAL 5 ON
```

When signal 5 is received from another task, the interrupt routine starting at line 1000 will be executed. (For more information about signals, see *6-2-5 Inter-task Communication*.)

PC Interrupts

The `ON PC GOSUB` statement defines an interrupt routine to be executed when an interrupt from a PC is received. For example:

```
100 ON PC(2) GOSUB 1000
110 PC(2) ON
```

When interrupt 2 is received from a PC (the user program in the CPU Unit executes a SEND(192) or RECV(193) instruction), the interrupt routine starting at line 1000 will be executed. (For more information about PC communications, see 6-4 *PC Communications*.)

Interrupts from the CPU Unit are enabled and disabled using `PC ON` and `PC OFF`. `PC STOP` will operate the same as `PC OFF`.

Error Processing

Error processing is slightly different than other interrupt processing. If the BASIC Unit encounters an error (for example, if the program attempts to divide by zero), execution is normally terminated and an error message is printed. If an error-handling “interrupt” routine is defined with the `ON ERROR GOTO` statement, the BASIC Unit will instead execute that routine. The routine can take whatever action is necessary to correct the error and continue.

The `ON ERROR GOTO` statement defines an interrupt routine to be executed if the BASIC Unit encounters an error. For example:

```
100 ON ERROR GOTO 1000
```

If the BASIC Unit encounters an error, the interrupt routine starting at line 1000 will be executed. To restore the default error action, specify line 0.

The line number on which the error occurred and a number indicating the error type can be obtained with the `ERL` and `ERR` functions. (For a list of error codes and corresponding error conditions, see *Section 8-1-1 Error Messages*.)

To exit from the error processing routine, use the `RESUME` statement instead of the `RETURN` statement. `RESUME` can take one argument, which can be a line number where execution should continue, 0 to indicate that the error line should be re-executed, or `NEXT`, to resume execution at the line after the error. If no argument is supplied to `RESUME`, the BASIC Unit attempts to execute the error line again.

Note There is no `ERROR ON`, `ERROR OFF`, or `ERROR STOP` statement. Error processing is always enabled.

6-1-5 Interrupt Processing Details

The BASIC Unit maintains three system variables, `INTRB`, `INTRL`, and `INTRR`, which can be examined in an interrupt service routine to find information about the current interrupt. `INTRR` contains a number indicating the interrupt source:

Interrupt source	**INTRR**
User-defined signal (1 to 5)	1 to 5
Communication port COM (1 to 3)	6 to 8
Signal (STOP)	10
Signal (PC watchdog timer error)	11
Signal (cyclic error)	12
Signal (battery error)	13
Alarm	14
Timer	15
Time	16
SRQ (service request from GP-IB)	17
FINS (network)	18
Numeric key input (0 to 9)	20 to 29
PC (1 to 15)	31 to 45

`INTRB` contains the number of the line to be executed next when the interrupt occurred; when the service routine returns, execution will continue at that line.

INTRL contains the line number of the statement that was aborted by the interrupt, or 0 if no statement was aborted. Some BASIC Unit instructions take an indefinite amount of time to complete. For example, the INPUT statement causes the Unit to wait until the user has entered a value at the terminal. If an interrupt occurs while the Unit is waiting for such a statement to complete, the statement will be aborted and INTRL will contain the statement's line number. The instructions below may be aborted by an interrupt, and will cause a line number to be stored in INTRL if they are:

```
GET #
INPUT
INPUT WAIT
INPUT$
INPUT #
INPUT@
LINE INPUT
LINE INPUT #
LINE INPUT WAIT
LPRINT
LPRINT USING
PRINT
PRINT@
PRINT #
PRINT # USING
PUT #
RECEIVE
SEND
TWAIT
WRITE
WRITE #
PC READ
PC WRITE
```

Note INTRR and INTRB are saved before an interrupt routine is called and restored after the routine returns, so they always contain the correct values for the current interrupt, even if execution is not completed or a second interrupt occurs while the Unit is executing a different interrupt service routine.

This example shows one way to re-start a statement if it is aborted by an interrupt. If this type of programming is not implemented, the program line 100 may be aborted before completion.

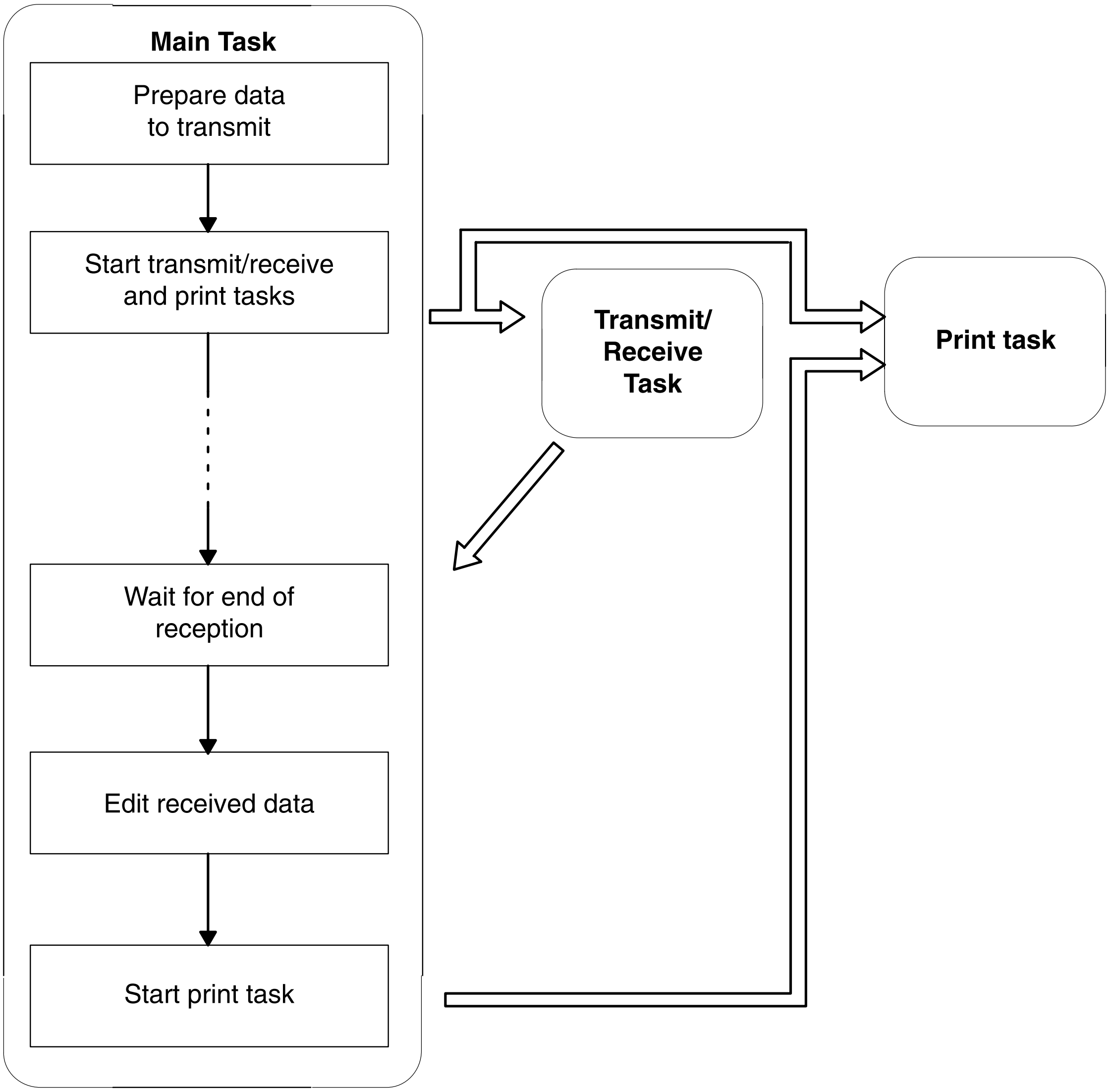
```
 10 ON TIMER 100 GOSUB *SUB
 20 TIMER ON
    ⋮
100 INPUT A$.
110 IF WAS_ABORTED = 100 THEN WAS_ABORTED = 0 : GOTO 100
    ⋮
480 *SUB
500 WAS_ABORTED = INTRL                         Subroutine
    ⋮
600 RETURN
```

6-2 Multitasking

6-2-1 Tasks

- A task is a series of instructions necessary for a computer to complete one process and is one unit of a program.
- Tasks are classified by the function they perform; for example, a print task prints data with a printer, a text display task displays characters on a CRT screen, and a CPU Unit communications task communicates with the CPU Unit.
- Multitasking is the ability to execute two or more tasks simultaneously on one computer. The BASIC Unit can execute up to 16 tasks simultaneously.

- The following example shows tasks that transmit and print data, and print received data.



Note Execution of tasks switches after each instruction, even for compound lines. Task execution begins with the task with the smallest task number and moves in order to all tasks in the READY status. If execution for a task is not possible when it is switched to (e.g., the task is waiting for input), the next task will be switched to immediately.

6-2-2 Declaration of Start & End of Task Program

The `PARACT` statement must be used to declare the beginning of each task program. The task program must end with the `END PARACT` command.

Declaring the Start of a Task Program

`PARACT` *task-no.* [`WORK` *no.-of-bytes*]

Here, *task-no.* is an integer from 0 to 15, and *no.-of-bytes* is the size of the task work area (default: 1024 bytes).

Statements between the `PARACT` and `END PARACT` statements constitute a task.

Task number 0 is the main task and will be executed first. If a program contains no task 0, an error will occur and the program will not be executed.

The number of `WORK` bytes is the number of bytes of work area used by the task. The default value is 1024 bytes.

The `PARACT` statement must appear on alone on a line; it cannot be used in a multi-statement line.

Declaring the End of Task Program

```
END PARACT
```

The `END PARACT` statement is used on the last line of the task program to declare end the task program.

The `END PARACT` statement must appear alone on a line; it cannot be used in a multi-statement line.

Examples of Programming Tasks

Single Task

If there is only one task, it must be task number 0.

```
10 RDIM . .................... Declaration of non-volatile variables
20 DIM . ..................... Declaration of global volatile variables
30 PARACT 0 . ............... Beginning of task
      ⋮
1000 END PARACT . .......... End of task
```

Multiple Tasks

Task 0 is will be executed first when the program is started. Other tasks may be started by the first task.

```
10 RDIM . .................... Declaration of non-volatile variables
20 DIM . ..................... Declaration of global volatile variables
30 PARACT 8 . ............... Beginning of task 8
                                                  Note that tasks
      ⋮                                           can be declared
                                                  in any order.
100 END PARACT . ........... End of task 8
110 PARACT 0 . .............. Beginning of task 0
      ⋮
300 END PARACT . ........... End of task 0
310 PARACT 1 . .............. Beginning of task 1
      ⋮
660 END PARACT . ........... End of task 1
```

6-2-3 Starting, Aborting, and Waiting for a Task

A task can be started with the `TASK` statement and aborted by the `EXIT` statement. In addition, one task can wait for the end of another task that has been started by using the `TWAIT` statement.

If an attempt is made to start, stop, or wait for a task number that has not been declared by a `PARACT` statement, an error occurs.

Starting a Task

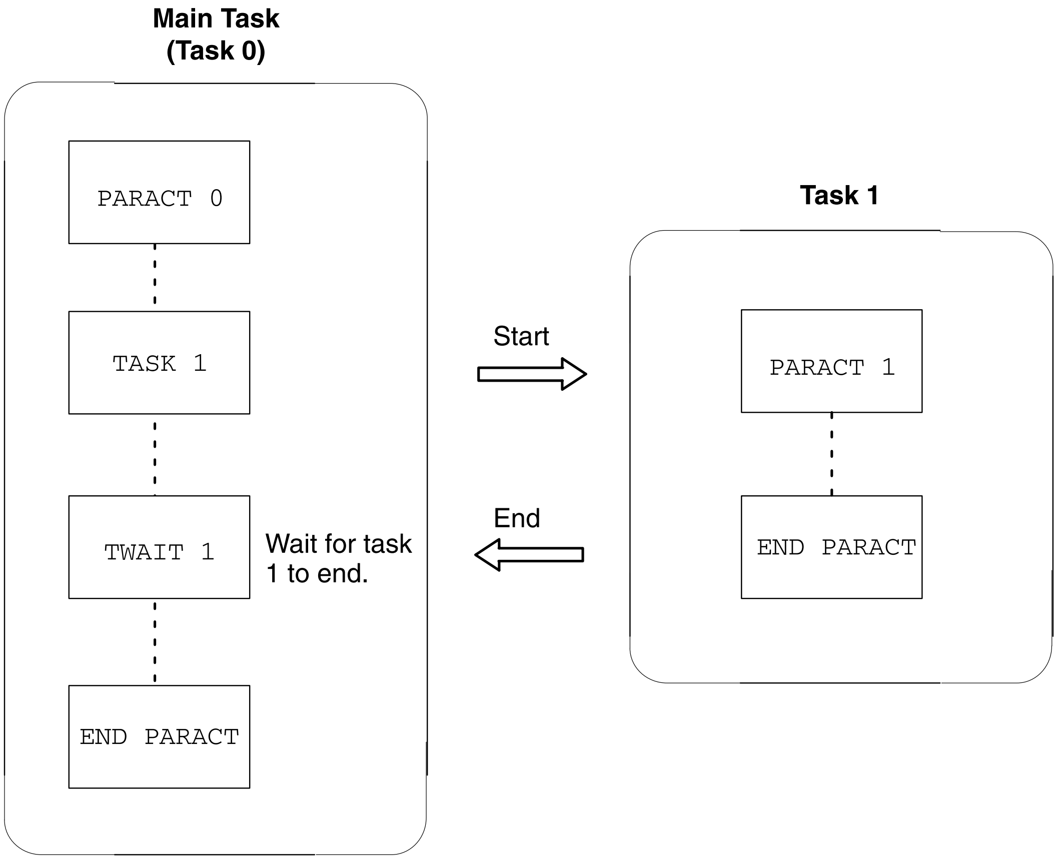
```
100 TASK 1 .................. Execution of task 1 is started from the
                              task's PARACT statement. If task 1 has
                              already been started, an error message
                              is displayed.
```

Aborting a Task

```
200 EXIT 1 .................. The EXIT command aborts a specified
                              task. If the is not running, an error mes-
                              sage is displayed.
```

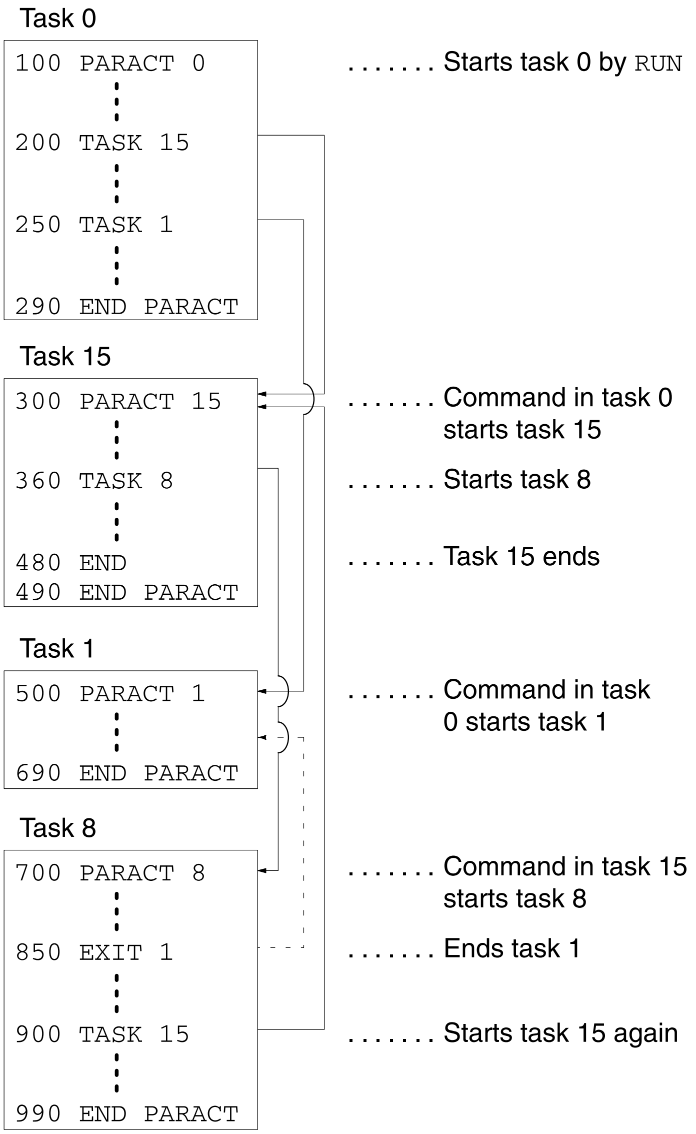
Waiting for End of Task

`300 TWAIT 1 .` .... The task that executed this `TWAIT` statement will wait for task 1 to exit before continuing. If an interrupt occurs while the task is waiting, the Unit will execute the task's interrupt routine and then resume waiting. If the specified task has already ended, an error message is displayed.



Example of Program Starting/Ending Task

When the RUN command is entered from the terminal or when the BASIC Unit is started by the RUN/STOP switch or by the setting of the automatic start setting area of the memory switch, task 0 is started. Task 0 can then start other tasks with the TASK command.



Task 0

```
100 PARACT 0
      :
200 TASK 15
      :
250 TASK 1
      :
290 END PARACT
```

Task 15

```
300 PARACT 15
      :
360 TASK 8
      :
480 END
490 END PARACT
```

Task 1

```
500 PARACT 1
      :
690 END PARACT
```

Task 8

```
700 PARACT 8
      :
850 EXIT 1
      :
900 TASK 15
      :
990 END PARACT
```

.... Starts task 0 by RUN

.... Command in task 0 starts task 15

.... Starts task 8

.... Task 15 ends

.... Command in task 0 starts task 1

.... Command in task 15 starts task 8

.... Ends task 1

.... Starts task 15 again

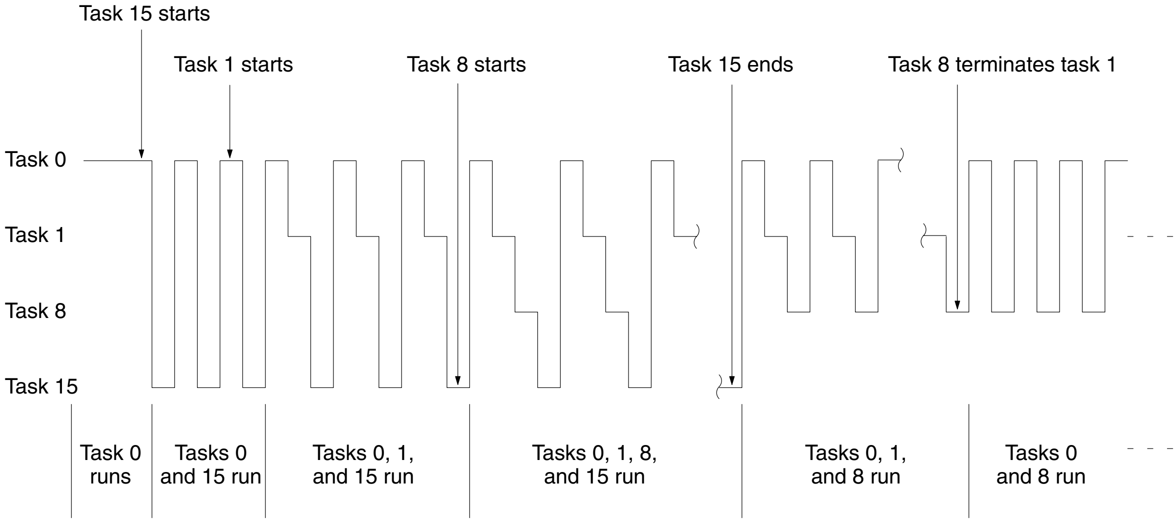
Note

1. Task 0 is started when the program is started.
2. Tasks 15 and 1 are started by the TASK commands in task 0.
3. Task 8 is started by the TASK command in task 15.
4. Task 15 ends when it executes line 480.
5. Task 1 is terminated by the EXIT command in task 8.
6. Task 15 can be started again by the TASK command in task 8 even after it has exited once.

Switching Tasks

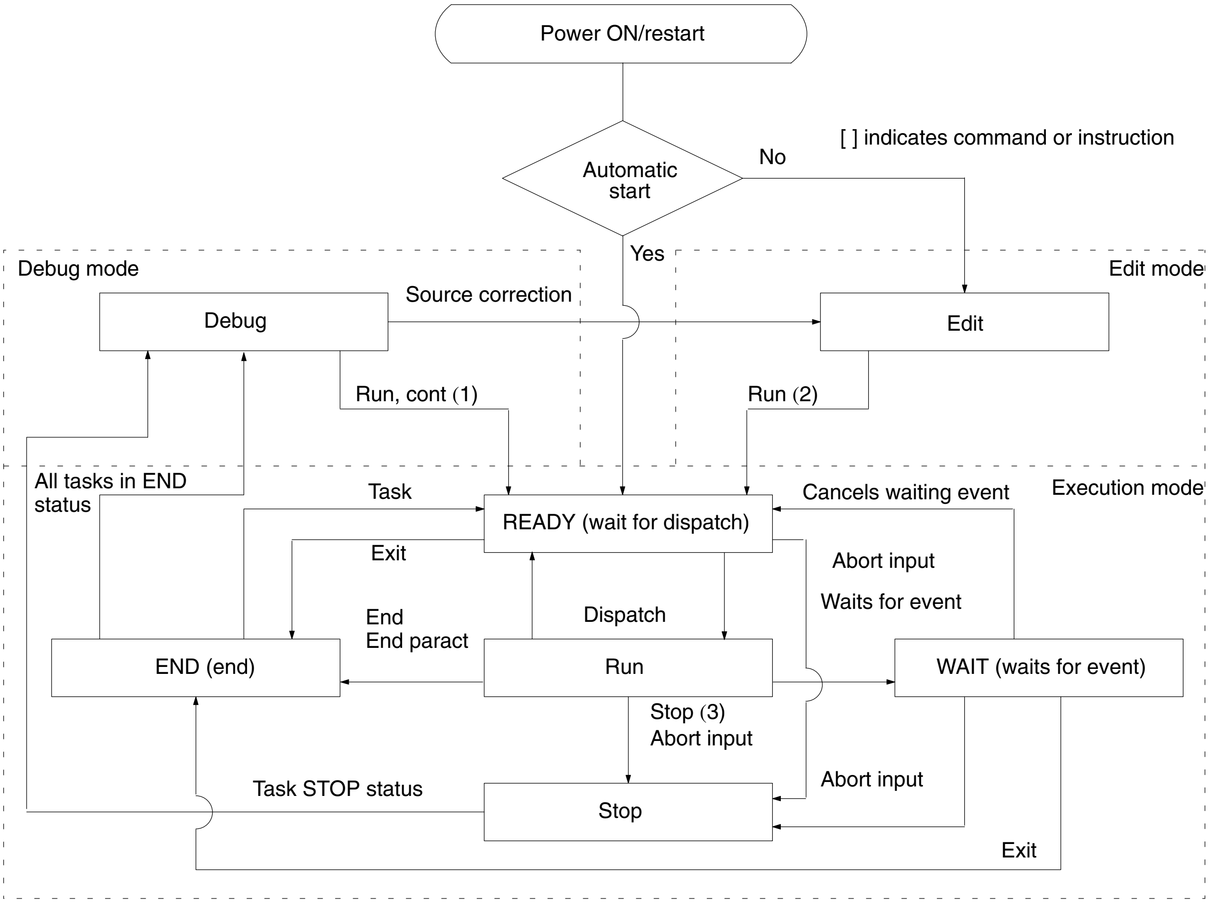
When two or more tasks have been started, the BASIC Unit switches between active tasks in round-robin fashion, executing a single statement from each task in turn. In each execution cycle, the next statement from each active task is executed in order of its task number. If a task uses an input or output statement such as `PRINT` or `INPUT`, or some other statement which involves waiting time, that task is excluded from the round-robin until the input/output processing or waiting is completed.

In the previous example, statements are executed from the active tasks in the following order:



6-2-4 BASIC Unit Status and Transitions

After the BASIC Unit has been started, the internal status of program execution and termination changes as illustrated below.



Note
1. `CONT` is valid only after `STOP` is executed.
2. Can also be started by the RUN/STOP switch.
3. Can also be stopped by the RUN/STOP switch or a `BREAK` point setting.

BASIC Unit Modes

Edit All the tasks are in the END state and the source program is being created or edited. The program can be edited on the terminal.

Debug All the tasks are stopped and the source program is not being edited. The program can be debugged through operations on the terminal.

Execute One or more tasks are running. The debug mode can be set when an abort operation is performed on the terminal, when the RUN/STOP switch is operated, or when a `STOP` statement is executed.

Task Status

RUN A statement from the task is being executed. Only one task can have this status at a time.

READY The task is waiting for its turn in the round-robin.

WAIT The task is waiting for the end of an input/output operation or for an interrupt.

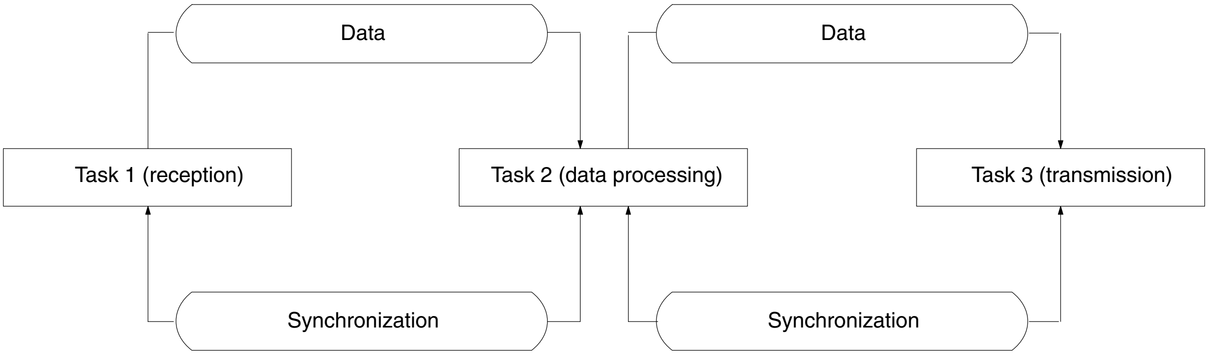
END The task is not running.

STOP The task is temporarily stopped, but can be resumed by a `CONT` or `STEP` command.

6-2-5 Inter-task Communication

When a multitasking program is executed, it may be necessary to transfer data between tasks or to synchronize execution of tasks. Transfer of information among tasks is generically called inter-task communication.

For example, consider an application which requires the BASIC Unit to receive some information, perform a calculation on the data, and send the result back. A multitasking version of such a program could consist of three tasks: task 1 performing data reception, task 2 performing data processing, and task 3 performing data transmission. Each task in this program must be synchronized with the others to exchange data properly. For example, task 2 must wait for task 1 to receive some data before it can begin calculations, and task 3 must wait for task 2 to finish its processing before the results can be sent.



In the multitasked approach, the variables used by each task are *local* to the task; that is, the variables of one task cannot be directly referenced by the other tasks. To perform inter-task communication, messages to transfer data between tasks and global variables that can be accessed by each task are used to transfer data between tasks. In addition, a signal may be used to inform a task of the occurrence of an event in another task.

Types of Inter-task Communication

The BASIC Unit supports three different methods of inter-task communications. The simplest method is the signal, which one task can use to inform another task that some event has taken place. The second method is the message, which a task can use to send information to another task. The third method is the use of global variables, which can be accessed by any task.

Signals

Signals can be used to inform a task of the occurrence of an event in another task, and are useful when it is necessary to establish synchronization between tasks. A task in which an event has occurred sends a signal to another task with the `SENDSIG` statement. The other task must define a processing routine with the `ON SIGNAL GOSUB` statement. Then, when the second task wishes to receive signals from the first task, it executes the `SIGNAL ON` statement. Signal processing works the same as interrupt processing; see *Section 6-1 Interrupt Operation* for details.

Sending a Signal

A task sends a signal by executing the `SENDSIG` statement:

`SENDSIG` *signal-no., task-no.*

Signal-no. must be an integer from 1 to 5 or 10 to 13. Signals 10 through 13 have pre-defined meanings; signals 1 through 5 are available for user definition. The meanings of the pre-defined signals are:

Signal	Meaning
10	STOP
11	PC watchdog timer error
12	Cyclic error
13	Battery error

Defining a Signal Processing Routine

A task that wishes to receive a signal must first define an interrupt processing routine to be executed when the signal is received. The routine is defined with the `ON SIGNAL GOSUB` statement:

`ON SIGNAL (`*signal-no.*`) GOSUB` {*line-no.* | *label*}

Enabling / Disabling / Stopping Signal Interrupts

After the `ON SIGNAL GOSUB` statement has been executed, the task must execute `SIGNAL ON` when it is ready to receive signals. When the task is no longer interested in the signal, it should execute `SIGNAL OFF`. To temporarily disable processing of a signal, execute `SIGNAL STOP`. The difference between `SIGNAL OFF` and `STOP` is that `STOP` records any signals received while the signal is `STOP`ped, and interrupt processing is executed if the interrupt is later enabled by `SIGNAL ON`. Signals received while `SIGNAL OFF` is in effect are ignored.

Signal Program Example

```
10 PARACT 0 . ............... Beginning of task 0
20 TASK 1 . ................. Start execution of task 1
      ⋮
80 PRINT "Task 0 -> Task 1"
90 PRINT "Send signal 3"
100 SENDSIG 3, 1 . .......... Send signal 3 to task 1
      ⋮
190 END PARACT . ........... End of task 1
200 PARACT 1 . .............. Beginning of task 1
210 ON SIGNAL 3 GOSUB 300    Define signal processing routine
220 SIGNAL 3 ON . .......... Enable interrupts for signal 3
230 PAUSE . ................. Wait for a signal
      ⋮
290 END
300 REM Signal 3 processing routine
310 PRINT "Received signal 3"
      ⋮
390 RETURN
400 END PARACT . ........... End of task 1
```

Result of execution:

```
Task 0 -> Task 1
Send signal 3
Received signal 3
```

Note

1. If the signal receiving task has no processing to do until the interrupt occurs, it can execute the `PAUSE` statement to wait for an interrupt to occur.
2. In the example, if task 0 sends the signal to task 1 immediately after starting task 1, the signal may not be received because task 1 may not have finished defining the signal processing routine and enabling interrupts.

   If it is important that task 1 receive *every* signal, the program could be re-written so that task 1 signalled task 0 when it was ready to receive signals.

Messages

Tasks can use messages to communicate when the information to be sent is more complicated than the simple on/off that a signal can indicate.

To communicate with messages, the two tasks must first acquire a message number. Then, the transmitting task sends the message with the `SEND` statement, and the receiving task gets the message with the `RECEIVE` statement. When the tasks are done communicating, they should release the message number.

Each instruction is explained in more detail below.

Allocating Message Number

To use a message, both tasks must allocate the message number with the `MESSAGE` statement:

`MESSAGE` *function*, *message-no.*

Function is 0 (allocate message number), and *message-no.* is an integer from 1 to 32767. Each task can acquire up to four message numbers, and a total of eight message numbers can be acquired for the entire program.

Transmitting Message

Next, the transmitting task prepares the message and sends it with the `SEND` statement:

`SEND` *message-no.*, *character-expression*

Message-no. is the message number acquired in the first step, and *character-expression* contains the information the task wishes to send. *Character-expression* can be up to 538 characters long.

Receiving Message

The receiving task gets the message with the `RECEIVE` statement:

`RECEIVE` *message-no.*, *character-variable*

Message-no is the message number acquired in the first step, and *character-variable* is the name of a variable into which the message will be stored. If the receiver executes `RECEIVE` before the transmitter executes `SEND`, the receiving task will wait until a message is transmitted.

Releasing Message Number

When the tasks are done communicating, they should release the message number with the `MESSAGE` instruction:

`MESSAGE` *function*, *message-no.*

Function is 1 (release message number), and *message-no.* is the number acquired in the first step.

Message Program Example

```
10 PARACT 0 . ............... Beginning of task 0
20 TASK 1 . ................. Start task 1
30 MESSAGE 0, 1 . .......... Acquire message number 1
40 A$ = ″START!″ . ......... Prepare data to send to task 1
50 SEND 1, A$ . ............ Send the message
      :
80 MESSAGE 1, 1 . .......... Release message number 1
90 END PARACT . ............ End of task 0
100 PARACT 1 . ............. Beginning of task 1
110 MESSAGE 0, 1 . ......... Acquire message number 1
120 RECEIVE 1, B$ . ........ Receive a message
130 PRINT ″Message from task 0...″; B$
      :
180 MESSAGE 1, 1 . ......... Release message number 1
190 END PARACT . ........... End of task 1
```

Global Variables

All the variables declared between the beginning of the program and the first `PARACT` statement can be accessed by every task.

These variables are called global variables. Global variables can be used to transfer data between tasks and to hold data common to two or more tasks.

An example in which task 0 stores data in global variables A and B and task 1 performs a calculation using the data is shown below.

Global Variable Program Example

```
10 RDIM A . ................. Declare non-volatile global variable A
20 DIM B  . ................. Declare (volatile) global variable B
30 PARACT 0 . ............... Beginning of task 0
40 A = 15  . ................ Store 15 in global variable A
50 B = 3  . ................. Store 3 in global variable B
60 TASK 1 . ................. Start task 1
      ⋮
90 END PARACT  . ............ End of task 0
100 PARACT 1 . .............. Beginning of task 1
110 C = A  . ................ Copy global variable A to local variable
                              C
120 D = B  . ................ Copy global variable B to local variable
                              D
130 E = C + D  . ............ Add the local copies of A and B and
                              store the result in local variable E
140 PRINT E
      ⋮
190 END PARACT  . ........... End of task 1
```

Inter-task Communication with Non-volatile Variables

Variables declared with the RDIM statement retain data even after the power has been turned off. These variables are called non-volatile variables, and are stored in battery-backed memory. Non-volatile variables can be declared only in the global definition block, i.e., from the beginning of the program to the first PARACT statement.

Variables declared with RDIM must appear before those declared with DIM.

Non-volatile variables are not cleared even when the power has been turned off. To clear these variables, execute the OPTION ERASE or RUN ERASE command. Non-volatile variables can be saved to or loaded from a file with the VSAVE or VLOAD commands.

6-3 Machine Language

The BASIC Unit provides support for machine language programming. Machine language subroutines can be called from BASIC programs, access BASIC variables, and return results to the program.

Machine language programs can be entered, modified, and debugged when the Unit is in the machine language monitor mode. Use the MON command to enter this mode.

The BASIC Unit's CPU is a V25 (NEC µPD70322), and the monitor assembler accepts most (but not all) V25 mnemonics and notations. See *Appendix E* for more information.

The machine language program can be entered in these ways:

1, 2, 3...

1. Enter one instruction at a time with the machine language monitor's mnemonic assembler.
2. Store the subroutine object code as data in the BASIC program and use POKE to place the code in memory.
3. Load the subroutine object code from a file with the LOAD instruction.

6-3-1 Segments and Offsets

Memory addresses used by the BASIC Unit consist of two parts: the *segment* and the *offset*. Both are 16-bit integers. The actual memory address used by an instruction is calculated by multiplying the segment number by 16 and adding the offset. For example, segment &H0050 and offset &H1234 give the actual memory address &H01784:

Segment address			0	0	5	0
Multiply by 16		0	0	5	0	0
Offset address	+		1	2	3	4
Actual address		0	1	7	8	4

The segment address is specified by the `DEF SEG` statement in the BASIC program, and is contained in DS0 in the machine language monitor mode. The `G`, `T`, and `B` commands, however, use PS (program segment).

6-3-2 Developing a Machine Language Program

This section describes how to develop a machine language program. Only the major commands are described. For details, refer to *Appendix E Machine Language Monitor Reference*.

Allocate Memory

First, allocate an area in memory to hold the machine language subroutine.

The machine language program area is located before (at lower addresses than) the user program area. The BASIC program area capacity is reduced by the amount allocated for the machine language program.

To allocate the area, use the `MSET` command:

`MSET &H4000` . ... The machine language program area is from address `&H500` to `&H3FFF`. Addresses `&H4000` and those that follow contain the BASIC program and variables.

If `MSET` is entered without an argument, the current set value is displayed. The value set with `MSET` is stored in battery-backed memory, so it is not necessary to execute `MSET` each time power is turned on.

Note When the BASIC Unit is started for the first time, the beginning of the BASIC program area is set to &H500, and no machine language program area is allocated. Be sure to allocate the machine language program area with `MSET` before developing a machine language program.

Enter the Machine Language Program

To enter a machine language program from the terminal, first set the BASIC Unit's machine language monitor mode with the `MON` command. The Unit's RUN indicator will light, and the `*` prompt will be displayed. All subsequent input must use upper case letters only.

Use the `A` (Assemble) command to start assembling the program. When this command has been entered, the prompt will change to an exclamation point (`!`). Next, enter the program start address (in hexadecimal, followed by a colon) and the first machine language instruction. When you type return (↵), the BASIC Unit will reply with the address, object code, and corresponding mnemonic.

```
MON↵
*A↵
!3000:MOV␣AW,PS↵
3000 8CC8      MOV   AW,PS
```

No address is necessary if you wish to continue entering the program; the BASIC Unit automatically increments the location counter appropriately. When you have finished entering the program, type X↵ to return to the * prompt.

Corrections can be made by deleting with the Backspace Key until the carriage return key is input.

As an example, here is a simple program that adds 7 to the contents of location &H1000 and stores the result at &H1000.

```
*A ......................... Begin assembling
!3000:MOV AW,PS ........... See note
3000 8CC8      MOV    AW,PS
!MOV DSO,AW
3002 8ED8       MOV    DS0,AW
!MOV AL,7 ................. Load 7 into AL
3004 B007       MOV    AL,07
!MOV BL,[1000] ............ Load the contents of &H1000 into BL
3006 8AIE0010   MOV    BL,[1000]
!ADD AL,BL ................ Add BL to AL (result stored in AL)
300A 00D8       ADD    AL,BL
!MOV [1000],AL ............ Store the result in &H1000
300C A20010     MOV    [1000],AL
!BR 300F
300F E9FDFF     BR 300F
!X
*
```

Note The first two instructions, MOV AW,PS and MOV DS0,AW are used to make the data segment equal to the program segment.

Check the Program

To verify the program just entered, display it with the I (Inverse Assemble) command. This command displays the object code and mnemonics of the program.

```
*I3000.300F ............... Disassemble from &H3000 to &H300F
3000 8CC8       MOV    AW,PS
3002 8ED8       MOV    DS0,AW
3004 B007       MOV    AL,07
3006 8AIE0010   MOV    BL,[1000]
300A 00D8       ADD    AL,BL
300C A20010     MOV    [1000],AL
300F E9FDFF     BR      300F
*
```

If the display end address is omitted, 20 lines of the program are displayed from the specified start address. If the start address is omitted, the display starts at the next address after the end of the previous display. If both the start and end addresses are omitted, 20 lines are displayed, starting at the address after the end of the previous display.

Run the Program

To execute the program, use the G (Go) command. Breakpoints can be set with the B command and cleared with the N command. The T (Trace) command can be used to execute the program one instruction at a time. These commands used PS for segments.

```
*B3006↵ .................... Set a break point at &H3006.
*B300F↵ .................... Set another break point at &H300F.
*B↵ ........................ The B command with no arguments dis-
                             plays the current break point(s).
B=3006 300F
*G3000↵ .................... Begin execution at &H3000. If the CPU
                             encounters a break point, execution is
                             stopped and the current contents of the
                             flags and registers are displayed.
*T↵ ........................ Execute the next instruction.
*T3000↵ .................... Execute the instruction at &H3000.
```

Displaying Memory and Register Contents

The contents of memory can be displayed with the `D` (Dump) command. For example:

`*D4000.4008↵` . . . . Display the contents of memory from `&H4000` to `&H4008`.

```
4000 - 00 07 00 00 12 34 FB C2
4008 - 5A
```

If the end address is omitted, only one byte is displayed. If the start address is omitted, the contents of memory from the address after the end of the previous display to the end address are displayed.If both the start and end addresses are omitted, 8 bytes are displayed starting at the address after the end of the previous display.

The contents of the registers and flags can be displayed with the `R` (Register) command.

```
*R↵
R2 R1 R0 V  D  I  B S  Z F1  A F0  P IB C
-- -- -- * -- -- -- * -- -- -- -- -- -- --
AW-FFFF BW-0000 CW-0000 DW -0000 SP-0000 BP -0000
IX-0000 IY-0000 PS-0000 DS0-0000 SS-0000 DS1-0000 PC-3006
*
```

The contents of a register or flag can also be changed:

`*RAW=0005↵` . . . . `AW` is the register or flag name and `0005` is the data.

The register names are: `AW`, `BW`, `CW`, `DW`, `SP`, `BP`, `IX`, `IY`, `PS`, `DS0`, `SS`, `DS1`, and `PC`.

The flag names are: `R2`, `R1`, `R0`, `V`, `D`, `I`, `B`, `S`, `Z`, `F1`, `A`, `F0`, `P`, `IB`, and `C`.

Data must be 4 characters or less of hexadecimal numbers; leading zeros may be omitted.

Saving and Loading Programs

The contents of the machine language area may be saved to the memory card or the connected terminal with the `S` (Save) command. The syntax of the command is:

`S` *device* [*format*] *start-address*`.` *end-address*`.` *file-name*

Device `F` is the memory card; `R` is the terminal.

Format `H` is hexadecimal; format `S` is Motorola S-records. (If *format* is omitted, the default is S-records.) *Format* `H` must be used for Memory Cards.

For example,

`*SFH4000.400F.FILE3↵` . . . . . Save the contents of memory locations `&H4000` to `&H400F` in hexadecimal format on the memory card in a file named `FILE3`.

Note When the CVSS is used and the program is to be saved to the terminal, the `S` command does not have to be entered by the user because the save operation is performed through the menu screen of the Terminal Pack.

To load a file from the memory card or terminal, use the `L` (Load) command. The syntax of the command is:

`L` *device* [*format*] *offset*`.` *file-name*

Device and *format* are the same as in the `S` command. *Offset* can be used to force the contents of the file to be stored in a different location in memory. (The contents of the file are placed at *saved-address* + *segment-address* (DS0) + *offset*.)

For example,

`*LFH0.FILE3↵` . . . . The contents of hexadecimal `FILE3` on the memory card are loaded into memory.

When saving to or loading from EEPROM, use the `ROMSAVE`/`ROMLOAD` commands for the entire source code (S code) area and the BASIC program.

To check whether the program has been correctly saved or loaded, use the `X` command immediately after the `S` or `L` command.

If an error has occurred, an error message (`SAVE ERROR` or `LOAD ERROR`) is displayed.

Common Programming Mistakes

Keep the following points in mind when developing a machine language subroutine:

- Don’t forget to allocate memory for the machine language program with the `MSET` command.
- Remember that the storage address for the machine language program is the sum of the segment address (DS0) and the offset (the specified address).
- Be careful not to erase or damage the system and BASIC program areas by assembling or loading to the wrong section of memory.
- Before calling the machine language routine, use `DEF SEG` to define the machine language routine’s segment address.
- To return from the machine language routine to the BASIC program, use the `RETF` instruction. Make sure that the value of the stack pointer is the same as when the machine language routine was called. Other registers and flags are restored by the system.
- Do not disable interrupts in the machine language program.
- To use some of the memory allocated by the `MSET` command as a work area, turn OFF the memory protect switch (write enable status).
- Instructions that are used for transferring data to or from the CPU Unit, or for port operation such as `PC READ` and `PC WRITE` or `PRINT` and `INPUT` cannot be programmed using the machine language.

6-3-3 Examining and Altering Memory with BASIC

To write data to the machine language program area from a BASIC program, use the `POKE` statement. To read data, use `PEEK`.

Note The memory protect switch must be turned OFF for `POKE` to work.

Reading & Writing Memory

Before reading or writing data in the machine language program area, define a segment address with `DEF SEG`.

`10 DEF␣SEG = &H400 . .......` Use segment `&H400`

To write data, use the `POKE` statement. (The memory protect switch must be turned OFF.)

`30 POKE &H100, &H41 . ......` Store `&H41` at location `&H4100` (segment `&H400` + offset `&H100`).

To read the contents of memory, use the `PEEK` statement.

`40 N = PEEK(&H100) . .......` Read the contents of location `&H4100` and store in `N`.

Here is a simple program that stores a value in memory, then reads it back and displays it:

```
10 PARACT 0
20 DEF SEG = &H400
30 POKE &H100, &H41
40 N = PEEK(&H100)
50 PRINT CHR$(N)
60 END
70 END PARACT
```

In this program, addresses and data are specified as hexadecimal numbers. However, they can also be specified in other formats or as variables.

Note that the data read or written by the `PEEK` and `POKE` instructions in byte units.

6-3-4 Calling a Machine Language Subroutine

To call the machine language subroutine from the BASIC program, use the `CALL` statement or the `USR` function. Machine language subroutines that return a value to the BASIC program must be called by the `USR` function.

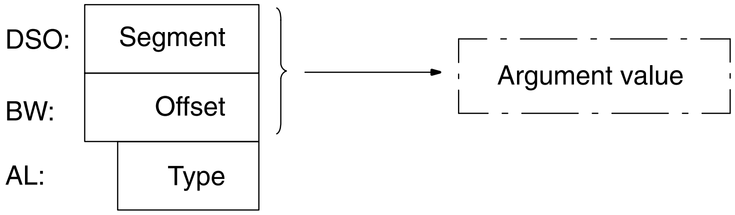
USR

Ten `USR` functions, `USR0` through `USR9`, can be defined and used. Before using any `USR` function, the machine language subroutine segment must be defined with `DEF SEG`. Then, the start address for each subroutine must be defined with `DEF USR`.

For example:

```
100 DEF SEG = &H400
110 DEF USR1 = &H100 . ..... USR1 starts at offset &H100 in segment
                             &H400 (absolute address &H04100).
120 N = USR1( 5 ) . ........ Call the subroutine, passing it the argu-
                             ment 5. The result is stored in N.
```

When the machine language subroutine is called, information about the argument is passed as follows:



The argument type in AL will be one of these values:

0 : Integer

1 : Single-precision floating point

2 : Double-precision floating point

3 : Character variable

The beginning of the argument value is specified by the address in DS0 and BW. For information about the argument value's storage format in memory, refer to *6-3-5 Storage Formats*.

The machine language subroutine must return its result in the same type and using the same area in memory.

Sample Program

This program uses a machine language subroutine which squares an integer to print a list of squares from 1 to 10. However, the program does not use the `USR` argument to pass the number to square; rather, it stores the number in a fixed location (with `POKE`). The machine language routine gets it from that location and places the result at another fixed location.

```
10 PARACT 0
20 DEF SEG=&H400 . ......... Define segment address (&H400)
30 DEF USR1 = &H100 . ...... Define subroutine start address (&H100)
40 FOR I = 1 TO 10
50 POKE &H200, I . ......... Save the value to square at offset
                             &H200 (absolute location &H4200)
60 N = USR1(0) . ........... Call the subroutine
70 A = PEEK(&H202) . ....... Get the squared value which the subrou-
                             tine has stored at offset &H202
80 PRINT I; A . ............ Print the number and its square
90 NEXT I
100 END
110 END PARACT
```

Here is the machine language portion of the program. It must be loaded in memory at segment &H400, offset &H100 (absolute location &H4100).

```
MOV AW,PS . ................. Make data segment equal to program
                              segment
MOV DS0,AW
MOV AL,[200] . .............. Get the value to square from &H200
MUL AL . .................... Square the value
MOV [202],AL . .............. Save result at &H202
RETF . ...................... Return to BASIC program
```

Note When writing machine language programs, allocate space with the MSET instructions and remember that the storage address is the segment address (DS0) plus the offset (the specified address). DS0 will be 0050 when the machine language monitor mode is entered. If a program is input immediately, the first offset address will be 4100 – 0500, or 3C00.

CALL

The CALL statement executes a machine language subroutine from the BASIC program. Before using CALL, the machine language subroutine's segment must be specified with the DEF SEG statement. Then, the subroutine's offset address must be stored in an integer variable which will be used in the CALL.

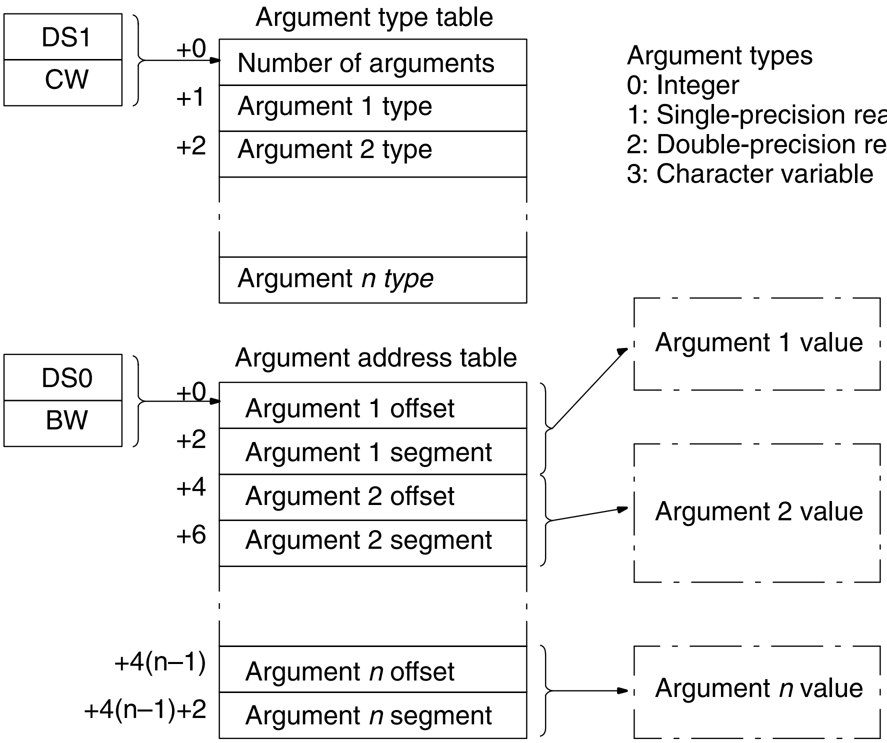
Several argument values may be passed to the subroutine when it is CALLed. The BASIC Unit passes information about the arguments in type tables and address tables; the table addresses are passed in DS0, BW and DS1, CW as follows:

DS1 contains the argument type table segment.

CW contains the argument type table offset.

DS0 contains the argument address table segment.

BW contains the argument address table offset.



The argument value address in the address table indicates the beginning of the argument value. For information about the argument value storage formats, refer to *6-3-5 Storage Formats*.

The area of the argument to which the execution result of the machine language program has been given is returned to the BASIC program as the value of the same type.

Sample Program

The following program inputs two numbers (A% and B%) and calls a machine language subroutine which stores the larger of the two numbers in C%.

```
10 PARACT 0
20 DEF SEG = &H300
30 OFADR% = &H200
40 A% = 0
50 INPUT B%,C%
60 CALL OFADR%(A%,B%,C%)
70 PRINT A%
80 END
90 END PARACT
```

Here is the machine language subroutine. It must be loaded at segment &H300, offset &H200 (absolute address &H3200).

```
MOV CW,A[BW] . .............. Get C% argument segment
MOV DS1,CW
MOV IX,8[BW] . .............. Get C%argument offset
DS1:
MOV AW,[IX] . ............... Get C% argument value
MOV CW,6[BW] . .............. Get B% argument segment
MOV DS1,CW
MOV IX,4[BW] . .............. Get B% argument offset
DS1:
CMP AW,[IX] . ............... Compare values (C% to B%)
BGE 321B  . ................. Jump if C% >= B%
DS1:
MOV AW,[IX] . ............... Move B% value to AW
MOV CW,2[BW] . .............. Get A% argument segment
MOV DS1,CW
MOV IX,0[BW] . .............. Get A% argument offset
DS1:
MOV [IX],AW . ............... Move AW to A% area
RETF
```

Note

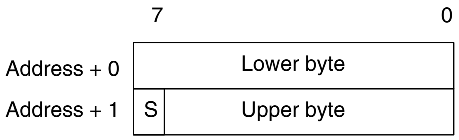
1. To return from the machine language subroutine to the BASIC program, be sure to use the RETF instruction (op code &HCB). This is because the machine language program segment is different from the BASIC program segment, so the RET instruction (op code &H3C) will not work. If a subroutine is used within the machine language program, near CALL and RET instructions may be used.
2. Remember that the storage address for the machine language program is the sum of the segment address (DS0) and the offset (the specified address).

6-3-5 Storage Formats

Variables are stored in memory as follows depending on their types:

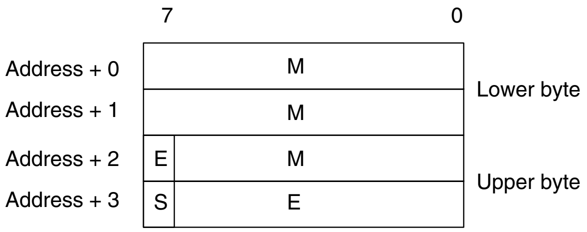
Integers

Integers are stored as two-byte (16-bit) 2's complement numbers. The low-order byte is stored in the lower-addressed of the two bytes occupied.



Single-precision Floating Point Values

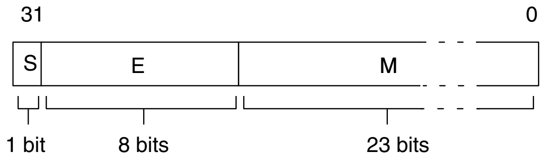
Single-precision floating point values are stored in four consecutive bytes (32 bits), in IEEE 32-bit floating point format.



S: sign bit (0: positive, 1: negative)

E: exponent (8 bits, offset 127)

M: mantissa (23 bits)

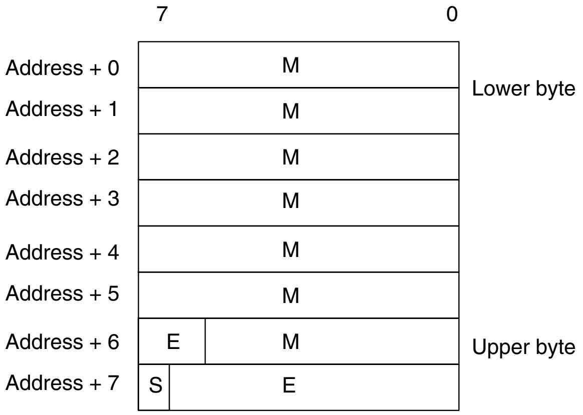


Actual value = $(-1)^S 2^{E-127}(1.M)$

Note: Binary value

Double-precision Floating Point Values

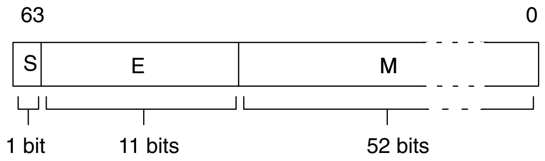
Double-precision floating point values are stored in eight consecutive bytes (64 bits), in IEEE 64-bit floating point format.



S: sign bit (0: positive, 1: negative)

E: exponent (11 bits, offset 1023)

M: mantissa (52 bits)

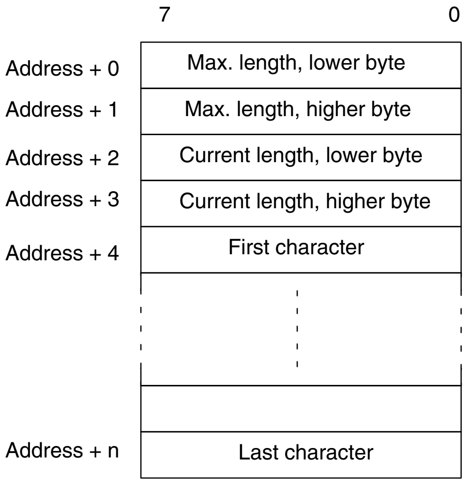


Actual value = $(-1)^S 2^{E-1023}(1.M)$

Note: Binary value

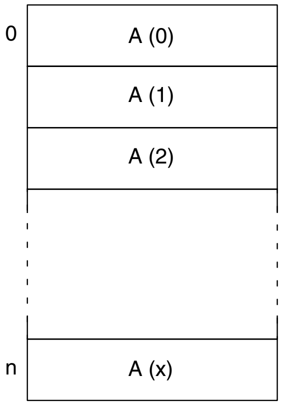
Character Strings

Character strings are stored with 4 bytes of header information (2 bytes for maximum length and 2 bytes for current length), followed by the characters in the string. A pad byte is appended if necessary so that the number of bytes used is even. The pad byte's value is undefined.



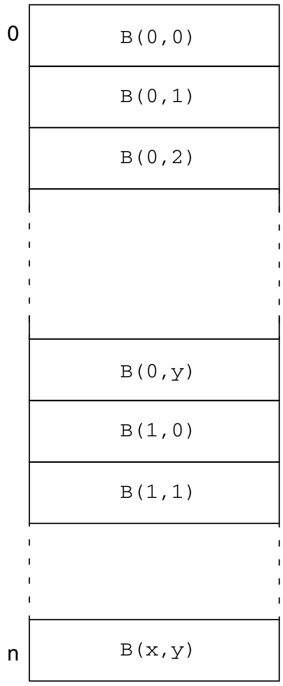
Array Values

Arrays are stored contiguously in memory; each element of the array occupies the same number of bytes. (The size of each element is the same as the size for a simple value of the same type.)



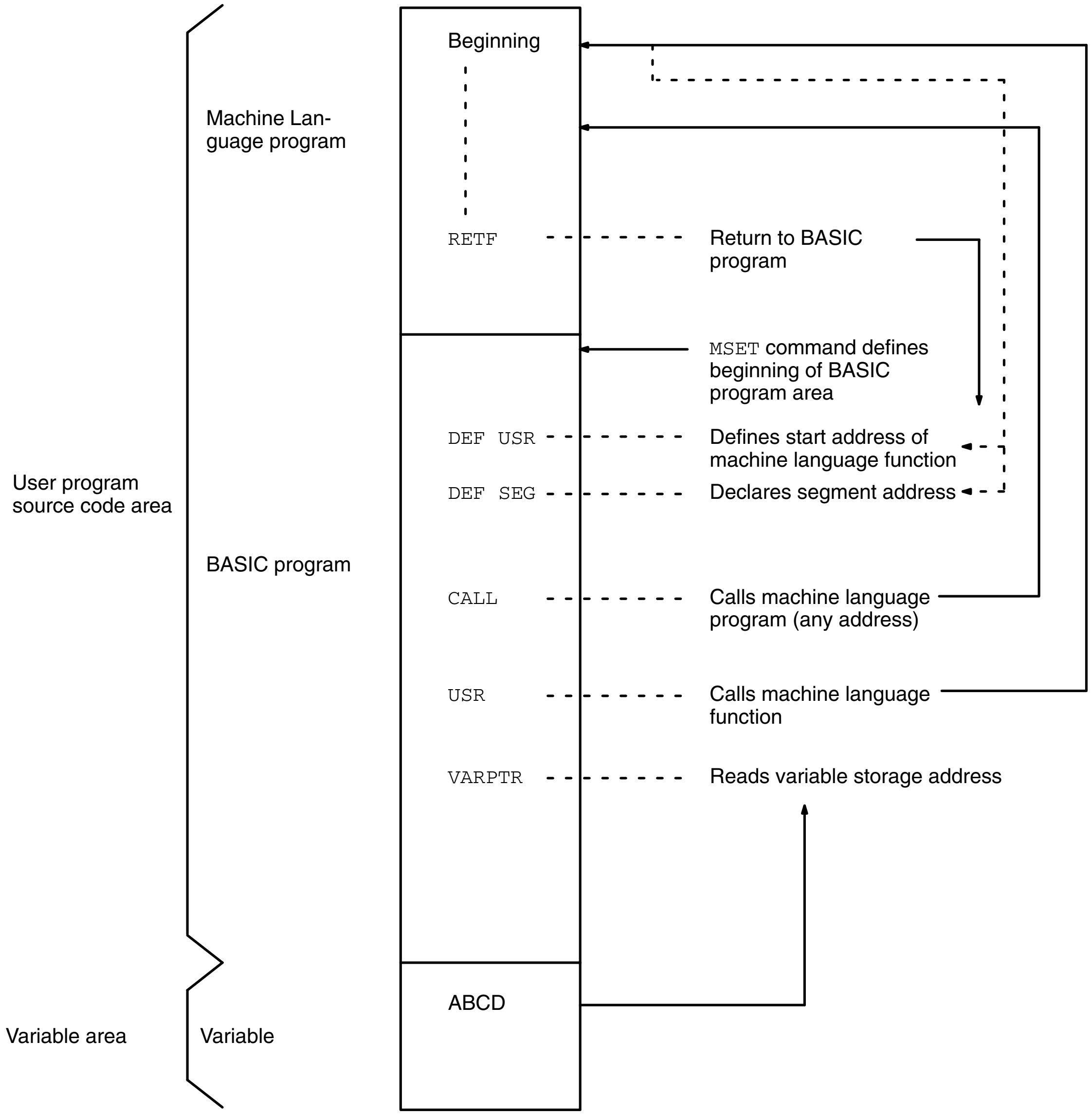
Multi-dimensional Array

Multi-dimensional arrays are stored in row-major form; that is, all the elements of one row are stored before the first element of the next row. The diagram below shows the layout of an X×Y array.



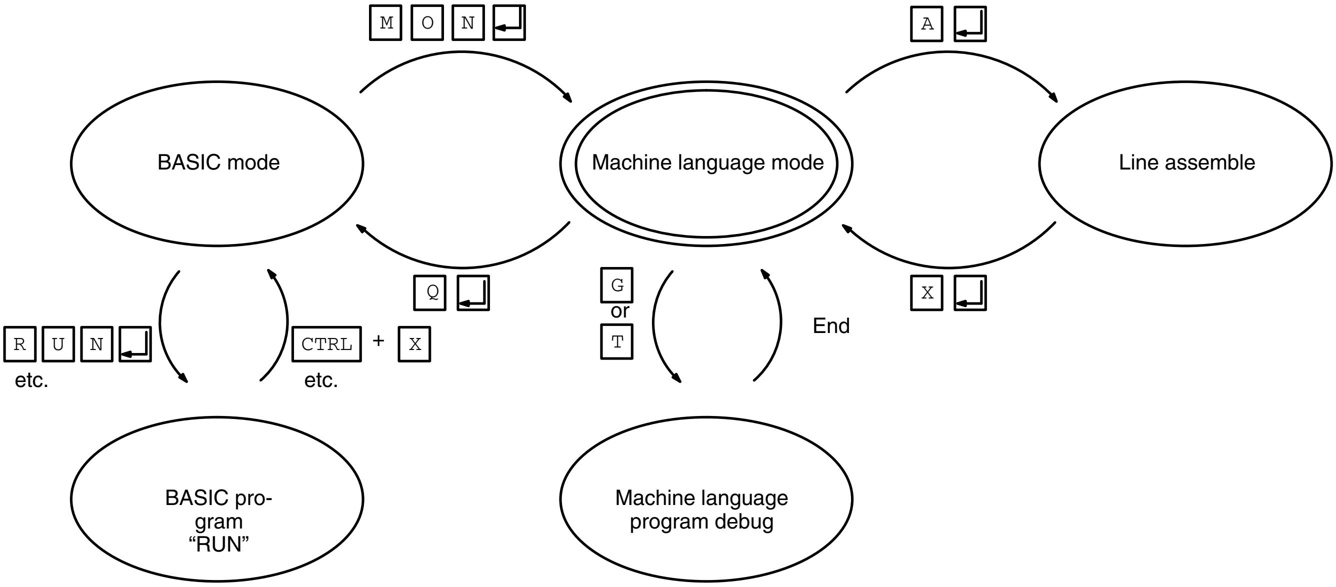
6-3-6 Machine Language Programming Summary

To call a machine language program from the BASIC program, use the CALL statement or USR function.


Beginning
Machine Language program
RETF
Return to BASIC program
MSET command defines beginning of BASIC program area
DEF USR
Defines start address of machine language function
DEF SEG
Declares segment address
User program source code area
BASIC program
CALL
Calls machine language program (any address)
USR
Calls machine language function
VARPTR
Reads variable storage address
ABCD
Variable area
Variable

Machine Language Monitor Mode and BASIC Mode

This diagram shows the commands that are used to move between the machine language monitor mode and BASIC mode.



When `MON↵` is typed in the `OK` display status or command input status in BASIC mode, the Unit enters machine language mode. At this time, the BASIC RUN indicator goes off.

To return to BASIC mode, type `Q↵` at the `*` prompt.

6-3-7 Machine Language Monitor Commands

This table lists the machine language monitor commands and gives a brief description of each command's function. Detailed descriptions of each command may be found in *Appendix E Machine Language Monitor Reference*.

Command	Function
D	Displays memory contents at specified address
W	Changes memory contents at specified address
M	Transfers memory contents
C	Compares memory contents
A	Assembles one line
I	Disassembles
S	Saves machine language program
L	Loads machine language program
V	Verifies machine language program
X	Checks result of saving, loading, or verifying machine language program
B	Sets or displays break point
N	Cancels break point
G	Executes machine language program
T	Executes one step of machine language program
R	Displays or changes register contents
K	Addition or subtraction in hexadecimal number
ESW	Sets or displays memory switch

6-4 PC Communications

To transfer data between the CPU Unit and BASIC Unit, the `PC READ` or `PC WRITE` command is usually used from the BASIC Unit. However, the CPU Unit can also interrupt the BASIC Unit by executing the SEND(192) or RECV(193) instruction or by using FINS commands.

6-4-1 SEND(192) and RECV(193)

The NETWORK SEND (SEND(192)) and NETWORK RECEIVE (RECV(193)) instructions can be used in the ladder-diagram program of the CPU Unit to send data to or receive data from a BASIC Unit. Communications with the CPU Unit using these instructions are handled as interrupts by the BASIC Unit.

Note

1. No signal is generated to indicate the end of the `SEND(192)` or `RECV(193)` instruction. If it is necessary to confirm completion of `PC READ` or `PC WRITE` in the CPU Unit program, confirmations data can be written to specific area in the CPU Unit and checked by the CPU Unit program.
2. To prevent communications problems when executing more than one `SEND(192)` or `RECV(193)` instruction, use a different port for each instruction or write the CPU Unit program to ensure that only one instruction is executed at a time.
3. It is more efficient to combine data transfer operations to reduce the number of `PC READ` and `PC WRITE` commands.
4. Only one write request (`PC WRITE`) is executed by the CPU Unit during each CPU Unit cycle. If more than one request is received, the other write requests must wait until the next cycle. This includes requests from other BASIC Units, other CPU Bus Units, and Link Units (SYSMAC LINK, SYSMAC NET, etc.)5

CPU Unit Interrupt Processing Program

The `ON PC GOSUB` statement is used to define a service routine for PC interrupts.

`ON PC(2) GOSUB 1000 . .....` `2` is the interrupt number and `1000` is the first line number of the interrupt routine.

Interrupts 1 to 15 can be specified.

To generate an interrupt from the CPU Unit, the `SEND(192)` or `RECV(193)` instruction is executed by the user program in the CPU Unit.

When the CPU Unit generates an interrupt, the `PC READ` command is used to read the data from the CPU Unit:

`PC READ ″S10H4″; A(0) . ....` `S10H4` is the format and `A(0)` is the variable which will receive the data.

When an interrupt has been generated from the CPU Unit, use the `PC WRITE` command to write data to the CPU Unit:

`PC WRITE ″S10H4″; B(0) . ...` `S10H4` is the format and `B(0)` contains the data to send.

The format is specified as shown in this table. For details, refer to the *BASIC Unit Reference Manual (W207-E1)*.

Name	Format	Meaning
I	*m*I*n*	*n*-digit decimal data of *m* words (*n*: 1 to 4)
H	*m*H*n*	*n*-digit hexadecimal data of *m* words (*n*: 1 to 4)
O	*m*O*n*	*n*-digit octal data of *m* words (*n*: 1 to 4)
B	*m*B*n*	*n*th bit data of *m* words (*n*: 0 to 15)
A	*m*A*n*	ASCII character data specified by *n* of *m* words (*n*: 1 to 3)
S	S*mXn*	*n*th (*n*th bit) data specified by *X* of *m* words (Type S is of array type of type I, H, O, or B, and *X* indicates I, H, O, or B.)

- If *m* is omitted, 1 is assumed.
- Make sure that 1 word of types I, H, O, and B corresponds to 1 variable.
- Type A must correspond to 1 variable in format units.
- Type S correspond to 1 array variable in word units, but must correspond to 1 array variable in format units for description. Use one-dimensional array as the array variable.

Transferring Data from the CPU Unit

To transfer data from the CPU Unit, the CPU Unit interrupts the BASIC Unit with the `SEND(192)` instruction.

```
      (192)
────[ SEND     S        D        C ]
```

S: 1st source word

D: 1st destination word

C: 1st control word

Word	Bits 00 to 07	Bits 08 to 15
C	Number of words (1 to 0990 in 4-digit hexadecimal, i.e., `$0001` to `$03DE`)	
C+1	Destination network address (0 to 127, i.e., `$00` to `$7F`)	Bits 08 to 11: Interrupt number (`$1` to `$F`) Bits 12 to 15: Set to `0`.
C+2	Destination unit	Destination node address
C+3	Bits 00 to 03: No. of retries (0 to 15 in hexadecimal, i.e., `$0` to `$F`) Bits 04 to 07: Set to 0.	Bits 08 to 11: Transmission port number (`$0` to `$7`) Bit 12 to 14: Set to 0. Bit 15: ON: No response. OFF: Response returned.
C+4	Response monitoring time (`$0001` to `$FFFF` = 0.1 to 6553.5 seconds)	

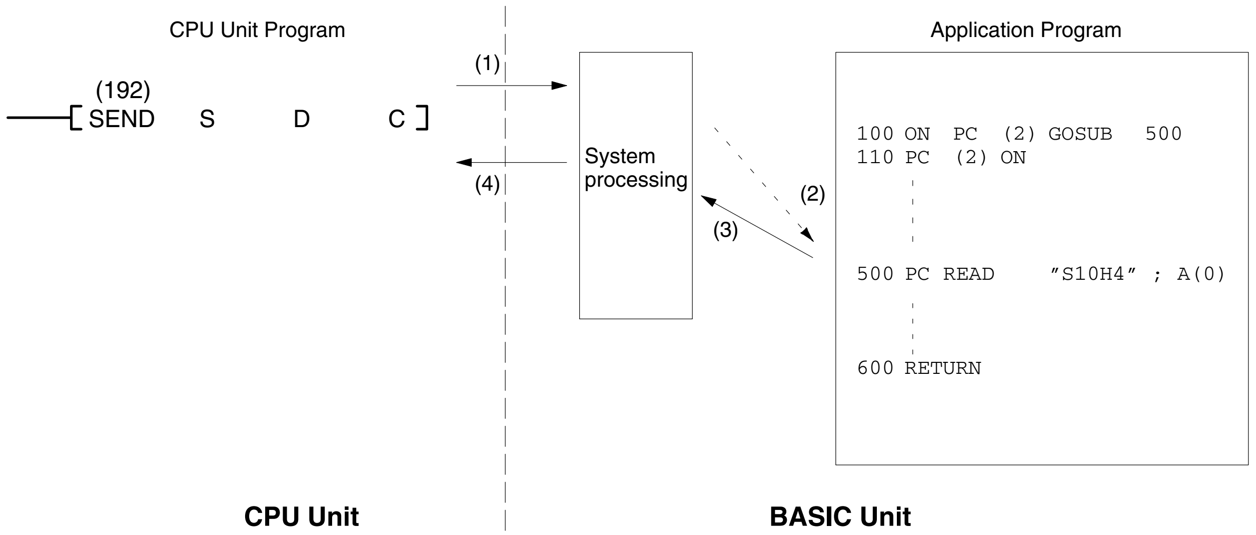
- The interrupt number must be the same as that used in the `ON PC GOSUB` and `PC ON`, `OFF`, and `STOP` statements in the BASIC program. The interrupt number must be a hexadecimal number from 1 to F.
- The destination Unit specification is the BASIC Unit's unit number plus 16 (a hexadecimal number from 10 to 1F).
- Set D to 0000; the BASIC Unit ignores this parameter.
- Refer to the *CV-series PC Operation Manual: Ladder Diagrams* for further details.

Here is the procedure for interrupt-driven data transfer from the CPU Unit to the BASIC Unit:

1, 2, 3...

1. Transfer data from the CPU Unit by executing the `SEND(192)` instruction with interrupt number set in `C+1`.
2. The BASIC Unit will be interrupted when the data arrives, and the PC interrupt service routine defined by the `ON PC GOSUB` statement will be called.
3. Data of a predetermined length is read with the `PC READ` instruction and is stored in the variable(s). The length set for the `PC READ` instruction must be the same as that set for the `SEND(192)` instruction.

4. The `PC READ` command returns a response (`1`) to the CPU Unit.



```
100 ON  PC  (2) GOSUB  500
110 PC  (2) ON

500 PC READ    "S10H4" ; A(0)

600 RETURN
```

Transferring Data to CPU Unit

To receive data from the BASIC Unit, the CPU Unit interrupts the BASIC Unit with the `RECV(193)` instruction.

```
     (193)
----[ RECV    S       D       C ]
```

S: 1st source word

D: 1st destination word

C: 1st control word

Word	Bits 00 to 07	Bits 08 to 15
C	Number of words (1 to 0990 in 4-digit hexadecimal, i.e., `$0001` to `$03DE`)	
C+1	Source network address (0 to 127, i.e., `$00` to `$7F`)	Bits 08 to 11: Interrupt number (`$1` to `$F`) Bits 12 to 15: Set to 0.
C+2	Source unit	Source node address
C+3	Bits 00 to 03: No. of retries (0 to 15 in hexadecimal, i.e., `$0` to `$F`) Bits 04 to 07: Set to 0.	Bits 08 to 11: Transmission port number (`$0` to `$7`) Bit 12 to 14: Set to 0. Bit 15: ON: No response. OFF: Response returned.
C+4	Response monitoring time (`$0001` to `$FFFF` = 0.1 to 6553.5 seconds)	

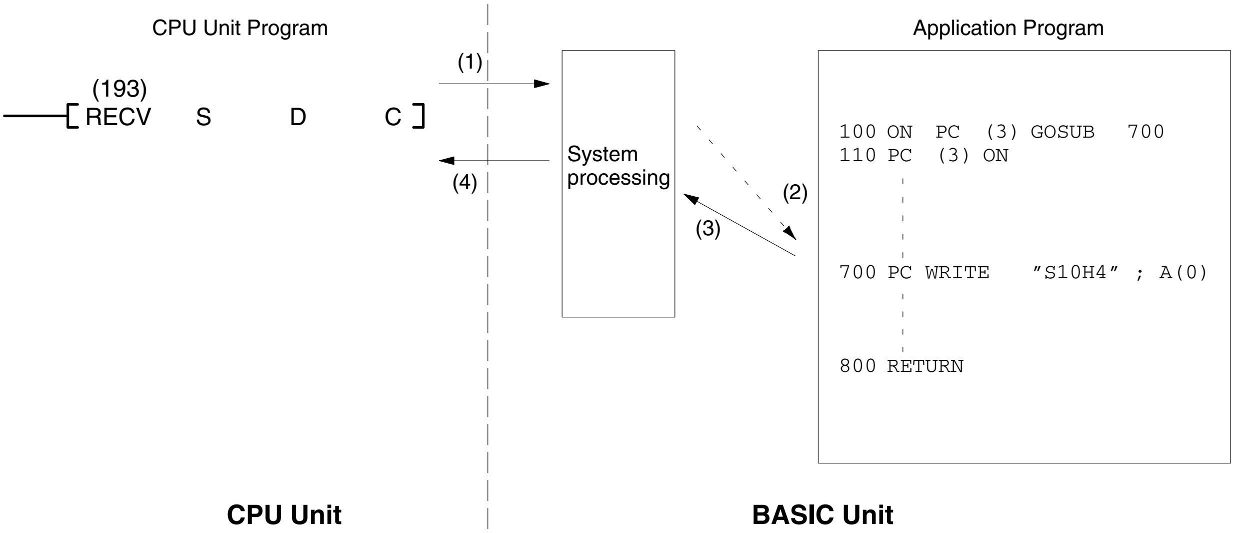
- The interrupt number must be the same as that used in the `ON PC GOSUB` and `PC ON`, `OFF`, and `STOP` statements in the BASIC program. The interrupt number must be a hexadecimal number from 1 to F.
- The source unit specification is the BASIC Unit's unit number plus 16 (a hexadecimal number from 10 to 1F).
- Set S to 0000; the BASIC Unit ignores this parameter.

Here is the procedure for interrupt-driven data transfer from the BASIC Unit to the CPU Unit:

1, 2, 3...

1. Execute the `RECV(193)` instruction from the CPU Unit with the interrupt number in C+1.
2. The PC interrupt service routine defined with `ON PC GOSUB` will be called when the `RECV(193)` instruction has been executed.

3. Data of the predetermined length is sent from the BASIC Unit with the `PC WRITE` instruction. The length set for the `PC WRITE` instruction must be the same as that set for the `RECV(193)` instruction.
4. The `PC WRITE` instruction returns a response to (`1`) to the CPU Unit.



```
100 ON  PC  (3) GOSUB   700
110 PC  (3) ON
700 PC WRITE   "S10H4" ; A(0)
800 RETURN
```

6-4-2 CV-series (FINS) Commands

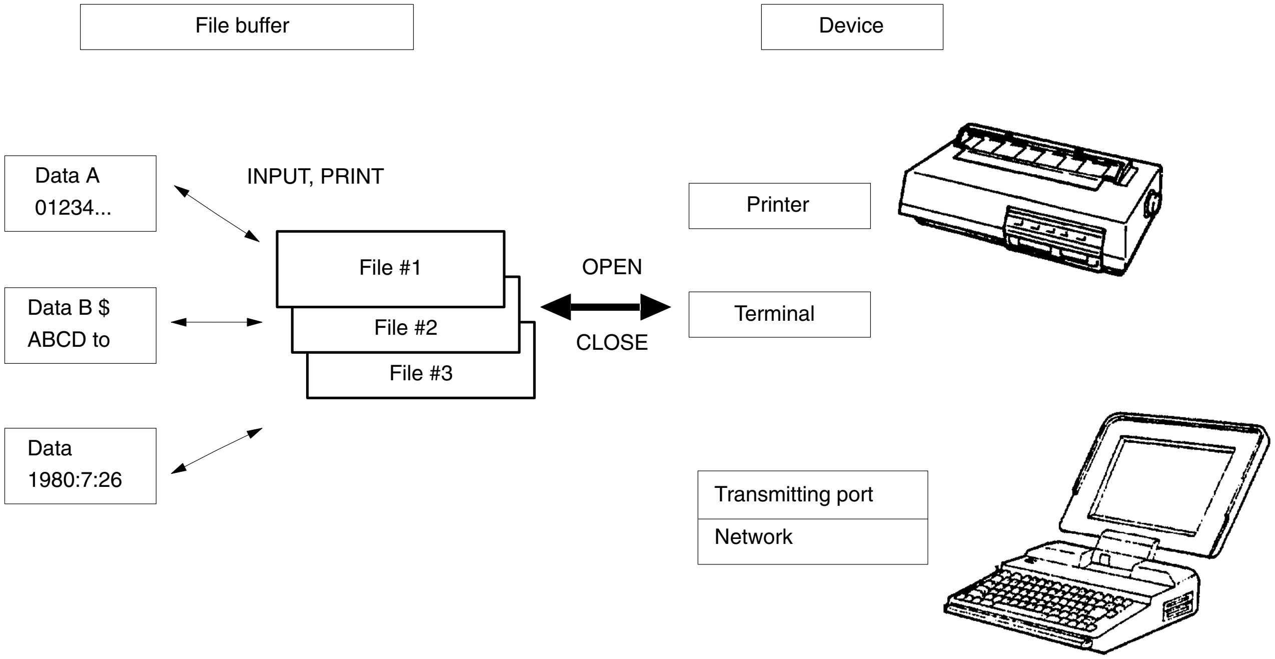
The BASIC Unit supports automatic processing for certain FINS commands transmitted via PC networks. Refer to the *FINS Command Reference Manual* for details.

SECTION 7
Peripherals

This section information relating to the use and programming for the peripheral devices. The GB-IB Interface programming is also provided for use with the peripherals.

7-1 Peripheral Devices

Various devices such as a terminal, printer, communication port, and network can be connected to the BASIC Unit. These devices can be opened, and data can be read and written, in the same way as a regular file.



As shown in this figure, the devices can be associated with a file buffer from 1 to 15 by the `OPEN` statement.

After that, when data is read from or written to the file buffer with commands such as `INPUT` and `PRINT`, the data is automatically sent to or received from the device. When the device is no longer necessary, it can be dissociated from the file buffer by the `CLOSE` statement.

To open a device, use one of the following names in the `OPEN` command:

Name	Device
`COM1:`	Communication port 1
`COM2:`	Communication port 2
`COM3:`	Communication port 3
`KYBD:`	Terminal keyboard
`SCRN:`	Terminal screen
`LPRT:`	Printer
`FINS:`	Network

The name used for the communications port can also include information specifying the communications parameters to use on the port.

7-1-1 Using Devices

This section describes how to use the devices.

Opening a Device

Before using a device, open it with the `OPEN` statement.
For example, open a communications port as follows:

`OPEN ″COM2:″ AS #4 . ......` `″LPRT:″` is the device name, and `#4` is the file number.

When the device has been opened, it is associated with the file buffer of the specified file number. Therefore, the same file number cannot be used by any other file or device until the first device is closed.
File numbers must be integers between 1 and 15.

Note To establish communication between BASIC Units, specify `FIN` as the device name in an `OPEN` command, a network address, node address, and Unit address, and send or receive data using the `PRINT` or `INPUT` statements.

Communication Ports

The communication ports can be opened by the `OPEN` statement using device name `COM1:`, `COM2:`, or `COM3:`. For example,

```
OPEN ”COM1:9600,E,8,2,XN” AS #4
```

`COM1:` is the device name, `9600,E,8,2,XN` is the communications setting (described below), and `#4` is the file number.

A character string can be specified after the device name to set various communications parameters such as the baud rate, bit length, and parity.

If these parameters are not specified, the value set by the memory switch is used for the baud rate; the character length is 8 bits, 2 stop bits are used, and flow control is disabled.

Specify communications parameters as follows:

```
9600,E,8,1,N,RS,CS10,DS0,LF
```

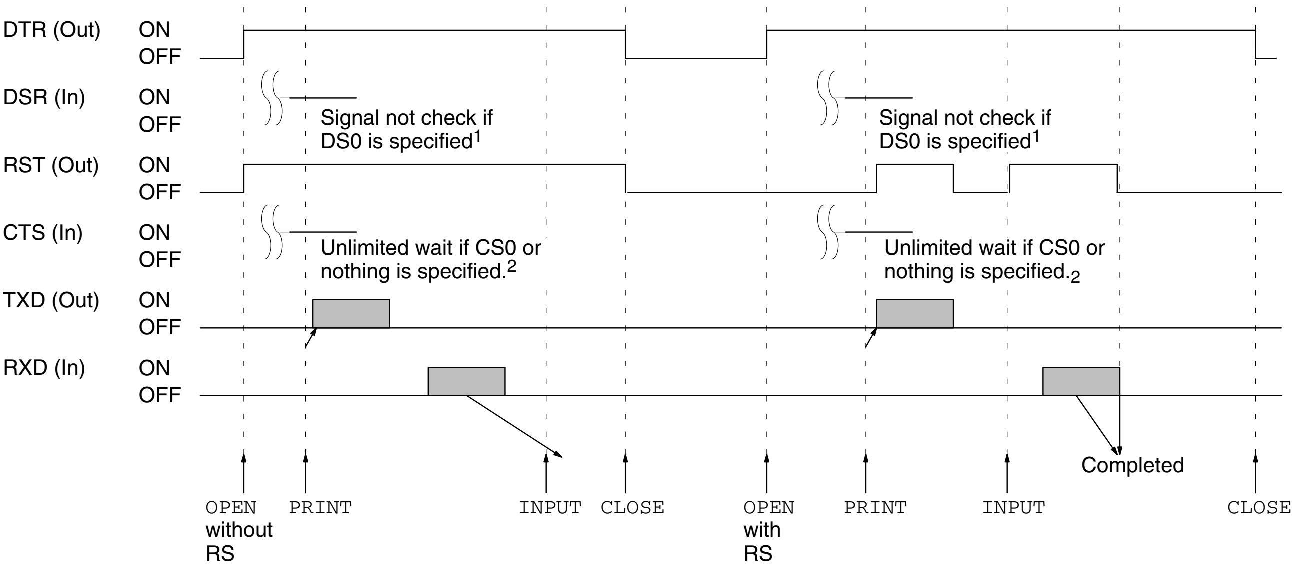
Here `9600` is the baud rate, `E` is for even parity, `8` is the data length, `1` is the number of stop bits, `N` controls XON/XOFF flow control, `RS` controls the RTS signal, `CS10` monitors transmissions, `DS0` controls the handling of the DSR signal, and `LF` enables the LF-after-CR function.

The details of the communication control parameters are as follows:

Parameter	Setting	Remarks	Default
Baud rate	300, 600, 1200, 2400, 4800, 9600, 19200	Sets transfer rate (bits/second (bps))	Setting of memory switch. 9600 if memory switch is not set
Parity	E O N	Even parity Odd parity No parity	N
Data length	7 8	7 bits per character 8 bits per character	8
Stop bit	1 2	1 stop bit 2 stop bits	1
XON/XOFF	X XN	Performs XON/XOFF flow control Does not perform XON/XOFF flow control	X
RTS control	RS None	Turns ON RTS (request to send) signal on execution of I/O command. RTS is OFF for all other commands. Always turns ON RTS signal. If a communication port is set as the printer port or terminal port, control using RTS is not possible. In this case, therefore, do not set RTS control.	None
Transmission monitor	CS*n* None	If CTS (clear to send) signal is ON, transmits and waits *n*ms for the end of the send, where *n* is 0 to 30000 (in units of 100). When 0 is specified, wait time is indefinite If CTS signal is ON, transmits and waits indefinitely.	CS0
DSR control	DS0 None	Does not check DSR (data set ready) signal Checks DSR signal	None
LF	LF None	Sends line feed character after carriage return Does not send line feed	None

- Communications control using RTS/DTR signals is not possible for the ports set as the terminal and printer ports. To perform communications control using RTS/DTR signals, change the ports set as the terminal and printer ports to ports other than the ones for which RTS/DTR control is to be used. This is done using memory switch 3.
- With the COM3 (RS-422) port, after send processing is completed, approximately 60 ms is required until receive processing is possible. Be sure to allow for this time.

The timing of the communications control parameters is shown in the following diagram.


DTR (Out)
DSR (In)
RST (Out)
CTS (In)
TXD (Out)
RXD (In)
ON
OFF
Signal not check if DS0 is specified1
Signal not check if DS0 is specified1
Unlimited wait if CS0 or nothing is specified.2
Unlimited wait if CS0 or nothing is specified.2
Completed
OPEN without RS
PRINT
INPUT
CLOSE
OPEN with RS
PRINT
INPUT
CLOSE

Note: 1. The signal is check if nothing is specified and an “RS-232C not ready” error occurs if the sign is not ON.

2. If CS100 to CS30000 is specified, the system will wait for from 100 ms to 30 s for PRINT to finish. If time expires or the signal goes OFF before PRINT finishes, an “I/O timeout” error will occur.

To send data, use the WRITE or PRINT instructions:

WRITE #4, A$, B$ .... #4 is the file number, and A$ and B$ contain the data to send.

PRINT #4, A$, B$ .... #4 is the file number, and A$ and B$ contain the data to send.

Character data stored in character variables A$ and B$ are output through file buffer 4 in the order of A$ and B$.

To receive data, use the INPUT instruction:

INPUT #4, A$, B$ .... #4 is the file number, and A$ and B$ are the variables in which the data is stored.

Data is read from the device through file buffer 4 and stored in A$ and B$.

The device is dissociated from the file buffer by the CLOSE or END statements:

CLOSE #4 . ... #4 is the file number.

7-1-2 User Indicators

The eight user indicators (0 through 7) on the front panel of the BASIC Unit can be lit or extinguished by the BASIC program.

The system provides a subroutine that controls the indicators. This subroutine is called by setting the segment and address of the subroutine and passing arguments that turn on, off, or blink the indicators.

```
100 ’***INDICATOR CONTROL PROGRAM***
110 PARACT 0
120 ’DEFINITION OF FUNCTION
130 DEF FNINT (X) =- (X<32768)*X- (X>32767) * (X–65536)
140 DEF FNCNV (H%, L%) =FNINT (H%*256+L%)
150 DEF FNWORD (A%) =FNCNV (PEEK (A%+1), PEEK (A%))
160 DEF ENOFF (V%) =FNWORD (V%*4)
170 DEF FNSEG (V%) =FNWORD (V%*4+2)
180 ’LED VECTOR READ
190 DEF SEG=&H0
200 LED%=FNOFF (64)
```

```
210 LEDSEG%=FNSEG (64)
220
      ⋮
500 'LED ON/OFF/BLINK EXECUTION
510 DEF SEG=LEDSEG%: LO%=0
520 CALL    LED% (LO%, LOFF%, LON%, LBLINK%)
530 RETURN
      ⋮
560 END PARACT
```

Note

1. Enter lines 120 through 210 shown above as is (the comment line can be omitted).
2. Set the arguments LON%, LOFF%, and LBLINK% to these values according to the number of the indicator to be controlled:

Indicator No.	0	1	2	3	4	5	6	7
Set value	&H01	&H02	&H04	&H08	&H10	&H20	&H40	&H80

   - To control more than one indicator, the values of the indicators are ORed. For example: indicators 1, 3, and 6 can all be turned on by LON% = &H4A
   - If the same values are set for the arguments in duplicate, each of the arguments is assigned priority as follows:
     LBLINK% > LON% > LOFF%

3. Call the indicator subroutine including the segment definition statement (GOSUB 510).

Example:

```
100 '***INDICATOR CONTROL PROGRAM***
110 PARACT 0
120 'DEFINITION OF FUNCTION
130 DEF FNINT (X) =- (X<32768) *X- (X>32767) * (X-65536)
140 DEF FNCNV (H%, L%) =FNINT (H%*256+L%)
150 DEF FNWORD (A%) =FNCNV (PEEK (A%+1), PEEK (A%))
160 DEF ENOFF (V%) =FNWORD (V%*4)
170 DEF FNSEG (V%) =FNWORD (V%*4+2)
180 'LED VECTOR READ
190 DEF SEG=&H0
200 LED%=FNOFF (64)
210 LEDSEG%=FNSEG (64)
220 '
230 '
240 '
250 LOFF%=&HFF: LON%=0: LBLINK%=0
260 GOSUB *LEDSUB
270 FOR I=0 TO 5000
280 NEXT I
290 LOFF%=0: LON%=&H4A: LBLINK%=0
300 GOSUB *LEDSUB
310 FOR I=0 TO 5000
320 NEXT I
330 LOFF%=&H8: LON%=&H10: LBLINK%=&H82
340 GOSUB *LEDSUB
350 FOR I=0 TO 5000
360 NEXT I
370 GOTO 230
380 '
390 '
400 *LEDSUB     'LED ON/OFF/BLINK EXECUTION
410 DEF SEG=LEDSEG%: LO%=0
420 CALL LED% (LO%, LOFF%, LON%, LBLINK%)
430 RETURN
```

```
440 '
450 '
460 END PARACT
```

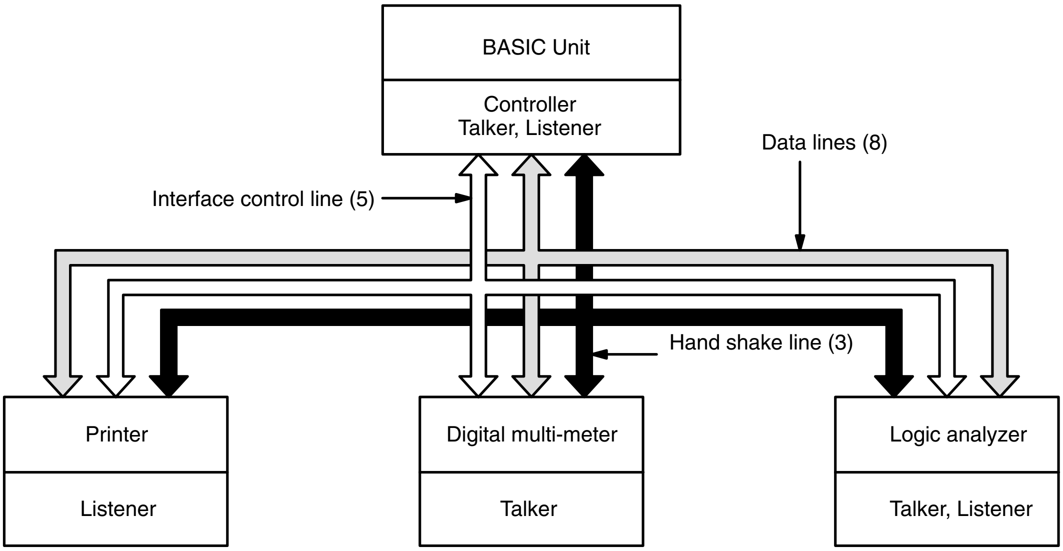
Remarks:

- Lines 120 through 210 and 400 through 430 are as shown on the preceding page.
- Lines 250 and 260 extinguish all indicators 0 through 7.
- Lines 290 and 300 light indicators 1, 3, and 6.
- Lines 330 and 340 extinguish indicator 3, light indicator 4, and blink indicators 1 and 7. At this time, indicator 6 lighted by lines 290 and 300 remains lit.
- The sequence is repeated.

7-2 GP-IB Programming

GP-IB stands for General-Purpose Interface Bus and is an interface used for connecting various kinds of measuring instruments with a computer. This interface is standardized by IEEE-488 and IEC-625. GP-IB has functions called talker, listener, and controller. Talker transfers data, listener receives data, and controller controls the system. Each function has a *my-address* of 0 to 30, and the controller manages the devices in the system using this address.

Each device can have only talker and/or listener functions, or all three functions.



Data are transferred with eight data lines, five interface control lines, and three handshake lines.

Each device can request the controller for a service by using the SRQ line.

The BASIC Unit has 16 BASIC instructions and 2 functions to control the GP-IB, so that data can be transferred with each GP-IB device without having to be concerned with the details of the GP-IB transfer procedures.

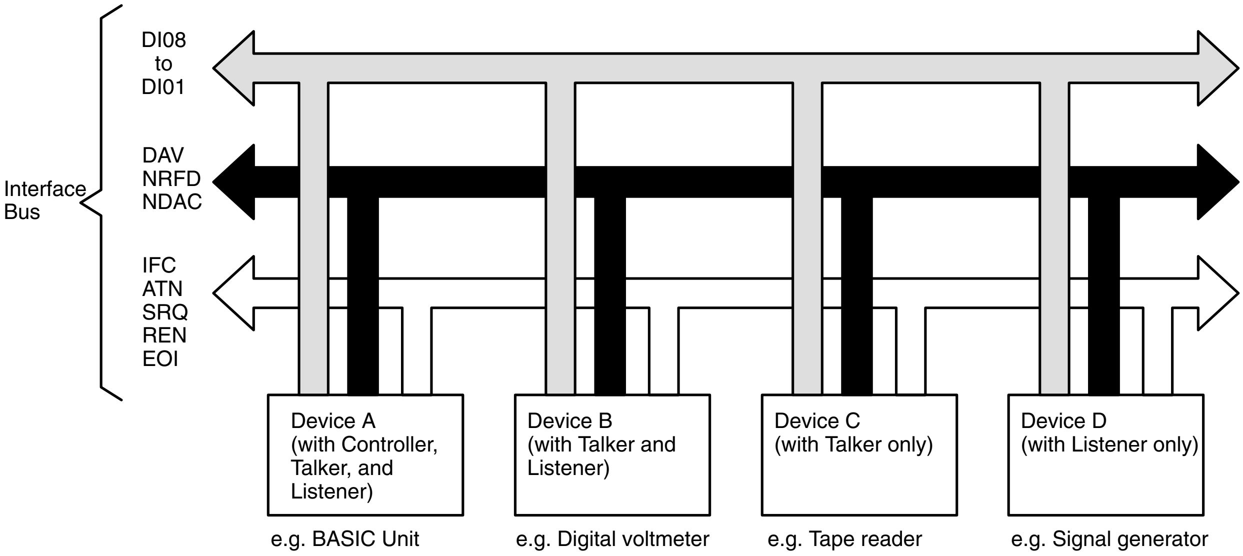
7-2-1 GP-IB System Configuration

In a GP-IB system, all devices are connected in parallel as shown below.

Up to 15 interfaces (devices) can be connected to one system.

The total length of the connecting cables is 20m or the number of devices connected to the same bus ×2m, whichever smaller.

The maximum cable length between two devices is 4m.


Interface Bus
DI08 to DI01
DAV
NRFD
NDAC
IFC
ATN
SRQ
REN
EOI
Device A (with Controller, Talker, and Listener)
Device B (with Talker and Listener)
Device C (with Talker only)
Device D (with Listener only)
e.g. BASIC Unit
e.g. Digital voltmeter
e.g. Tape reader
e.g. Signal generator

The functions (roles) of the devices connected in a GP-IB system can be divided into controller, talker, and listener. These three functions are outlined below.

Controller

This function is to control the entire GP-IB system and is effected by a computer. The controller specifies the destination of data and commands (listener) and the transfer source of data (talker) to control the overall system.

Usually, only one controller is permitted for one system. If more than one device with the controller function is connected to the same bus, only one of the devices can serve as a controller at a time.

The controller that actually operates as a controller is called the active controller. If there are several controllers, one has the special function of system controller.

The system controller is always active when the system is started, and can specify another controller to serve as the active controller if necessary.

Note The BASIC Unit is designed to serve as a system controller and active controller when set in the master mode.

Talker

A talker transfers data under the control of the controller. For example, a talker can be a digital voltmeter that outputs measured values. Only one talker can operate in a system at a time.

Listener

A listener receives data under the control of the controller. For example, a listener can be a printer. Unlike the talker, more than one listener can operate simultaneously in one system.

7-2-2 Signal Lines of GP-IB

The GP-IB consists of 16 signal lines and 8 ground lines. The signal lines are divided into the following three groups based on their functions.

Data Lines (DIO1-DIO8) These 8 lines are the bi-directional data bus.

Handshake Lines (DAV, NRFD, NDAC)

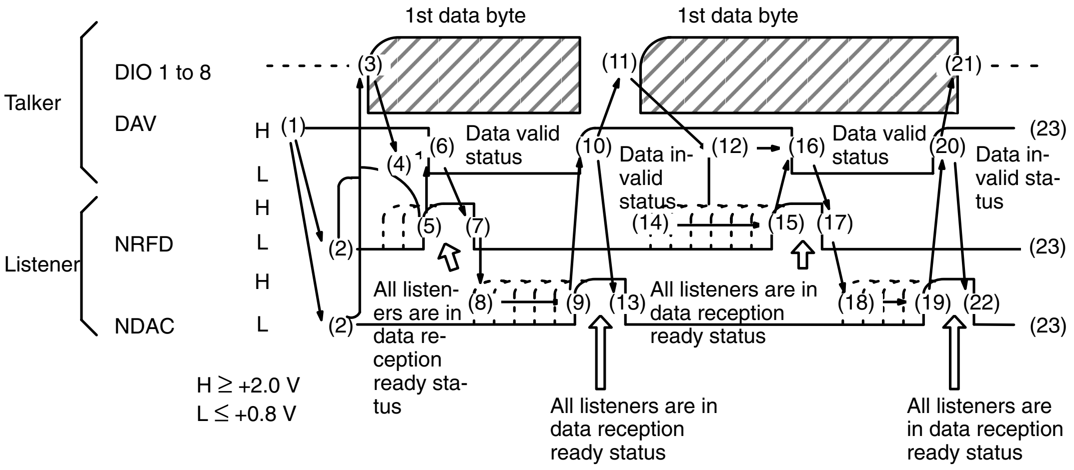
Signal name		Function
DAV	Data Valid	When low, indicates that data on DIO1 through DIO8 sent from controller are valid
NRFD	Not Ready For Data	When low, indicates that listener is busy
NDAC	Not Data Accepted	When low, indicates that listener has not yet completed reception

Interface Control Lines (IFC, ATN, SRQ, REN, EOI)

Signal name		Function
IFC	Interface Clear	Initialize interface when low
ATN	Attention	Indicates command mode when low
SRQ	Service Request	Indicates that device is requesting controller for service when low
REN	Remote Enable	Enables each device to be remotely controlled when low
EOI	End Or Identify	Used as a delimiter when more than 1 byte is transferred. Also used for parallel polling (in combination with ATN)

Three-line Handshaking To synchronize data transfer through GP-IB, a three-line handshaking technique is employed. This handshaking is automatically performed by the GP-IB interface LSI in the BASIC Unit. Therefore, you need not be concerned with it when programming the Unit. However, this section briefly explains the technique for those interested.

A typical timing chart of three-line handshaking between the talker and a listener is shown below as an example.



Operation of Three-line Handshaking

1, 2, 3...

1. The talker makes the DAV line high, indicating that the data is not valid.
2. The listener makes the NRFD line low, indicating that the listener is not yet ready to receive data.
   The listener may also makes the NDAC line low, indicating that reception of data has not been completed.
   At first, the DAV line is high, and NRFD and NDAC are low.
3. The talker sets data on the DIO lines.
4. The talker waits until the DIO lines stabilize.
5. The listener makes the NRFD line high when it is ready to receive.

6. After confirming that the NRFD line is high, the talker makes the DAV line low, indicating that the data on the DIO lines is valid.
7. After confirming that the DAV line is low, the listener makes the NRFD line low, indicating that it has started receiving the data.
   While the data is being transferred, the DAV line is low, NRFD is low, and NDAC is low.
8. Each listener allows its NDAC line to go high when it has finished receiving the data. When all the listeners are finished, the talker will see the NDAC line go high.
9. After confirming that the NDAC line has gone high, the talker makes the DAV line high.

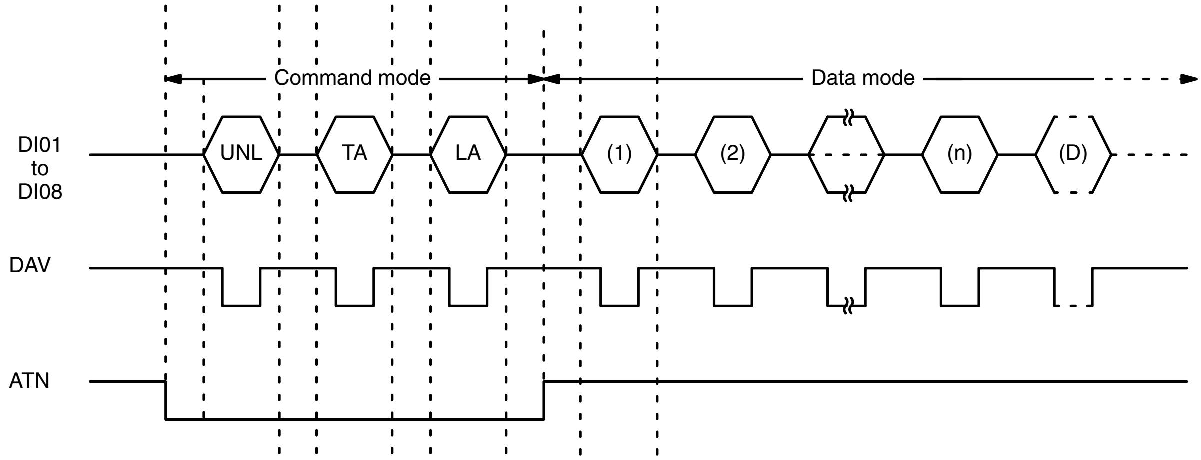
10. through 22.
The next byte is transferred by means of handshaking in the same manner.

7-2-3 Transferring/Receiving Commands and Data

Command and Data Transfer Procedure

To operate a GP-IB device, the controller first makes the ATN line low to transfer a command to all the devices connected to the bus.

When the ATN line is low, the bus enters the command mode, and each device receives the data on the data bus as a command, and performs the operation specified by the command.



UNL: Unlisten command
TA: Talker address
LA: Listener address

(1) to (n): Data
(D): Delimiter (CR + LF, CR, LF or EOI)

1, 2, 3...

1. The controller makes ATN low and transfers the UNL command. This releases all the devices from the current status.
2. The controller transfers TA (talker address) and LAs (listener addresses). This selects a new talker and listeners, which enter the standby status.
3. The controller makes ATN high to set the data mode, in which data is transferred between the talker and listeners.

My Address

Each device in a GP-IB system has an address called *my-address*.

My-address is an integer from 0 to 30, and is used to identify each device.

The controller uses *my-address* to select a talker or listener. The BASIC Unit’s *my-address* is set by the memory switch.

Interface Message

Interface messages are called bus commands or interface commands and are messages to control a GP-IB system. These messages can be transferred only by the controller.

The interface messages are divided into two types: uni-line messages and multi-line messages.

The BASIC Unit automatically transfers an interface message each time it executes a statement. The interface message is information necessary for performing complicated operations.

Uni-line Message

A uni-line message is given a meaning by only one signal line and is transferred using the control bus (ATN, IFC, SRQ, REN, and EOI).

Multi-line Message

A multi-line message is transferred by using the data bus (DIO1 through DIO8) and by means of handshaking. DIO1 through DIO7 of the data bus are used to transfer a multi-line message and DIO8 is ignored. In a narrow sense, a multi-line message is called an interface message.
A multi-line message is a common command of the GP-IB interface, unlike the commands (program codes), which are peculiar to each type of device.

Multi-line messages can be classified into the following five types:

a) Universal command
This command is for all the devices connected to the bus.

b) Address command
This command is for a specified device and is transferred with a listener address specified.

c) Listener address
This is a command to specify a listener.

d) Talker address
This is a command to specify a talker.

e) Secondary command
This command is suffixed to a listener address or talker address to specify the secondary address of an extra listener or talker.

A list of multi-line messages is shown on here.

Codes in Command Mode

								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	COLUMN ROW	0	1	2	3	4	5	6	7
			0	0	0	0	0			a)	a)	b)	b)		
			0	0	0	1	1	GTL	LLO						
			0	0	1	0	2								
			0	0	1	1	3								
			0	1	0	0	4	SDC	DCL						
			0	1	0	1	5	PPC	PPU						
			0	1	1	0	6								
			0	1	1	1	7								
			1	0	0	0	8	GET	SPE						
			1	0	0	1	9	TCT	SPD						
			1	0	1	0	10 (A)								
			1	0	1	1	11 (B)								
			1	1	0	0	12 (C)								
			1	1	0	1	13 (D)								
			1	1	1	0	14 (E)								
			1	1	1	1	15 (F)				UNL		UNT		

a) MLA listener address

b) MLA talker address

Column 0: Address Command Group (ACG)

Column 1: Universal Command Group (UCG)

Column 2 and 3: Listener Address Group (LAG)

Column 4 and 5: Talker Address Group (TAG)

Column 6 and 7: Secondary Command Group (SCG)

Column 1 through 5: Primary Command Group (PCG)

Group	Name		Function
Address command group	GTL	Go To Local	Localizes
	SDC	Selected Device Clear	Initialize
	PPC	Parallel Poll Configure	Sets acknowledge bit of parallel polling function
	GET	Group Execute Trigger	Triggers
	TCT	Take Control	Selects active controller
Universal command group	LLO	Local Lock-out	Disables local function
	DCL	Device Clear	Initialize
	PPU	Parallel Poll Unconfigure	Cancels acknowledge bit of parallel polling function
	SPE	Serial Poll Enable	Sets serial polling mode
	SPD	Serial Poll Disable	Release serial polling mode
Listener address group	UNL	Unlisten	Cancels listener specification
Talker address group	UNT	Untalk	Cancels talker specification

Codes in Data Mode (ASCII Codes)

								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	COLUMN / ROW	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SPACE	0	@	P		p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	”	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	’	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10 (A)	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11 (B)	VT	ESC	+	;	K	[	k	{
			1	1	0	0	12 (C)	FF	FS	,	<	L	\	l	|
			1	1	0	1	13 (D)	CR	GS	–	=	M	]	m	}
			1	1	1	0	14 (E)	SO	RS	.	>	N	^	n	~
			1	1	1	1	15 (F)	SI	US	/	?	O	_	o	

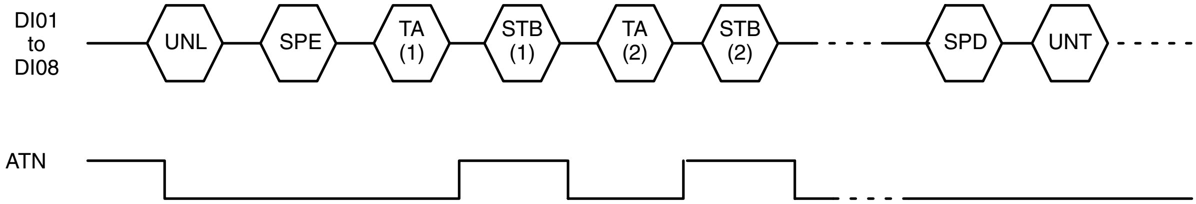
7-2-4 Service Requests

Service Request and Serial Polling

A device connected to the GP-IB can interrupt the controller by making the SRQ line low to request a service.

When the SRQ line has gone low, the controller executes serial polling to find the device that generates the interrupt.

The serial polling is performed with the controller requesting each device to sequentially transfer a status byte, as follows:



SPE: Serial poll enable
TA: Talker address
STB: Status byte

SPD: Serial poll disable
UNT: Untalk

Parallel Polling

Parallel polling is a method by which the controller checks the presence or absence of requests from more than one device (up to eight devices) at a time. One of the DIO through DI8 lines and an acknowledge line is assigned to each device, and the controller checks whether each device is making a request by making ATN and EOI low simultaneously.

7-2-5 Developing a GP-IB Program

The BASIC Unit has 16 commands and 2 functions which control the GP-IB interface.

Data Transfer/Reception Commands

`INPUT@, LINE INPUT@, PRINT@, RBYTE, WBYTE`

Interface Control Commands

`IRESET REN, ISET IFC, ISET SRQ, ISET REN, POLL, PPOLL, CMD DELIM, CMD PPR, CMD TIMEOUT`

SRQ Interrupt Commands

`ON SRQ GOSUB, SRQ ON/OFF/STOP`

Functions

`IEEE(0)` through `IEEE(7)`, `STATUS`

To develop a GP-IB program, the operations of the measuring instrument to be connected must be understood.

For example, if a digital voltmeter is to be connected, the measurement modes such as DC voltage and AC voltage, and measurement range of voltage are selected by pressing appropriate buttons of the voltmeter. With a BASIC program, however, the control codes of the measuring instrument called program codes are transferred by the `PRINT@` command.

The measured voltage is received by the `INPUT@` command.

In this manner, a human operator could be replaced by a BASIC program.

A procedure to develop a GP-IB program is briefly explained on the following pages. For details, refer to the BASIC Unit Reference Manual (W207-E1).

Program Codes

Program codes are used to control each operation of the measuring instrument.

These program codes are different for each instrument. Refer to the instrument's manual for details.

The program codes are entered in the BASIC program in the sequence expected by the measuring instrument and executed by the measuring instrument.

Each program code is specified by a character code with the operation of each measuring instrument enclosed by literal ("). In addition, a program code can also be specified by a variable when, for example, an output frequency is continuously changed.

Example: Digital Multimeter 3478A

`PRINT@24; "F1RAN5T4"` . ..... `24` is the listener address, `F1` is the measuring function (DC voltage), `RA` is the auto range, `N5` is the 5 1/2 digit display, and `T4` is the trigger (hold).

Initializing GP-IB

Before transmitting data with the GP-IB, it is necessary to initialize the interface bus and measuring instruments.

1, 2, 3...

1. Initialize the GP-IB by making IFC (interface clear) of the interface control bus low:

   ```
   ISET IFC
   ```

2. Enable the GP-IB to be remotely controlled by making REN (remote enable) low.

   ```
   ISET REN
   ```

3. Transfer the DCL (device clear) command in the command mode to initialize the measuring instrument.

   ```
   WBYTE &H14;
   ```

4. This completes initialization. Some measuring instruments takes a long time to be initialized. Be sure to wait for the time specified by the instrument's manual before issuing the next command.

Transfer/Reception with GP-IB

To transfer commands or data to the GP-IB, the `PRINT@` or `WBYTE` statements are used. To receive data, the `INPUT@`, `LINE INPUT@`, or `RBYTE` instructions are used.

For example, suppose a 3478A digital multi-meter is connected.

Transfer the program code determined for each measuring device.

`PRINT@24; ″F1RAN5T4″` ..... `24` is the listener address, `F1` is the measuring function (DC voltage), `RA` is the range (auto range), `N5` is the 5 1/2 digit display, and `T4` is the trigger mode (hold).

Next, trigger the measuring instrument.

`PRINT@24;″T3″` .... `24` is the listener address, and `T3` is the trigger mode (Single trigger).

Receive and store the data in a variable.

`INPUT@24;I` .... `24` is the talker address and `I` is the name of data storage variable.

As you can see, data can be transferred to and from a GP-IB device with a simple BASIC program.

Service Request Interrupt Processing

If a service request is generated by a GP-IB measuring instrument, the interrupt service routine defined with `ON SRQ GOSUB` will be called.

`ON SRQ GOSUB *LABEL` ...... `*LABEL` is the label or first line number of the SRQ interrupt routine.

After the routine has been defined, the interrupt must be enabled when the program is ready to accept interrupts.

```
SRQ ON
```

To disable the interrupt, use `SRQ OFF`. To stop it temporarily, use the `SRQ STOP` statement.

When the SRQ interrupt service routine has been called, execute serial polling with the `POLL` statement.

The device status will be stored in a variable.

`POLL 24, S` .... `24` is the talker address, and `S` is the device status storage variable.

If bit 6 of the device status is 1, the SRQ interrupt was generated by the specified talker.

This statement can be used to check bit 6 of the status:

```
IF S AND 64 THEN *LABEL1
```

When the interrupting device has been found, read the data it is trying to send:

```
INPUT@24:R . ................ 24 is the talker address and R is the
                              data storage variable.
```

When the data has been received, the interrupt service routine can return (with the RETURN statement).

7-2-6 GP-IB Program Example

Preparations

The BASIC Unit controls GP-IB devices by sending “program codes” to them instead of pressing the panel buttons.

For example, to select the DC voltage mode as the measurement mode of Hewlett-Packard’s digital multi-meter 3478A, program code “F1” is sent instead of pressing the DC voltage mode switch.

Example 1

In this example, the DC voltage function is selected as the measurement function, the range is set to the auto range, display is set to 5 1/2 digit mode, and the measured current value is displayed on the terminal.

My-address of the digital multi-meter in this example is 24.

```
10 PARACT 0
20 '***********************************
30 '* GP-IB PROGRAM SAMPLE 1         *
40 '*   DMM = 3478A (ADDRESS ″24″)   *
50 '***********************************
60 ISET IFC . ............... Transfer interface clear command to ini-
                              tialize GP-IB interface
70 ISET REN . ............... Make REN (remote enable) line true
                              (low) to enable GP-IB to be remotely
                              controlled
80 WBYTE &H14; . ........... Send device-clear command (&H14) to
                              initialize 3478A
90 FOR J=0 TO 5000:NEXT J     Wait until 3478A has finished initializa-
                              tion
100 PRINT@ 24; ″F1RAN5T4″     Send program codes to set DC voltage
                              function (F1), auto-range (RA), 5 1/2 dig-
                              it display (N5), and trigger hold (T4)
110 PRINT@ 24; ″T3″ . ...... Send single trigger (T3) to 3478A
120 INPUT@ 24; I . ......... Receive data from 3478A
130 PRINT I . .............. Print data on terminal
140 END
150 END PARACT
```

Example 2

In this program, the two-line resistor function is selected as the measurement function, the range is set to auto range, the display is set in 5 1/2 digit mode, and the internal trigger is set. When the service request button on the front panel is pressed, the measured resistance is displayed on the terminal.
My-address of the digital multi-meter in this example is 24.

```
10 PARACT 0
20 '**********************************
30 '*GP-IB PROGRAM SAMPLE 2          *
40 '*   DMM = 3478A (ADDRESS ″24″)   *
50 '**********************************
60 ISET IFC . ............... Transfer interface clear command to ini-
                              tialize GP-IB interface
70 ISET REN . ............... Make REN (remote enable) line true
                              (low) to enable GP-IB to be remotely
                              controlled
80 WBYTE &H14; . ........... Send device-clear command (&H14) to
                              initialize 3478A
90 FOR J=0 TO 5000:NEXT J     Wait until 3478A has finished initializa-
                              tion
```

```
100 PRINT@ 24; ″F3RAN5T1″     Send program codes to set 2-line resistor function (F3), auto-range (RA), 5 1/2 digit display (N5), and internal trigger (T1)
110 PRINT@ 24; ″KM20″ . ....  Clear serial poll register (K) and set mask (M20) so that SRQ is generated only when SRQ button on front panel is pressed
120 ON SRQ GOSUB *SBOT . ...  Define SRQ interrupt service routine
130 SRQ ON . ................ Enable SRQ interrupts
140 PRINT ″SRQ KEY ON!″ . .   Display ″SRQ KEY ON!″ on terminal
150 *LOOP . ................. Wait here until SRQ key is pressed
160 GOTO *LOOP
170 ’--------------------------------
180 *SBOT . ................. SRQ interrupt service routine
190 POLL 24, S . ...........  Read device status
200 IF S<64 THEN 230 . .....  Make sure the device generated SRQ
210 INPUT@ 24;R . .........   Read data from the 3478A and display it on the terminal.
220 WBYTE $H5F;
230 PRINT ″R=″;R
240 RETURN
250 END
260 END PARACT
```

Note If the sampling cycle of the 3478A’s internal trigger is short, the previously held data may be transferred. If this occurs, it is necessary to clear the serial poll register. Change line 220 of the sample program as follows:

```
220 PRINT@24;″K″
```

Program Code Example for Digital Multi-meter 3478A

Type	Program code	Function
Measurement function	F1 to F7	DC voltage, AC voltage, 2-line resistor, 4-line resistor, DC current, AC current, expansion resistor
Range	R-2 R-1 R0 to R2 R3 to R7 RA	30 mVDC 300 mV, 300 mA AC/DC 3 V, 30 V, 300 V AC/DC or 3 A, 30 Ω, 300 Ω, 30 kΩ, 30 kΩ, 300 kΩ, 3 MΩ, 30 MΩ Auto range
Display	N3 to N5	3 1/2, 4 1/2, 5 1/2 display
Trigger	T1 to T5	Internal trigger, external trigger, single trigger, trigger hold, first trigger
Auto zero	Z0 Z1	Auto zero off Auto zero on
Write to display	D2 text D3 text D1	Displays message Displays message (display updating stopped) Normal display (D2, D3 display mode canceled)
Preset command	H0 H1 H2 to H7	Sets DC voltage, auto range, single trigger, 4 1/2 digit display, and auto zero ON. INPUT@ command disabled. Same, except INPUT@ command enabled. DC voltage, 2-line resistor, 4-line resistor, DC current, AC current, expansion current. Others same as H1
Binary status	B	Outputs status currently programmed by RBYTE command
Others	K E MXX S C	Clears serial poll register Reads error register Sets serial mask register (SRQ) Reads front rear switch Calibration

SECTION 8
Troubleshooting and Maintenance

This section provides the error messages and indications required for troubleshooting as well as general maintenance procedures for the BASIC Unit.

8-1 Troubleshooting

8-1-1 Error Messages

If an error occurs while a program is being entered or executed, an error message will be displayed on the terminal and the BASIC Unit will wait for operator input. At this time, an error code corresponding to the error message is set in ERR. This error code can be checked with the ERR function.

This table lists error messages, error codes, causes, and remedial actions. Some error messages do not have error codes. When these error messages are displayed, no error code is set in ERR.

Error message	Error code	Cause	Remedy
NEXT without FOR	1	FOR and NEXT are not correctly used in pairs	Use FOR and NEXT in pairs.
Syntax error	2	Instruction not used properly -RDIM is declared without global variable. -Number of arguments to FN function is different from the number declared. -System variable other than MID$, DATE$, and TIME$ is used on the left side of an assignment statement. -Incorrect line numbers are used in GOTO or GOSUB. -Multi-dimensional array variable was used in a PC READ or PC WRITE. -Number of arguments to a system function is wrong. -Character string is used for arithmetic operation other than addition.	Check the Reference Manual for the correct syntax and correct the program.
RETURN without GOSUB	3	GOSUB and RETURN are not correctly used in pairs.	This error occurs if a RETURN statement is encountered in a routine that was not called by GOSUB. Always call subroutines with GOSUB.
Out of DATA	4	No more data for a READ statement.	Match the number of data in DATA statements with the number used by READ. Check whether RESTORE is used correctly.
Illegal function call	5	A function was used incorrectly: an argument exceeds range permitted by the function, or the result exceeds the range of function. -Negative or 0 argument is specified for LOG function. -Negative argument is specified for SQR function. -Incorrect argument is used for MID$, LEFT$, RIGHT$, SPACE$, or INSTR. -Incorrect interrupt, signal, or key number is used for ON PC GOSUB, ON SIGNAL GOSUB, or ON KEY GOSUB. -Value of expression of ON GOSUB or ON GOTO is negative. -PC READ or PC WRITE variable and format types do not match. -FIELD, PUT, or GET was used on a non-random access file.	Check the Reference Manual to see how to use the function correctly.
Overflow (OV)	6	Operation result or numeric constant exceeds permitted range.	Check if data type and permitted range of values are correct.

Error message	Error code	Cause	Remedy
Out of memory	7	Memory capacity exceeded; program is too long. Memory capacity of compiler area, general-purpose memory area, S code area, E code area, and stack area exceeded.	Review program and remove unnecessary portions. Use PARACT to increase the work area for the task.
Undefined line number	8	The line number in a GOTO, GOSUB, or IF ... THEN ... ELSE does not exist. Non-existent line number specified in an EDIT command.	Check line number.
Subscript out of range	9	Value of subscript of array variable exceeds range defined by DIM and OPTION BASE.	Check range of subscript defined by array variable and range of array variable to be referenced.
Duplicate definition	10	An attempt was made to re-define an array or type. More than one OPTION BASE, OPTION LENGTH, or OPTION ERASE statement was found. More than one argument with the same name and type was defined by DEF FN.	Use a different array name. Define type once. Use OPTION BASE, OPTION LENGTH, or OPTION BASE only once. Check name of argument.
Division by Zero (/0)	11	The program attempted to divide by 0.	Do not divide by 0.
Illegal direct	12	Attempted to use a BASIC statement as an immediate mode command.	Use statements in programs.
Type mismatch	13	Types of variables do not match between right and left members of expression or in arguments of function.	Check data type.
Out of string space	14	Too much memory used by character strings.	Reduce character string length or character string array size.
String too long	15	One or more character strings is longer than 538 characters.	Make sure that no character string is longer than 538 bytes. (Long character strings may be split into several shorter strings). Increase the size of the work area allocated by PARACT.
Can't continue	17	When execution is stopped by the STOP statement or by CTRL+C or CTRL+X and the program is changed, execution cannot be resumed with CONT.	If you wish to use CONT, do not change the program while it is stopped. (There may also be places from which execution cannot be continued even if the program is not changed.)
Undefined user function	18	Undefined user function or machine language function is referenced.	Define functions with DEF FN or DEF USR before using them.
RESUME without error	20	RESUME was encountered outside of an error processing routine defined with ON ERROR GOTO.	Don't use RESUME outside of an error processing routine defined with ON ERROR GOTO.
FOR without NEXT	26	FOR and NEXT are not correctly used in pairs.	Use FOR and NEXT in pairs.
WHILE without WEND	29	WHILE and WEND are not correctly used in pairs.	Use WHILE and WEND in pairs.
WEND without WHILE	30	WHILE and WEND are not correctly used in pairs.	Use WHILE and WEND in pairs.
Duplicate label	31	The same label was defined more than once in the program.	Change labels so that each label is only defined once.
Undefined label	32	Undefined label was referenced.	Make sure that all labels referenced are defined.
Feature not available	33	The program attempted to use a non-existent device.	Make sure all necessary hardware exists.

Error message	Error code	Cause	Remedy
Routing error	37	The network specified by a PC READ or PC WRITE statement was not found in the CPU Unit's routing table.	Correct routing table or specify a different network.
READ or WRITE mismatch	38	PC READ/WRITE command without area specified and SEND(192) or RECV(193) instruction of CPU Unit are not correctly used.	Use PC READ and SEND(192), or PC WRITE and RECV(193), in pairs.
Not required from FINS	39	PC READ/WRITE was executed without area specified when SEND(192) and RECV(193) instruction of CPU Unit program is not executed.	Define a PC interrupt service routine with ON PC and use PC READ or PC WRITE in the routine.
FIELD overflow	50	FIELD length of more than 256 bytes was specified as the record length of a random file.	Field length must be less than 257 bytes.
Bad file number	52	The program attempted to use a file number outside the range 1 to 15.	Check the file number and reduce the number of files open simultaneously if necessary.
File not found	53	Specified file was not found while executing a file manipulation command such as LOAD, SAVE, KILL, or NAME. File name specified by LOC or LOF was not found. An attempt was made to open a non-existent file in APPEND or INPUT mode.	Specify correct file name.
File already open	54	OPEN, KILL, or NAME was executed on an open file.	Close the file before re-opening, deleting, or renaming it.
Input past end	55	There is no more data in the file.	Use functions such as EOF and LOF to detect end of file.
Bad file name	56	Incorrect file name was specified for a file manipulation command such as LOAD, SAVE, KILL, or OPEN.	Specify correct file name.
Direct statement in file	57	A direct statement (statement with no line number) was found when loading an ASCII program file.	Check file contents to see if line numbers have been damaged.
Sequential I/O only	59	I/O command other than sequential I/O command was used. Binary file was specified for MERGE.	Use sequential I/O command. Only ASCII file may be MERGEd.
File not open	60	The program attempted to use a file number which has not been opened in a command such as PRINT#, INPUT#, WRITE#, GET, or PUT.	Open file before executing I/O command.
File write protected	61	The program attempted to write to a write-protected file, or to write to a memory card whose write-protect switch is ON.	Turn off write protection.
Disk offline	62	The device specified in a file manipulation command such as LOAD, SAVE, KILL, or OPEN was not found.	Set the memory card correctly.
Disk I/O error	64	There was not enough space available in the Memory Card. An error occurred during memory card input or output.	Check to see if the Memory Card is correctly formatted and contains valid data
File already exists	65	File name specified by NAME already exists.	Specify a different file name or change the name of the existing file.
Disk full	68	There is no more room on the memory card for a SAVE, PRINT#, or PUT instruction.	Delete any unnecessary files or insert new memory card.
Bad drive number	70	Incorrect drive was specified in file name.	Drive number must be 0.

Error message	Error code	Cause	Remedy
Rename across disks	73	A file cannot be renamed from one drive to another.	Do not attempt to rename files from one drive to another.
Illegal operation	74	The program attempted to perform a file operation which is not allowed by the file's OPEN mode.	Check the mode used in the OPEN statement
RS232C board not ready	82	DSR is OFF.	Check connected device.
No Message queue	101	Message queue is missing or full. A task may have more than 4 message queues.	Reduce the number of messages used by the task.
Message queue not found	102	Message queue specified by SEND or RECEIVE was not found.	Use correct message number. Be sure to allocate the message number with MESSAGE before using it.
Message queue can't release	103	An error occurred during I/O. This is probably a result of internal stack manipulation.	Check program.
Cannot allocate message queue	111	No message queue is assigned.	Reduce number of message queues in use to eight or less.
Fatal Error	120	An error occurred during I/O. This is probably a result of internal stack manipulation.	Check program.
GPIB BIOS Error	121	An error occurred during GP-IB I/O.	Check GP-IB connections.
IEEE time out	128	Time out while processing time monitoring of GP-IB.	Check GP-IB connections and status of GP-IB devices.
IEEE interface clear	129	IFC was received during execution.	Correct the GPIB application so that IFC only goes ON once.
IEEE not controller	130	Command used by controller (master mode) is used.	Set controller (master mode).
IEEE not active device	131	Specified GP-IB device is not connected.	Check GP-IB devices and addresses.
I/O Timeout	200	Peripheral device is inoperable and monitor time (60 seconds) is exceeded.	Check device connections and status.
Illegal task number	201	Undefined or illegal task number was specified for TRON, TROFF, @ (current task switching), TASK, TWAIT, EXIT, or SENDSIG.	Make sure the task number is between 0 and 15, and that the task is defined in the program.
Illegal format	202	Incorrect characters were found in a PC READ or PC WRITE statement.	Check syntax of PC READ or PC WRITE.
Task already END	203	TWAIT was used to wait for a task that has already finished.	Check program.
Task already RUN	204	TASK was used to start a task that was already executing.	Check program.
Timer nothing	205	Timer cannot be acquired from system.	Check program.
Floating point exception	206	Valid range was exceeded in a floating-point operation.	Make sure that data does not exceed valid range.
FINS error response	207	Error occurred during execution of network instruction. Another error code is stored in ERR2 or ERR3. ERR2: Main response code ERR3: Sub-response code	Check devices on network. For details, refer to the descriptions of FINS.
Too many files OPEN	208	Too many files were opened simultaneously.	Check program.
Undefined Array	---	The program used an array that was not defined with DIM or RDIM.	Define arrays before they are used.
Illegal line number	---	A line number outside the range 0 to 65535 was referenced, or the program attempted to branch to a different task with GOTO or GOSUB.	Line numbers must be between 0 and 65535. Do not branch from one task to another.

Error message	Error code	Cause	Remedy
Verify Error	---	Contents of current program area do not coincide with contents of file specified for verification.	Check program area and file.
Program is protected	---	User program area is memory-protected.	Turn off memory protection with the DIP switch on front panel.
Undefined task0	---	The program has no task 0.	The program's main task must be task 0.
Too many variables	---	Space used by variables exceed memory capacity.	Reduce number of variables.
Compiler error	---	Error in system ROM.	Contact your OMRON representative.
Not enough memory	---	Out of memory during compilation in RUN status.	Increase program capacity, number of variables, or size of variable.
Switch is STOP	---	Attempt was made to RUN the program with RUN/STOP switch set to STOP.	Set switch to RUN.
System has fatal error	---	Error occurs during initialization. Program cannot be executed.	Refer to *8-1-2 Error Indication and Status*.
END PARACT without PARACT	---	END PARACT was encountered without a corresponding PARACT.	Check and correct program.
PARACT without END PARACT	---	END PARACT statement is missing.	Check and correct program.
Undefined line %u	---	Branch destination for GOTO or GOSUB was not found when RESUME was executed.	Check program.
Invalid ECODE	---	Execution code (ECODE) is wrong and program cannot be run.	ECODE is assumed to be missing during next RUN, and ECODE is created again and program is executed.

8-1-2 Error Indication and Status

Error List

Error	Problem	Correction
All indicators do not light	Power to PC is turned OFF.	Turn ON power to PC.
	BASIC Unit is not securely mounted.	Correctly mount BASIC Unit.
	Initialization between CPU Unit and BASIC Unit is not correctly performed. CPU Bus Unit error. BASIC Unit will not start at this time but CPU Unit can operate.	Clear cause of error and restart Unit by using the Restart Bit in Auxiliary Area word AR001 corresponding to Unit (turn the bit ON, and then OFF). If error persists, replace Unit.
Malfunctioning of connected devices	Power to connected devices is turned OFF.	Turn ON power to devices.
	Cable disconnected.	Connect cable and tighten screws.
	Break in cables, wrong wiring, or faulty connections.	Repair or replace cable.
	Baud rate and communication parameters do not match.	Check baud rate and communication parameters.
BAT LOW lights	Battery is not properly connected.	Check battery connections.
	Battery is discharged.	Replace battery.

Note The program area can be disturbed if a machine language program is run out of control. Re-initialize the program area using the following procedure if required.

- If the BASIC Unit operates with power turned on, but LIST or EDIT cannot be executed, input the following and then turn the power supply off and on:

```
          MON↵                      Enters machine language monitor
          *RDS0=40↵                 Sets segment register to 40 (address 400)
          *D0.3↵                    Displays the first four bytes
0000 - 04 12 90 19
*W0:0.0.0.0↵                        Overwrites the above four bytes.
```

- If the BASIC Unit does not operate at all, contact your OMRON representative.

The following errors may occur if the unit number is set incorrectly or if the memory switches cannot be read or written correctly. The BASIC program cannot be executed if any of these errors occur.

The error codes will be indicated as a binary value on the user indicators 0 through 7, with each indicating a binary digit between 2^0 and 2^7, i.e., indicator 0 turns ON to indicate a 1 in the 1's digit, indicator 1 turns ON to indicate a 1 in the 2's digit, indicator 2 turns ON to indicate a 1 in the 4's digit, etc.

Error code (on user indicators)	Problem	Correction
11	The same unit number has been set for two CPU Bus Units.	Check I/O table with CVSS and set I/O table correctly.
12	The unit number is already used for another Unit.	Check I/O table with CVSS and set I/O table correctly.
13	Unit is not registered in I/O table.	Update I/O table.
14	Unit number is not read correctly from CPU Unit.	Rotate unit number setting switch once and set correct unit number. If error still persists, Unit may be defective.
15 and 16	Cyclic interface operation error.	Turn the PC OFF and then ON. If error persists, BASIC Unit or CPU Unit may be defective.
07 through 09	Error occurs while reading or writing the CPU Unit's memory switches. BASIC Unit will operate with default memory switch values.	Turn the PC OFF and then ON. If error persists, Unit may be defective.

8-2 Maintenance

8-2-1 Replacing Units

- Before replacing the Unit, be sure to turn off the power.
- After replacing the Unit with a new one, check again to see if the old Unit is really defective.
- When sending a defective Unit to OMRON for repair, describe the symptoms of the error as clearly as possible.
- When the BASIC Unit malfunctions, the program in the internal RAM or EEPROM of the BASIC Unit cannot be read at all. It is therefore recommended that the program be saved to a memory card of the CPU Unit or to a floppy disk. For details, refer to *Section 4-4 Program Save and Load*.
- For quick recovery in case of trouble, always have at least one spare Unit available.

8-2-2 Battery Replacement

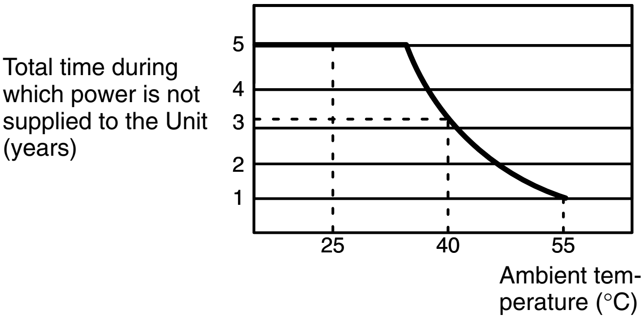
Battery Life and Replacement Period

The maximum life of the battery is 5 years, regardless of whether or not power is supplied to the Unit.

The battery life when power is not supplied to the Unit varies significantly with ambient temperature. The higher the temperature, the shorter the life of the battery.

The guaranteed and typical values for battery life when the power is not supplied to the Unit are shown below. The guaranteed value is based on memory backup at 55°C when the power is not supplied to the Unit. The typical value is based on memory backup at 25°C when the power is not supplied to the Unit

Effective life of battery		5 years
Memory backup battery life when power is not supplied	Guaranteed value	9,500 hours (Approx. 1 year)
	Typical value	43,000 hours (Approx. 5 years)

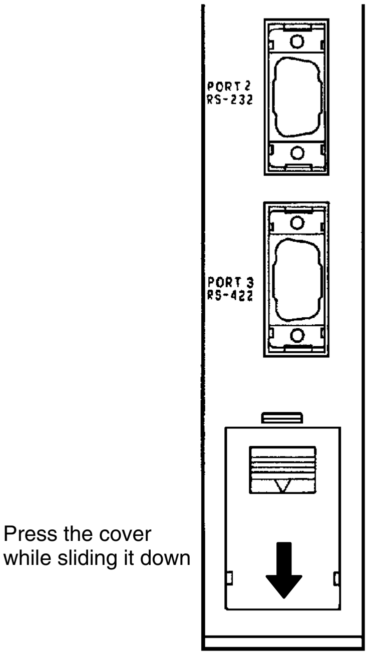


If the memory backup battery lifetime is exceeded, the BAT LOW indicator will light and the Battery Error Flag at bit 15 of word n+1 of the cyclic area input status will turn ON.

Replace the battery with a new one within 1 week after the BAT LOW indicator turns ON using the following replacement battery.

Name	Model no.
Battery Set	C500-BAT08

Battery Replacement Procedure



1, 2, 3... 1. Turn OFF the power to the Unit. If the power is already OFF, turn it ON for at least 1 minute and then turn OFF.

⚠ **Caution** It is possible to replace the battery with the power turned ON, but it is very dangerous because short-circuiting can easily occur.

2. While pressing the upper part of the battery compartment cover, slide it down and remove it.
3. Pull out the battery and connector and replace it with a new one. This procedure must be completed within 5 minutes.
4. Replace the battery compartment cover.

⚠ **DANGER** The battery may leak, catch fire, or explode if disposed of in fire. Do not short-circuit, charge, disassemble, heat, or incinerate the battery.

8-2-3 Inspection

Item of Inspection The main inspection items are as follows:

Item		Criteria	Check With
Ambient temperature	Is the temperature (in the control box) appropriate?	Must be 0° to 55°C	Thermometer
	Is the humidity (in the control box) appropriate?	Must be 10% to 90% with no condensation	Hygrometer
	Is the Unit clean?	Must be free from dust	Visual inspection
Mounting status	Are the cable connector screws tight?	Must not be loose	Screwdriver
	Is the cable okay?	Appearance must be normal	Visual inspection

Appendix A
Standard Models

BASIC Unit

Name	Specifications	Model number
BASIC Unit	Two RS-232C interfaces, RS-422 interface	CV500-BSC11
	Two RS-232C interfaces, RS-422 interface, EEPROM	CV500-BSC21
	Two RS-232C interfaces, Centronics interface	CV500-BSC31
	Two RS-232C interfaces, Centronics interface, EEPROM	CV500-BSC41
	RS-232C interface, GP-IB interface	CV500-BSC51
	RS-232C interface, GP-IB interface, EEPROM	CV500-BSC61

Option and Maintenance Parts

Name	Specification	Model number
Battery Set	Backup battery	3G2A9-BAT08
Connecting Cable	For connecting 14-pin and 36-pin connectors (printer cable) Cable length: 1.5 m	CV500-CN127

Appendix B
Specifications

Ratings

Conform to the SYSMAC CV-series Programmable Controllers.

Characteristics

Item	Specification			
CPU	μPD79011 (V25 + internal OS)			
Operating system	Real-time monitor (NEC)			
Program language	Interpreter-type multitasking BASIC and machine language (V25)			
Number of user tasks	16 (can be executed in parallel)			
Inter-task communication	Message transfer by SEND and RECEIVE instructions. Data sharing by global variable			
Inter-task synchronization	Notification of event occurrence by SENDSIG, ON SIGNAL GUSOB, and TWAIT instructions			
Task control method	Started by TASK instruction, and stopped by END, STOP, or EXIT instruction			
Debugging function	Tracing by TRON instruction, one-instruction execution by STEP instruction, pause and resumption by STOP, BREAK, and CONT instructions			
Memory	RAM	Source code area: 63K bytes		
		Variable area + executable code area: Approx. 110K bytes 32K bytes of variable area reserved for non-volatile variables.		
	EEPROM	To save source program: 63K bytes (BSC21, 41, and 61 only) The number of times the program can be written to the EEPROM is limited to 5,000. Do not exceed this limit.		
Interface with PC's CPU Unit	Cyclic	Total of 384 I/O words possible Default: 10 input words and 15 output words (via CPU Unit's I/O refresh)		
	CPU bus link	Default: No CPU bus link. To link CPU bus, CPU bus link must be set with the CVSS. Number of words read from CPU Unit: 128 max. Number of words read between CPU Bus Units: 8 max (The CPU Bus Link Area is refreshed by CPU Unit at 10-ms intervals.)		
	Event	When PC READ or PC WRITE instruction is executed: 512 bytes max. each for read and write When PRINT instruction is executed: 538 bytes max. each for read and write		
External interface	Interface	CV500-BSC11/BSC21	CV500-BSC31/BSC41	CV500-BSC51/BSC61
	RS-232C RS-422 Centronics GP-IB	2 ports 1 port --- ---	2 ports --- 1 port ---	1 port --- --- 1 port
Diagnosis function	BASIC Unit	Watchdog timer, low battery voltage detection		
	PC interface	Bus check, transfer/receive data horizontal parity check		
Battery life	5 years When the memory is backed up with no power applied, the life expectancy depends on the ambient temperature. When the BAT LOW indicator on the front panel of the Unit is lit, replace the old battery with a new one within 1 week.			
Current consumption	CV500-BSC11/BSC21/BSC51/BSC61: max. 0.5 A CV500-BSC31/BSC41: max. 0.3 A			
Dimensions	250 x 34.5 x 93 mm (HxWxD)			
Weight	550 gram max.			

I/O Interfaces

RS-232C (Port 1 or Port 2)

Item	Specification
Communication	Half duplex
Synchronization	Start-stop
Baud rate	300/600/1,200/2,400/4,800/9,600/19,200 bps
Transmission method	Point-to-point
Transmission distance	15 m max.
Interface	Conforms to EIA RS-232C

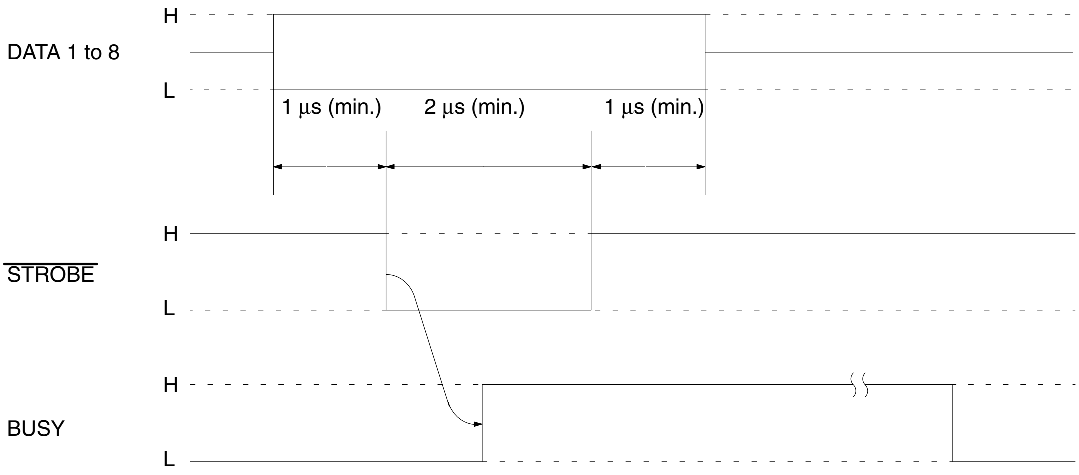
RS-422 (Port 3)

Item	Specification
Communication	Half duplex
Synchronization	Start-stop
Baud rate	300/600/1,200/2,400/4,800/9,600/19,200 bps
Transmission method (connection)	1:N Up to 32 Units can be connected to a PC. Termination resistance can be set by DIP switch.
Transmission distance	Total extension: 500 m max.
Interface (electrical characteristics)	Conforms to EIA RS-422 (Driver IC conforming to RS-485 is used.)
Termination resistance	220 Ω (built-in)

Centronics

Item	Specification	
Communication	Unidirectional communication	
Handshake	2-line handshaking with STROBE and BUSY lines	
Data transmission	8-bit parallel transmission	
Interface	TTL level	L level: Output ≤ 0.5 V, Input ≤ 0.8 V
		H level: Output ≥ 2.4 V, Input ≥ 2.0 V

Timing Chart



GP-IB

Item	Specification	
Communication	Half duplex	
Baud speed	Varies depending on device connected	
Handshake	Three-line handshaking	
Data transmission	8-bit parallel transmission	
Total cable length	20 m or number of devices connected to bus × 2 m, whichever is shorter	
Cable length between devices	4 m max.	
Number of devices connectable	15 max. including this Unit	
Interface	Conforms to IEEE Std 488-1978 (with 24-pin piggyback connector)	
Signal lines	Data lines: 8 (DIO1 through DIO8) Handshake lines: 3 (DAV, NRFD, NDAC) Control lines: 5 (ATN, REN, IFC, SRQ, EOI) Signal system ground: 8	
Signal logic	Negative logic	True: L level (max. 0.8 V)
		False: H level (min. 2.0 V)

GP-IB Interface

Operation	Symbol	Sub-function	Function
Source handshake	`SH`	`SH1`	`SH` all functions
Acceptor handshake	`AH`	`AH1`	`AH` all functions
Talker	`T`	`T6`	Basic talker Serial polling Talker cancellation by MLA
Expansive talker	`TE`	`TE0`	No `TE` function
Listener	`L`	`L4`	Basic listener Listener cancellation by MTA
Expansive listener	`LE`	`LE0`	No `LE` function
Service request	`SR`	`SR1`	`SR` all functions
Remote-local	`RL`	`RL1`	`RL` all functions
Parallel poll	`PP`	`PP1`	`PP` function by remote message
Device clear	`DC`	`DC1`	`DC` all functions
Device trigger	`DT`	`DT1`	`DT` all functions

Controller	`C`	`C1`
		`C2`
		`C3`
		`C4`
		`C26`
			System controller function
			Transmission of IFC
			Controller in charge
			Transmission of REN
			Transmission of message of interface responding to SRQ.
			Execution of parallel polling

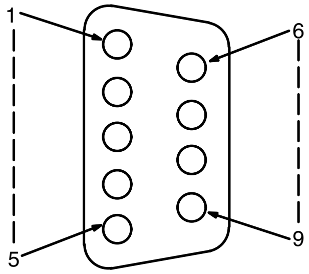
The BASIC Unit can be set in two modes: Master Mode and Slave Mode. In the Master Mode, the Unit always serves as the system controller. In the Slave Mode, the Unit only serves as the talker or listener.			

Appendix C
Hardware Interfaces

RS-232C Interfaces

Pin Configuration

Port 1 and Port 2 are RS-232C interfaces and are configured as follows:



Pin No.	Signal symbol	Signal name	Signal flow
1	FG	Frame ground	---
2	SD (TXD)	Send data	Output
3	RD (RXD)	Receive data	Input
4	RS (RTS)	Request to send	Output
5	CS (CTS)	Clear to send	Input
6	---	Unused	---
7	DR (DSR)	Data set ready	Input
8	ER (DTR)	Data terminal ready	Output
9	SG	Signal ground	---
Connector washer	FG	Frame ground	---

Applicable Connector

Plug: XM2A-0901 (OMRON)

Hood: XM2S-0911 (OMRON) or equivalent

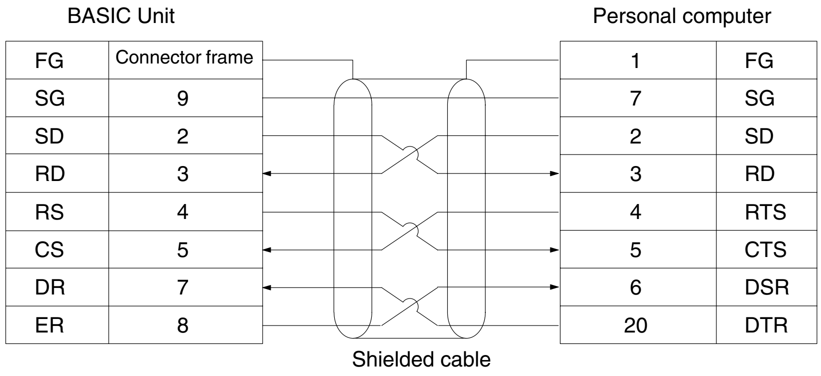
One plug and one hood are supplied for each port.
Connectors other than those on the left cannot be used.

Recommended Cables

AWG28 x 5P IFVV-SB (Fujikura Densen)
CO-MA-VV-SB 5P x AWG28 (Hitachi Densen)
Cable length: 15 m max.

Connection Examples

Personal Computers

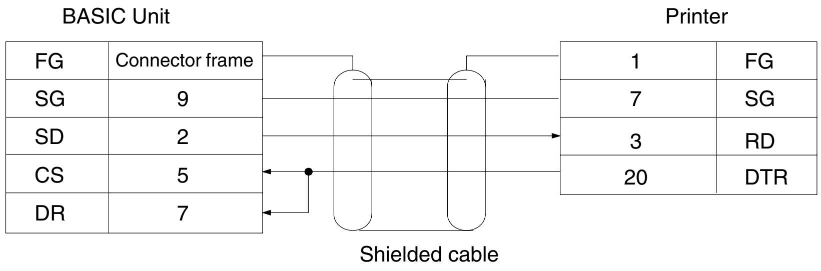


FG	Connector frame
SG	9
SD	2
RD	3
RS	4
CS	5
DR	7
ER	8

1	FG
7	SG
2	SD
3	RD
4	RTS
5	CTS
6	DSR
20	DTR

Numbers indicate pin numbers.

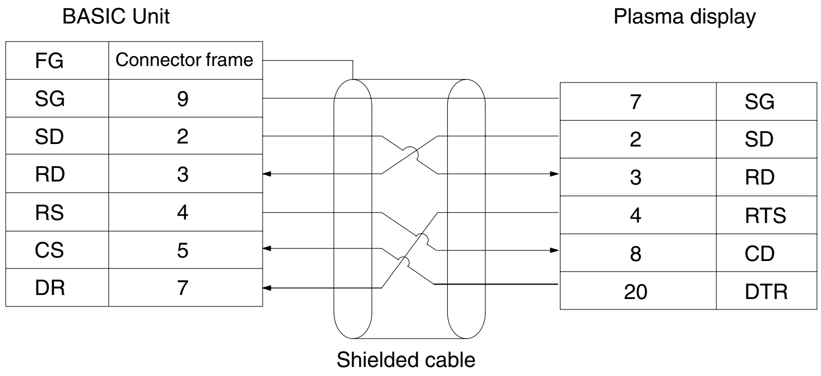
Printers



FG	Connector frame
SG	9
SD	2
CS	5
DR	7

1	FG
7	SG
3	RD
20	DTR

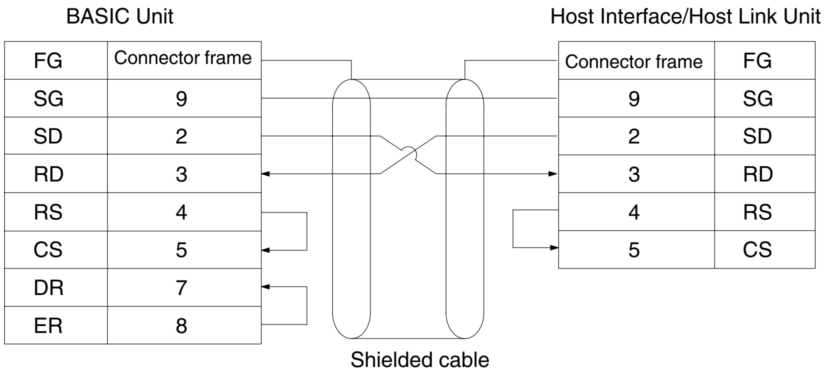
Plasma Displays



FG	Connector frame
SG	9
SD	2
RD	3
RS	4
CS	5
DR	7

7	SG
2	SD
3	RD
4	RTS
8	CD
20	DTR

CPU Unit Host Interface/Host Link Unit



FG	Connector frame
SG	9
SD	2
RD	3
RS	4
CS	5
DR	7
ER	8

Connector frame	FG
9	SG
2	SD
3	RD
4	RS
5	CS

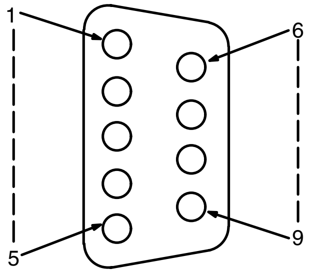
Note 1. If the cable is connected or disconnected while the power is being supplied to the BASIC Unit and peripheral device, the BASIC Unit may malfunction. Be sure to turn OFF the power before connecting the cable.

2. The above connection examples do not necessarily apply to all devices. Be sure to consult the manual for the peripheral device you are connecting.

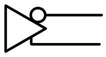
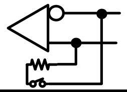
RS-422 Interface

Pin Configuration

Port 3 is an RS-422 interface and is configured as follows:



Signal name	Abbreviation	Pin No.	Signal flow
Send data	SD– (SDA)	9	Output
	SD+ (SDB)	5	
Receive data	RD– (RDA)	6	Input
	RD+ (RDB)	1	
Frame ground	FG	7	---
Frame ground	FG	Connector fixture	---

Send data	Receive data
	

Connector

Plug: XM2A-0901 (OMRON)

Hood: XM2S-0911 (OMRON) or equivalent

One plug and one hood are supplied for port 3.

Recommended Cables

AWG28 x 5P IFVV-SB (Fujikura Densen)
CO-MA-VV-SB 5P x AWG28 (Hitachi Densen)
Cable length: 500 m max.

Note 1. Connect only one side of the shield cable to FG so that no current flows through the shield. To connect the shield to FG, connect it to pin 7 of the connector or to the hood.

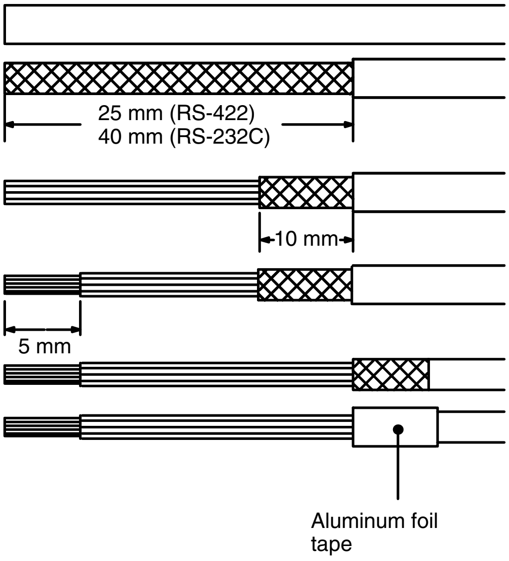
2. Turn ON the termination resistance (220 Ω, built-in) of the BASIC Units at both ends of the RS-422 communication line or Link Adapter. Turn OFF the termination resistance of the other Units. If the termination resistance is not set correctly, communications will not be possible.

3. Ground the FG terminal of the CPU Unit to less than 100 Ω.

Wiring the Connector

Connect and solder the cable according to the following procedure. Keep the cable length to within the length shown in the following figures.

Preparation when Connecting the Shield to FG



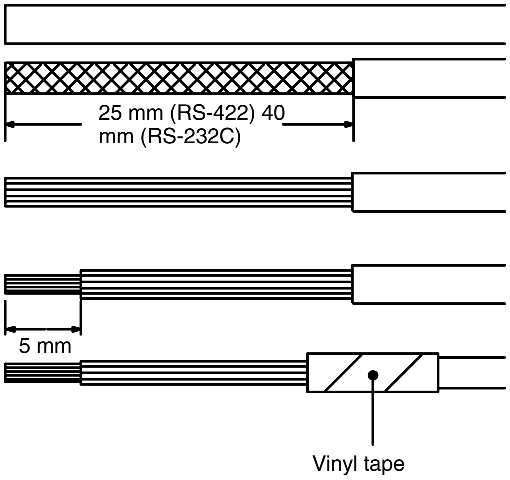
Cut the cable to the required length. Remove the sheath with a razor. Take care not to damage the shield (mesh).

Cut the shield with scissors.

Expose the core with a stripper.

Turn the shield over the cable and wind the cable with aluminum-foil tape.

Preparations When Not Connecting the Shield to FG



Cut the cable to the necessary length.
Remove the sheath with a razor.

Remove the exposed shield with scissors.

Expose the core with a stripper.

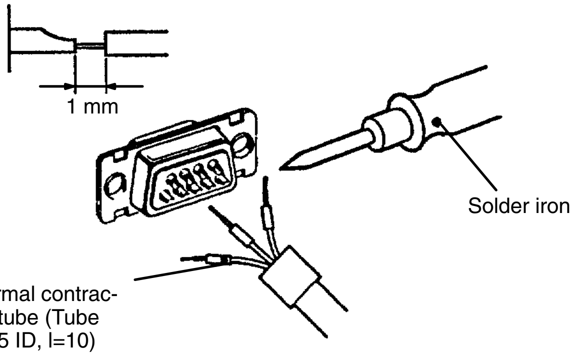
Wind the cut portion of the shield with vinyl tape.

Soldering

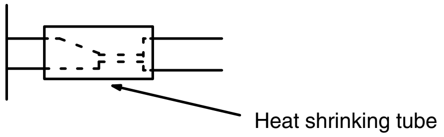
1, 2, 3...

1. Pass each line through a heat-shrinking tube.
2. Apply preliminary solder to each line and connector pin.

3. Solder each line.

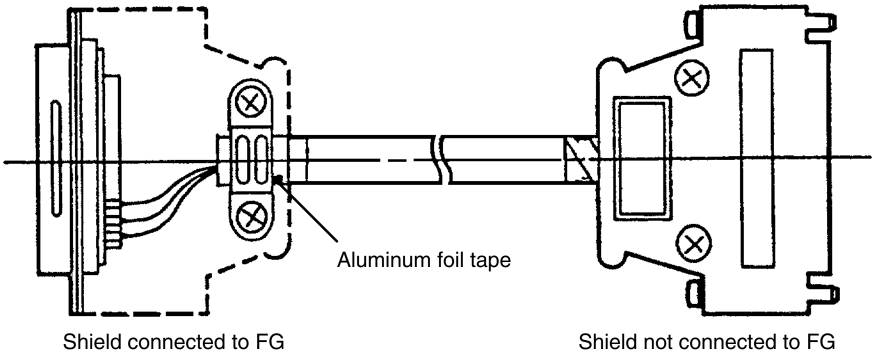


4. Slide the heat-shrinking tube over the soldered portion and heat the tube to shrink it into place.



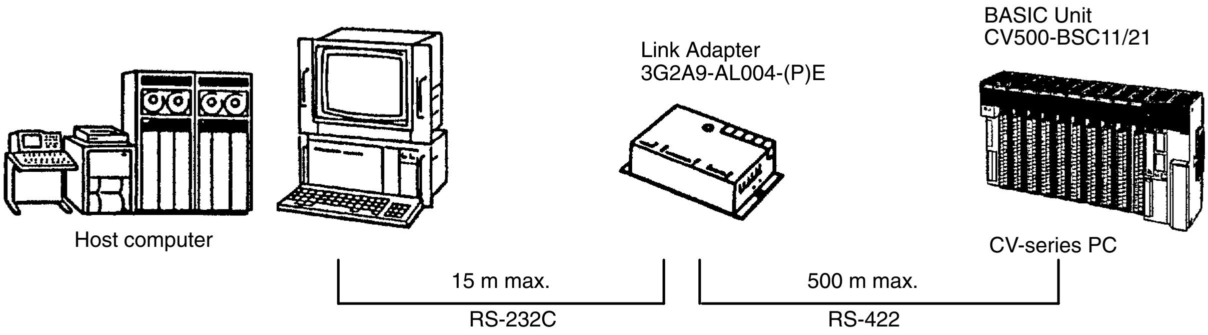
Hood Assembly

Assemble the connector hood as follows.

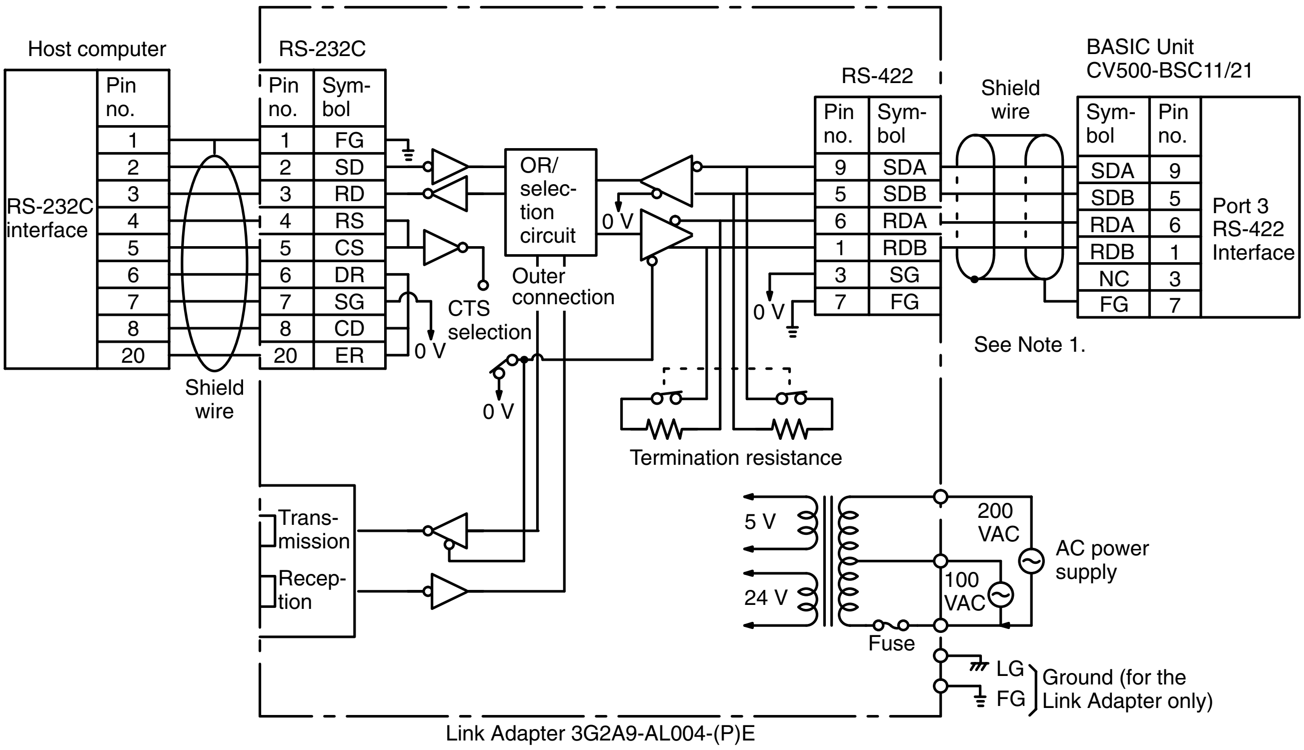


Point-to-point Connection

This section describes how to connect one BASIC Unit to one host computer.



Connection Example



Turn ON the internal termination resistance (220 Ω) by using the DIP switch (pin 4) on the front panel.

Setting Link Adapters

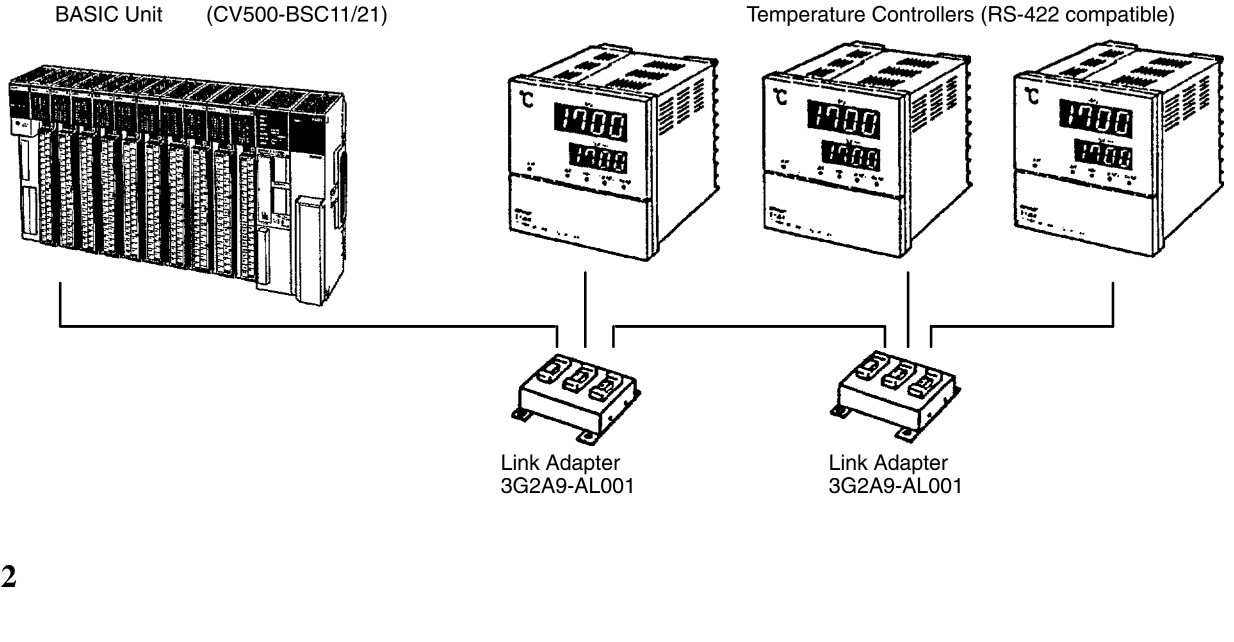
Turn ON the internal termination resistance (220 Ω). To keep ON the CTS (clear to send) signal, set the Link Adapter to 0 V. To receive the CTS signal from an external source, set to external. The Link Adapter is usually set to 0 V.

Note 1. Connect only one end of the shield to FG so that no current flows through the shield. To connect the BASIC Unit with a Link Adapter, connect the shield of the BASIC Unit to FG. To connect the shield to FG, connect it to pin 7 of the connector or to the connector hood.

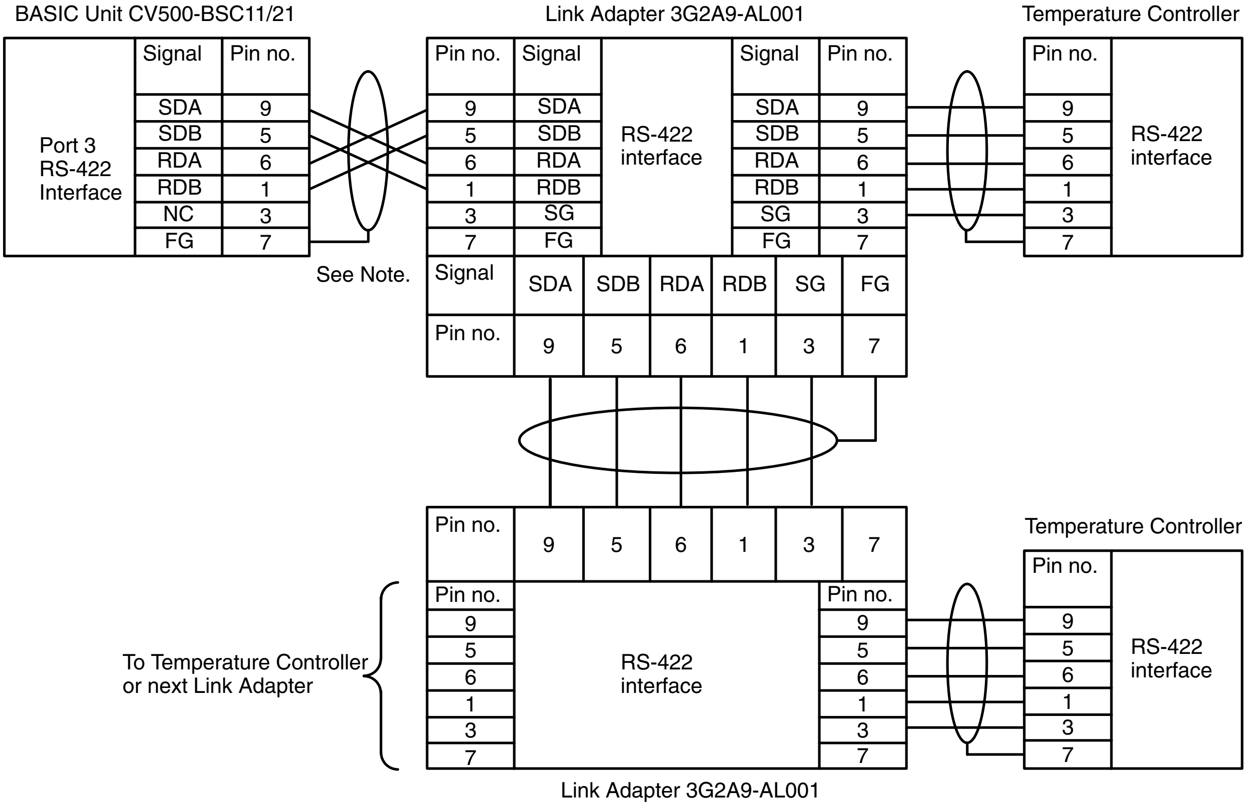
2. Be sure to cap all unused optical connectors. Errors will occur due to external light disturbances if unused connectors are left open.

Multidrop Connection, Example 1

In a multidrop connection, more than one RS-422 device can be connected to one BASIC Unit.


BASIC Unit (CV500-BSC11/21)
Temperature Controllers (RS-422 compatible)
Link Adapter 3G2A9-AL001
Link Adapter 3G2A9-AL001

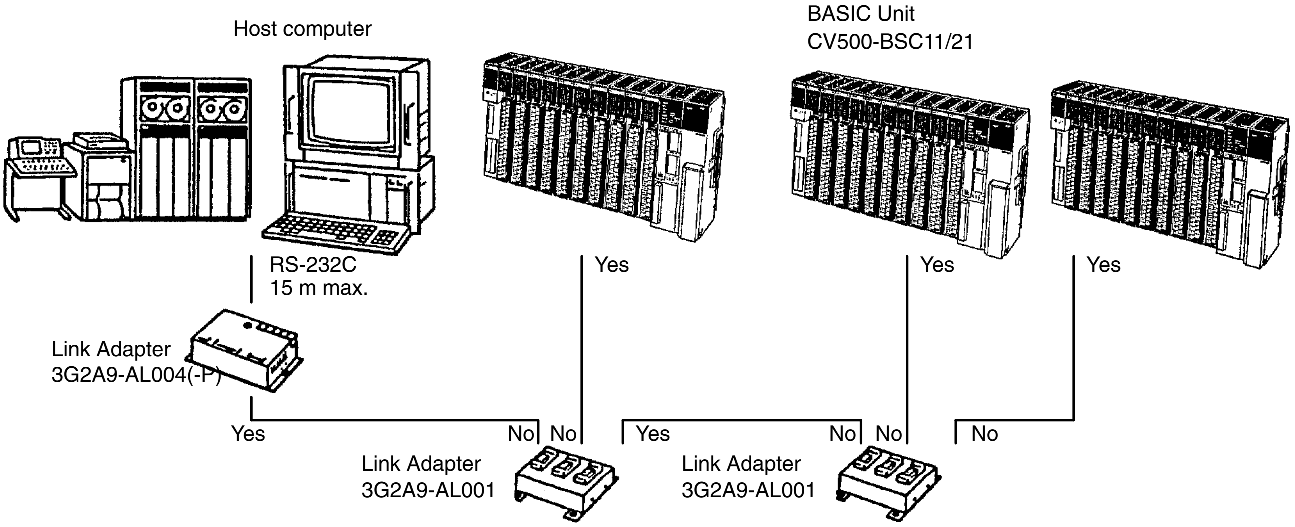
Connection Example



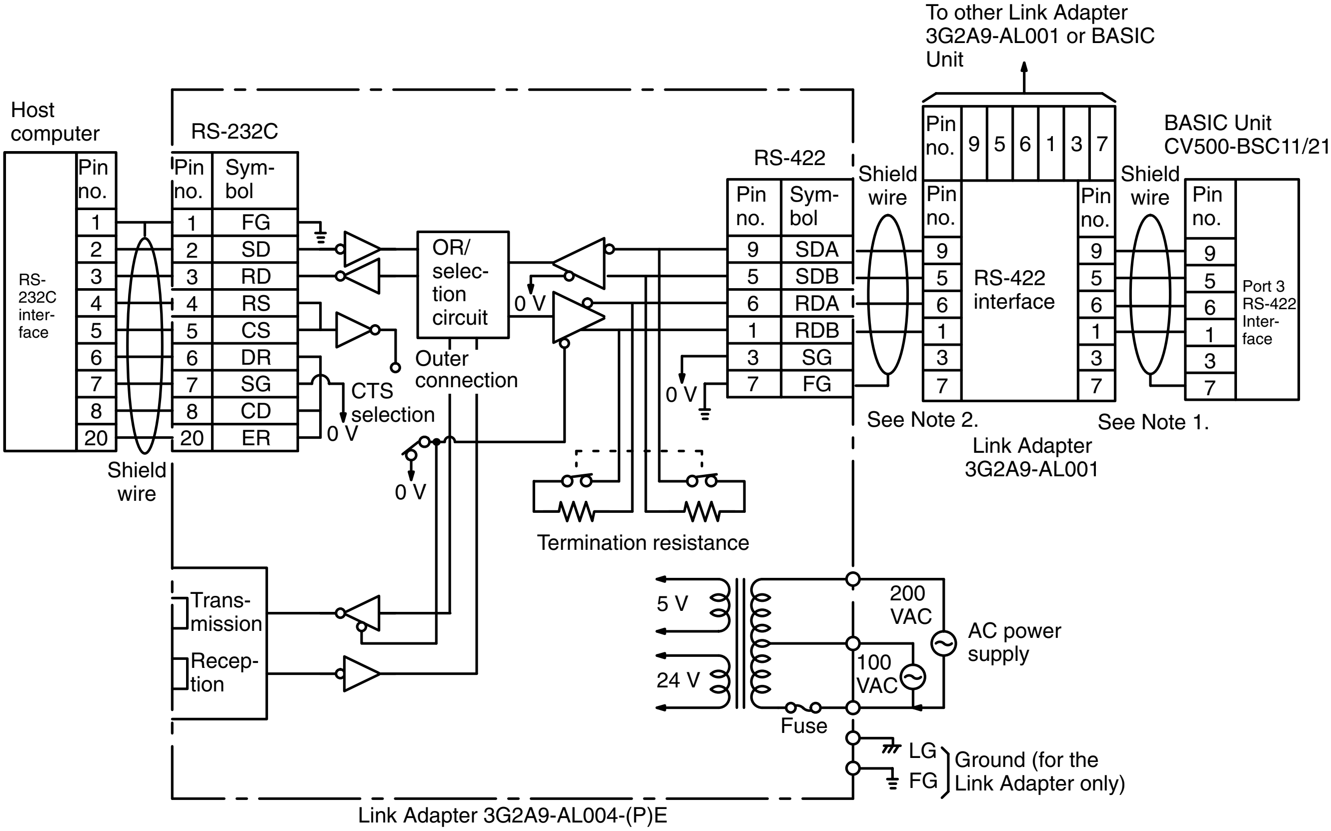
Note Connect the shield from the BASIC Unit to FG at the Link Adapter only.

Multidrop Connection, Example 2

In multidrop connection, more than one BASIC Unit can be connected to one host computer. In the following diagram, “Yes” means that the shield is connected to FG (frame ground) of the Unit, and “No” means that the shield is not connected to FG.



Connection Example



Note 1. Connect the shield from the BASIC Unit to FG at the Link Adapter only.

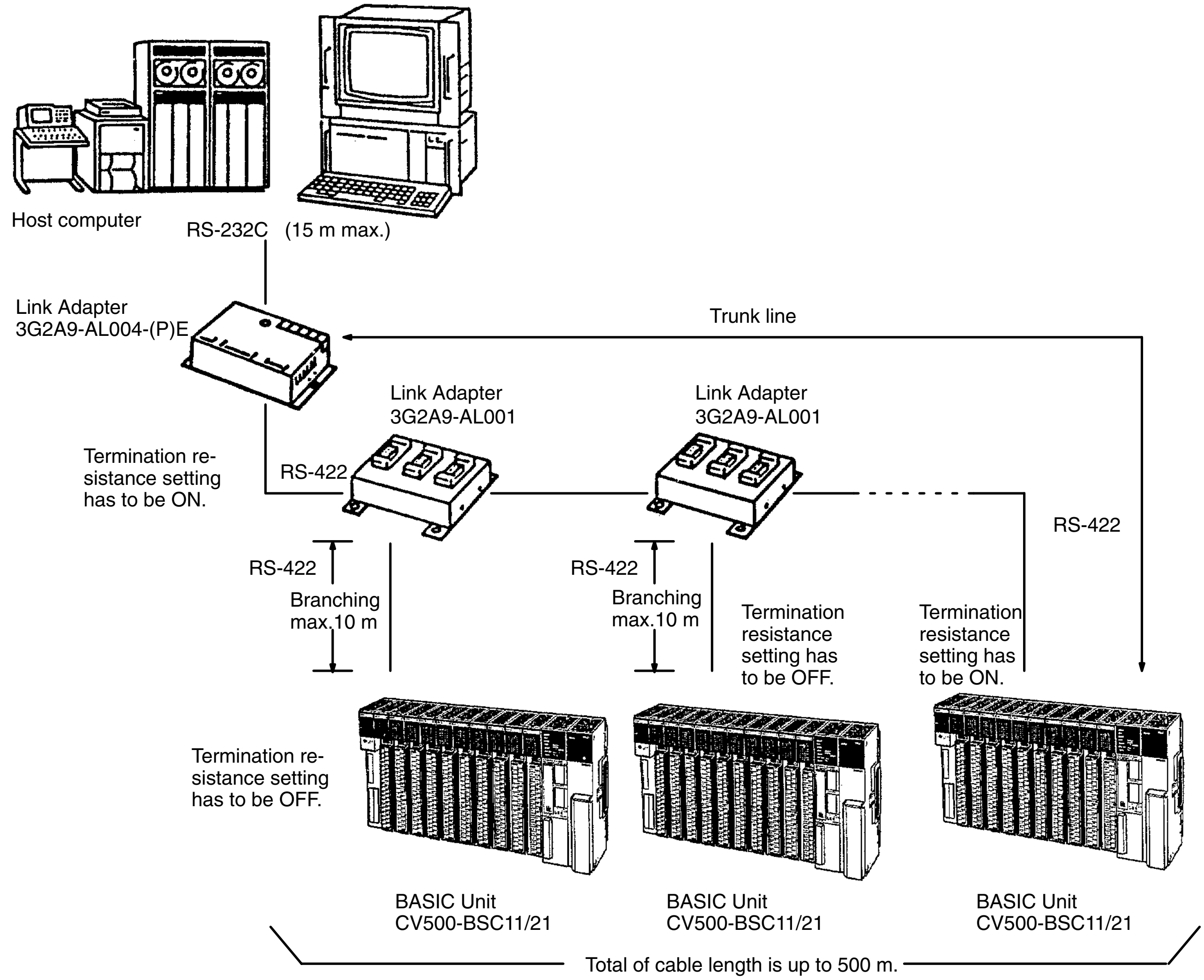
2. Connect the shield to FG at only one Link Adapter for lines connecting two Link Adapters.

Cable Length and Termination Resistance in Multidrop Configurations

Use shielded twisted pair cables. Route the cables keeping them separate from other signal lines. Keep the total cable length, including branch lines, to within 500 m. Keep the branch lines to within 10 m.

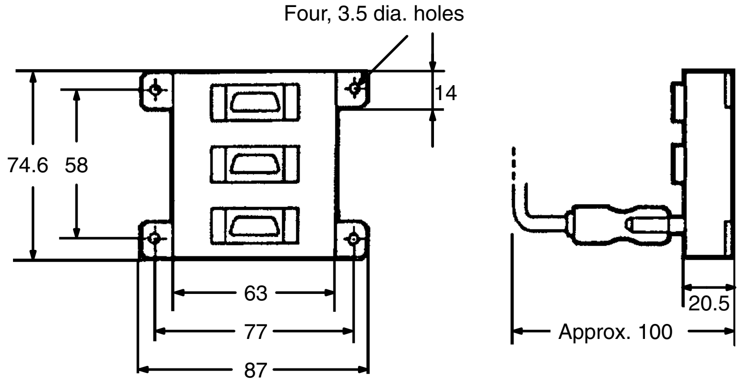
Turn on the termination resistance of the BASIC Units at both ends of the trunk line and that of the Link Adapters. Turn OFF the termination resistance of the other BASIC Units to OFF. Communications will not be possible if termination resistance is not set correctly.

Wire the system so that the branch lines extend from the trunk line.

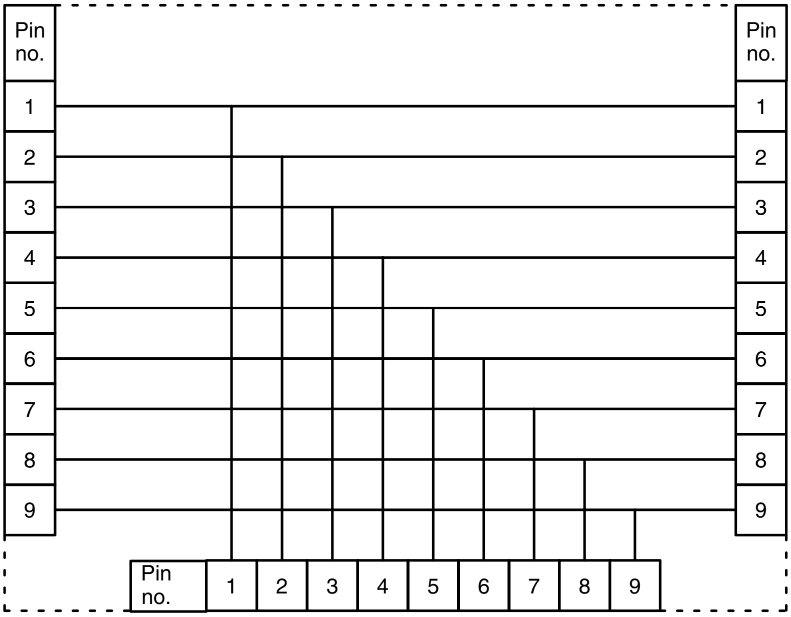


3G2A9-AL001 Link Adapter Specifications

Dimensions



Signals



RS-422 Link Adapter 3G2A9-AL001

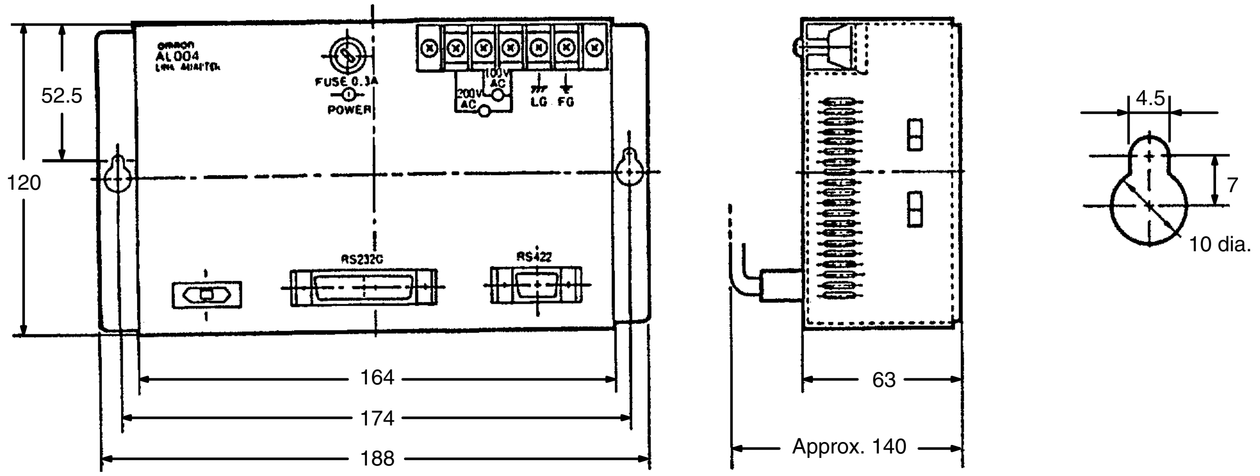
Applicable Connector

Connector:	XM2A-0901
Connector cover:	XM2S-0901

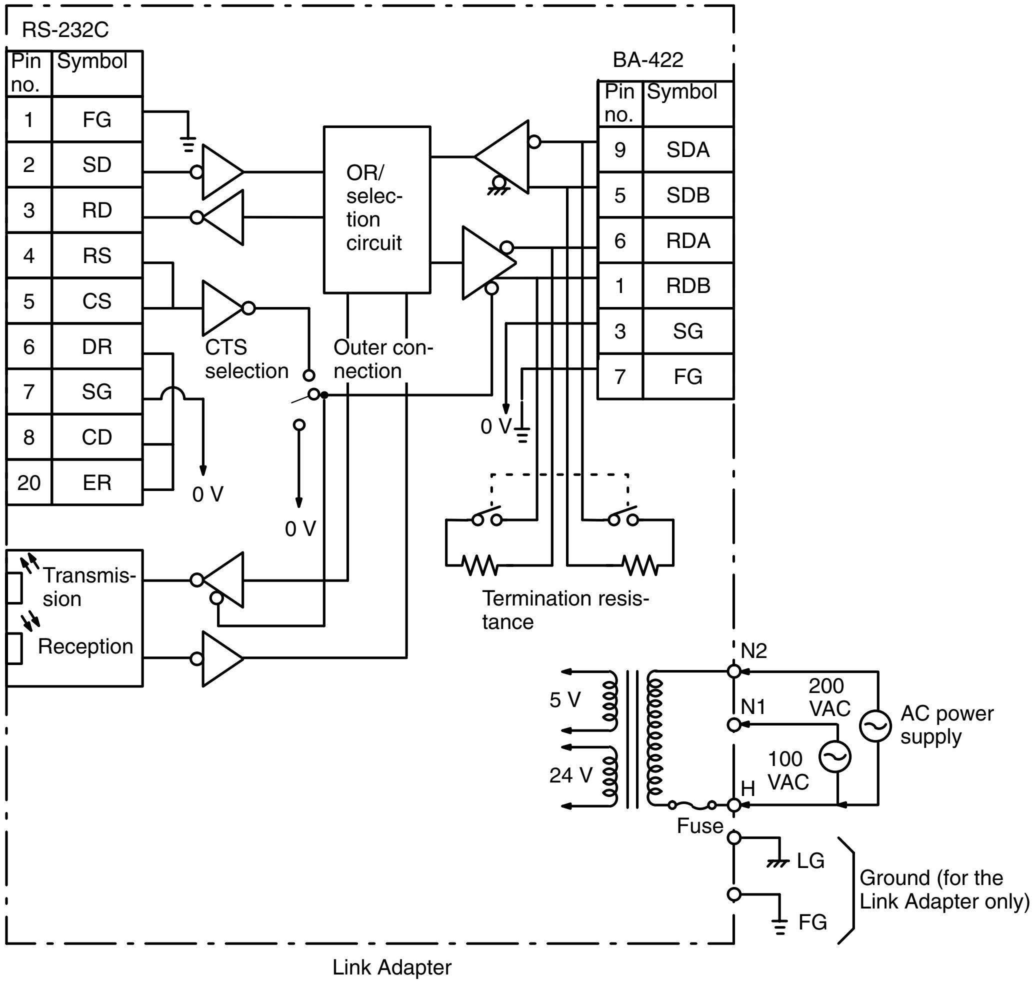
Three RS-422 connectors are supplied with 3G2A9-AL001.

3G2A9-AL004-(P)E Link Adapter Specifications

Dimensions



Internal Configuration



RS-232C

Pin no.	Symbol
1	FG
2	SD
3	RD
4	RS
5	CS
6	DR
7	SG
8	CD
20	ER

BA-422

Pin no.	Symbol
9	SDA
5	SDB
6	RDA
1	RDB
3	SG
7	FG

Link Adapter
3G2A9-AL004-(P)E

Cable Lengths (max.)

Cable unit	3G2A9-AL004-PE	3G2A9-AL004-PE
APF (All Plastic optical Fiber)	20 m	Not connectable
PCF (Plastic Clad optical Fiber)	200 m	800 m

Note Be sure to cap all unused optical connectors.

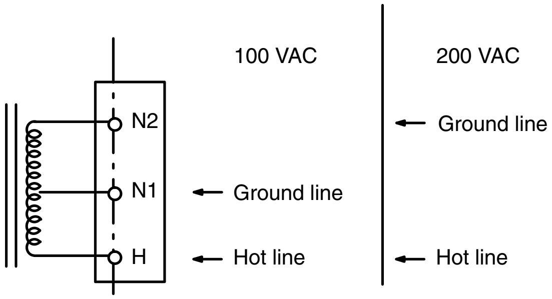
Selecting CTS (CS)

To keep ON the CTS (clear to send) signal, set to 0 V. To receive the signal from an external source, set to external.

Setting Termination Resistance

To connect the internal termination resistance (220 Ω), set the selector switch to ON. If the resistor is not to be connected, set the switch to the OFF position.

Power Supply



A fuse is provided at the common. Connect the AC hot line to the common terminal side when connecting the power supply.

Installing Link Adapters

To avoid electric shock, do not touch the terminal block when installing the Link Adapter in an office or on a desk.

Note 1. Do not use the Link Adapter with the terminal cover removed.

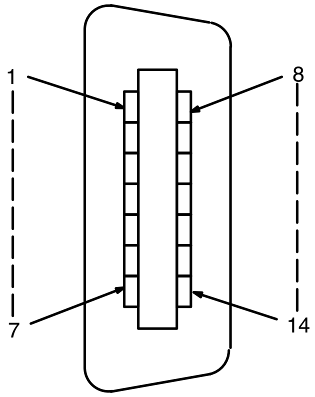
2. Securely mount the terminal block cover.

Centronics Interface

Communication Specifications

Conforms to Centronics specifications

Pin Configuration



Pin no.	Abbreviation	Name	Signal flow
1	STROB	Strobe	Output
2	DATA 1	Send data	Output
3	DATA 2	Send data	Output
4	DATA 3	Send data	Output
5	DATA 4	Send data	Output
6	DATA 5	Send data	Output
7	DATA 6	Send data	Output
8	DATA 7	Send data	Output
9	DATA 8	Send data	Output
10	NC	Not used	---
11	BUSY	Busy	Input
12	NC	Not used	---
13	NC	Not used	---
14	GND (0 V)	Ground	---

Applicable Connector

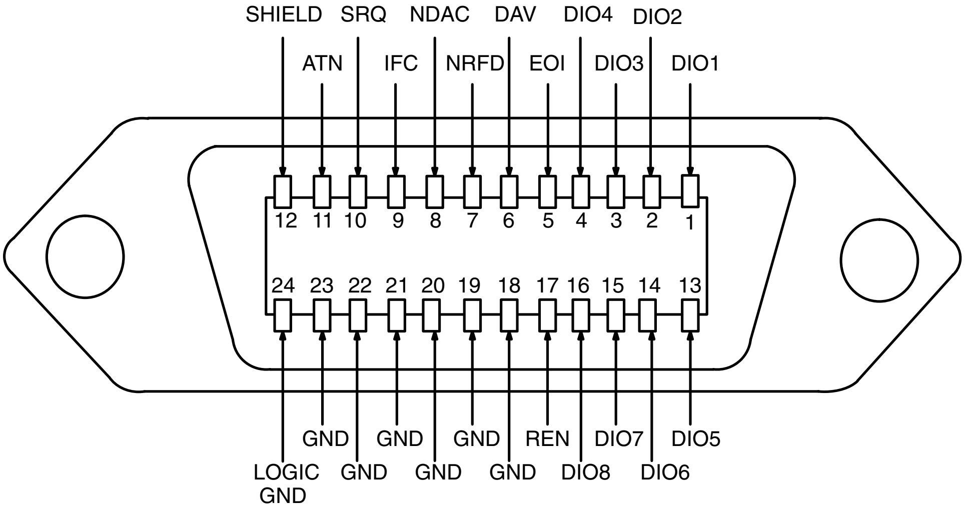
Connector:	57-30140 (DDK)
Cable:	CV500-CN127 (optional, cable length: 1.5 m, 14P-36P)

The cable supplied with the printer can also be used.

Note If the cable is connected or disconnected while power is being supplied to the BASIC Unit and the Centronics device, the BASIC Unit may malfunction. Be sure to turn OFF the power before connecting or disconnecting the cable.

GP-IB Interface

Pin Configuration



Signal Lines

Line		Bus	
Data bus	DIO 1 (Data Input/Output 1)	Transmit data Example: Address, Command, Measured data, Program data, Display data, Status	
	DIO 2 (Data Input/Output 2)		
	DIO 3 (Data Input/Output 3)		
	DIO 4 (Data Input/Output 4)		
	DIO 5 (Data Input/Output 5)		
	DIO 6 (Data Input/Output 6)		
	DIO 7 (Data Input/Output 7)		
	DIO 8 (Data Input/Output 8)		
Transfer bus	DAV (Data Valid)	Signal indicating validity of data	Perform acceptor and handshaking
	NRFD (Not Ready For Data)	Reception ready signal	
	NDAC (Not Data Accepted)	Reception completion signal	
Control bus	ATN (Attention)	Signal indicating that data on data bus is address or command	
	IFC (Interface Clear)	Signal initializing interface	
	SRQ (Service Request)	Signal requesting service	
	REN (Remote Enable)	Remote/local specifying signal	
	EOI (End Of Identify)	Indicates last byte of data, or indicates execution of parallel polling	

Note If the cable is connected or disconnected while power is being supplied to the BASIC Unit and the GP-IB device, the BASIC Unit may malfunction. Be sure to turn OFF the power before connecting or disconnecting the cable.

Recommended Cables

Maker	Model
DDK	408JE-10P5 (50 cm) 408JE-101 (1 m) 408JE-102 (2 m) 408JE-104 (4 m)
Honda Tsushin Kogyo	ADS-GP24-050 (50 cm) ADS-GP24-100 (1 m) ADS-GP24-200 (2 m) ADS-GP24-300 (3 m) ADS-GP24-400 (4 m)

Note Turn off the power to both the GP-IB and BASIC Unit before connecting or disconnecting the GP-IB and BASIC Unit. Otherwise, the BASIC Unit may malfunction.

Appendix D
Program Examples and Reserved Words

Single Task Program

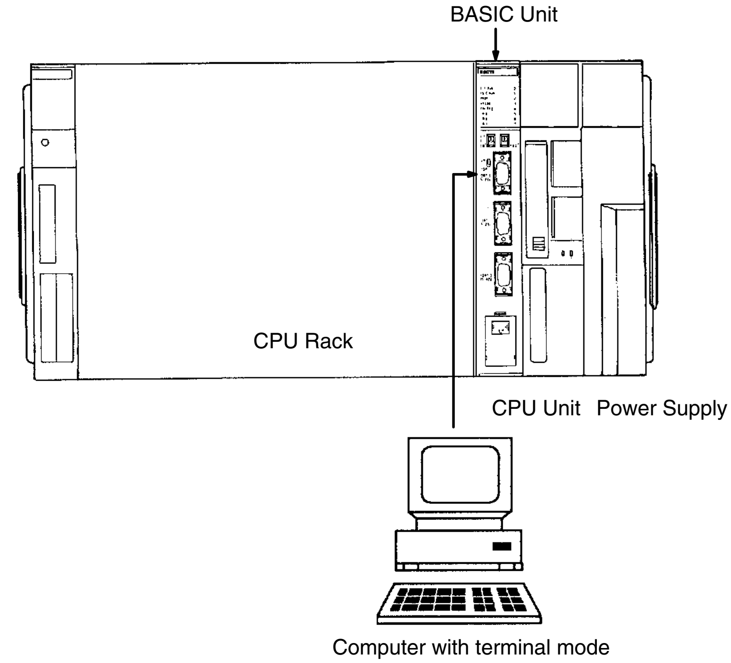
Operation

Calculates and displays the square root of an input numeric value.

If no data is input for 10 seconds, an error occurs, an error message is displayed. At this stage, the BASIC Unit waits for input.

The program is terminated when `E` is input.

Configuration



Example Program

```
10 PARACT 0
15 ON ERROR GOTO *INERROR
20 CLS:LOCATE 20,10
30 INPUT WAIT 100, ”Input numeric value whose square root is to be calculated.
(End: E)”,V$ . .... Input numeric value. If no input is made for 10 seconds, error message is displayed
40 IF V$=”E” OR V$=”e” THEN END . .................... Terminates when E or e is input
50 GOSUB *DSPLY
55 GOTO 20
60 *INERROR GOSUB *MESS
70 RESUME 20
80 END
90 ’
100 *MESS . ............................................ Message output subroutine
110 CLS
120 FOR K=0 TO 19
130 LOCATE 20, K:PRINT ”OMRON’s PC is best!”
140 FOR J= 0 TO 200:NEXT J
150 NEXT K
160 LOCATE 20,20:WRITE ”BASIC UNIT is also good.”
170 FOR J=0 TO 1000:NEXT J
180 RETURN
```

```
190 ’
200 *DSPLY ............................................ Calculation result display subroutine
210 OV%=VAL(V$) ...................................... Converts character into integer
220 ANS#=SQR(OV%) .................................... Calculates square root. Result is
                                                       double-precision integer
230 CLS:LOCATE 20,20
240 PRINT ”Square root of ”;V$;” is ”;ANS#;”.” . ... Result is displayed
250 FOR K=0 TO 10000:NEXT K
260 RETURN
270 END PARACT
```

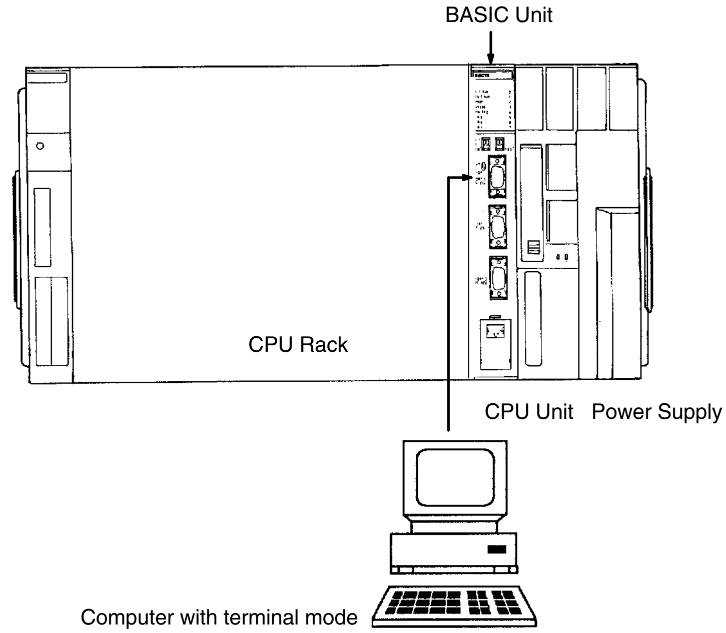
Multitask Program

Operation

Task 0 creates data in a random-access file and sends a message to task 1. Task 1 then waits for a message from task 0. When the message is received from task 0, data from the random file is read and displayed.

Task 0 waits until task 1 is terminated.

Configuration



Example Program

```
10 ’****************
20 ’*MULTI TASK (0)*
30 ’*              *
40 ’****************
50 DIM BUF1$10, BUF2$10, BUF3$10, BUF4$10
60 DIM MESS$30
70 PARACT 0
80 TASK 1
90 OPEN ”testfile” AS #1
100 FIELD #1, 10 AS BUF1$, 10 AS BUF2$
110 LSET BUF1$=”OMRONCV500”
120 LSET BUF2$=”BASIC UNIT”
130 PUT #1,1
140 CLOSE #1
150 MESSAGE 0,10
160 MESS$=”DATA is written to testfile”
170 SEND 10, MESS$
180 TWAIT 1
```

```
190 END
200 END PARACT
210 ’****************
220 ’*MULTI TASK (1)*
230 ’*              *
240 ’****************
250 PARACT 1
260 ’Data is received from common memory and message is transferred
270 MESSAGE 0,10
280 RECEIVE 10, MESS$
290 MESSAGE 1,10
300 PRINT ”Slave task”
310 PRINT ”Message is received from master task.”
320 PRINT ”Message is as follows: ”;MESS$
330 OPEN ”testfile” AS #1
340 FIELD #1,10 AS BUF3$,10 AS BUF4$
350 GET #1,1
360 CLOSE #1
370 KILL ”testfile”
380 PRINT ”Do you want to see this data?”
390 INPUT ”Press Y key, if YES: ”;A$
400 IF A$=”y” OR A$=”Y” THEN GOTO *SEE ELSE GOTO *E
410 *SEE
420 PRINT BUF3$
430 PRINT BUF4$
440 PRINT ”That is all for data.”
450 *E
460 END
470 END PARACT
```

Input/Output of Each Port

Operation

RS-232C Interface

Receives RS-232C data by means of an interrupt, and decides whether reception, transmission, or termination is to be performed according to the input data.

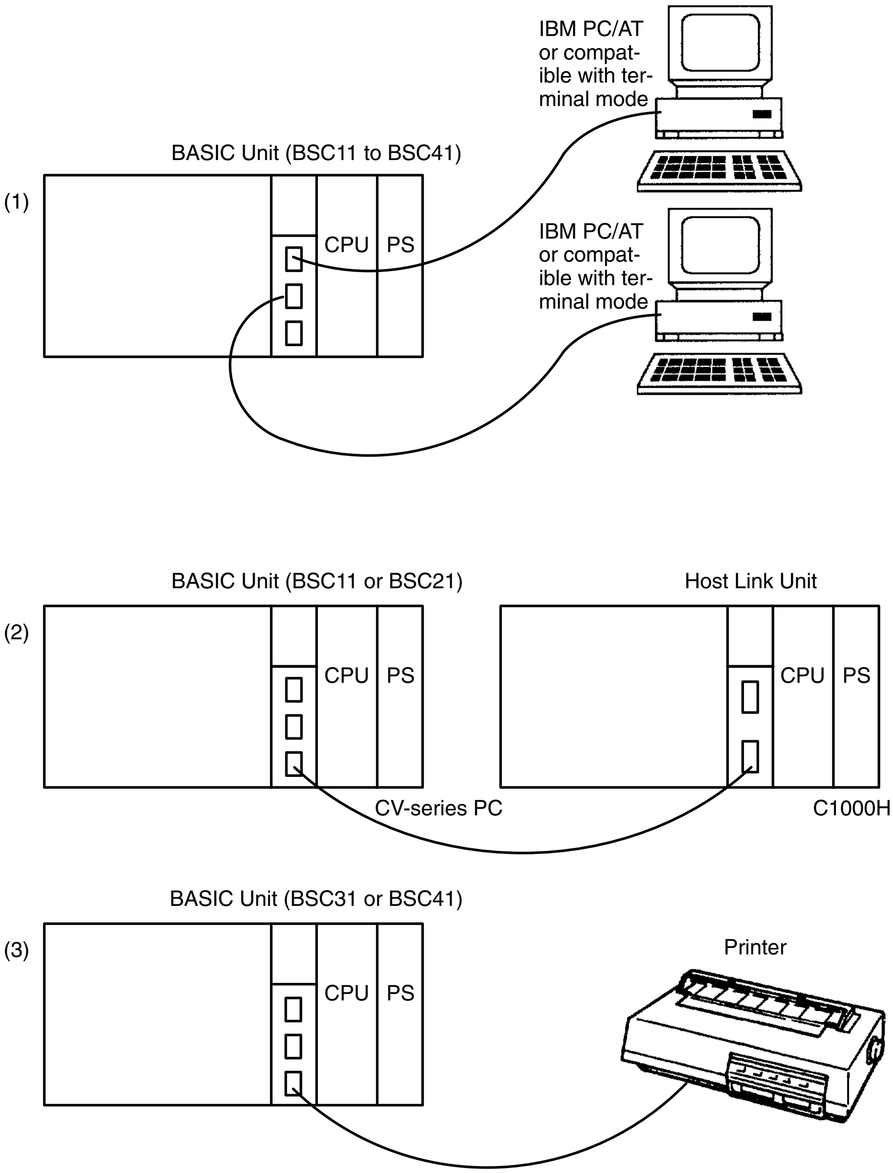
RS-422 Interface

Communicates with the C-series Host Link System, and writes data to the CPU Unit’s data memory on the Host Link System.

Centronics Interface

Outputs a square root to the printer.

Configuration Possibilities


IBM PC/AT or compatible with terminal mode
BASIC Unit (BSC11 to BSC41)
(1)
CPU
PS
IBM PC/AT or compatible with terminal mode
BASIC Unit (BSC11 or BSC21)
Host Link Unit
(2)
CPU
PS
CPU
PS
CV-series PC
C1000H
BASIC Unit (BSC31 or BSC41)
(3)
Printer
CPU
PS

Example Program

```
10 '******************************
20 '*RS-232C serial communication*
30 '*                            *
40 '******************************
50 PARACT 0
60 OPEN "COM2:N,8,2" AS #1 ........................... Sets RS-232C port to non-parity, 8 bits,
                                                        and 2 stop bits
```

```
70 ON COM (2) GOSUB *COMPRO ........................ Branches if interrupt is input to RS-232C
                                                     port
80 COM (2) ON ...................................... Enables port input of RS-232C
90 *START
100 INPUT ”Reception processing: R, transfer processing: T, termination process-
ing: E”;A$
110 IF A$=”R” OR A$=”r” THEN GOSUB *RCV ........... To reception processing
120 IF A$=”T” OR A$=”t” THEN GOSUB *TRNSFR ....... To transfer processing
130 IF A$=”E” OR A$=”e” THEN GOSUB * .............. To termination processing
140 GOTO *START
150 *E
160 COM (2) OFF .................................... Disables RS-232C port input
170 CLOSE #1
180 END
190 ’
200 *TRNSFR
210 COM (2) OFF .................................... Disables RS-232C port input
220 INPUT ”Input transfer data”;DATA$
230 PRINT #1, DATA$ ................................ Output to RS-232C port
240 COM (2) ON
250 RETURN
260 ’
270 *RCV
280 INPUT ”Stop reception? (Y/else)”;B$
290 IF B$=”Y” OR B$=”y” THEN COM (2) OFF ELSE COM (2) ON
300 RETURN
310 ’
320 *COMPRO
330 INPUT #1,DATA$ ................................. Input from RS-232C port
340 PRINT ”Sent data is: ”;DATA$
350 RETURN
360 END PARACT

10 ’*************************************************
20 ’*RS-422 Host Computer Program for C-series Host Link*
30 ’*                                                *
40 ’*************************************************
50 OPTION LENGTH 100
60 PARACT 0
70 OPEN ”COM3:E,7,2” AS #1 ......................... Open RS-422C port
80 *SND?
90 INPUT ”                                    ”;TD$
100 TC$=”@00WD0001”
110 T$=TC$+TD$
120 GOSUB *FCSSET
130 TXD$=T$+FCS$+”+”
140 PRINT ”TXD$=”;TXD$ ............................ Data transferred to Host Link Unit Trans-
                                                     fer
150 PRINT #1, TXD$ ................................. Transmission
160 *RCV
170 TUP=0
180 ON ALARM 100 GOSUB *TIMEUP
190 ALARM ON
200 INPUT #1,RXD$ .................................. Reception wait
210 ALARM OFF
220 IF TUP=1 GOTO *ERPRINT ......................... Judgment of reception timeout
230 R$=MID$(RXD$,6,2)
240 IF R$<>”00” GOTO *ER ........................... Response error
250 PRINT ”RXD$=”;RXD$+”                          OK”
260 *CMPLT
270 CLOSE #1
280 END
```

```
290 ’
300 *TIMEUP ...................................... Reception timeout processing
310 ER$=”TIME UP”
320 TUP=1
330 RETURN
340 ’
350 *ER .......................................... Response error processing
360 ER$=RXD$+”                          NG”
370 GOTO *ERPRINT
380 ’
390 *ERPRINT ..................................... Display of error
400 PRINT ”ERROR”
410 PRINT ”RXD$=”;ER$
420 GOTO *RCV
430 ’
440 *FCSSET ...................................... Calculation of data for frame check
450 L=LEN(T$)
460 A=0
470 FOR J=1 TO L
480 TJ$=MID$(T$,J,1)
490 A=ASC(TJ$) XOR A
500 NEXT J
510 FCS$=HEX$(A)
520 IF LEN(FCS$)=1 THEN FCS$=”0”+FCS$
530 RETURN
540 END PARACT
```

```
10 ’********************************
20 ’*Example Program of Printer Port*
30 ’*                              *
40 ’********************************
50 PARACT
60 FOR I=0 TO 10
70 ATAI=SQR(I)
80 TEXT$=STR$(ATAI)
90 LPRINT I;”Square root”+TEXT$
100 NEXT I
110 END PARACT
```

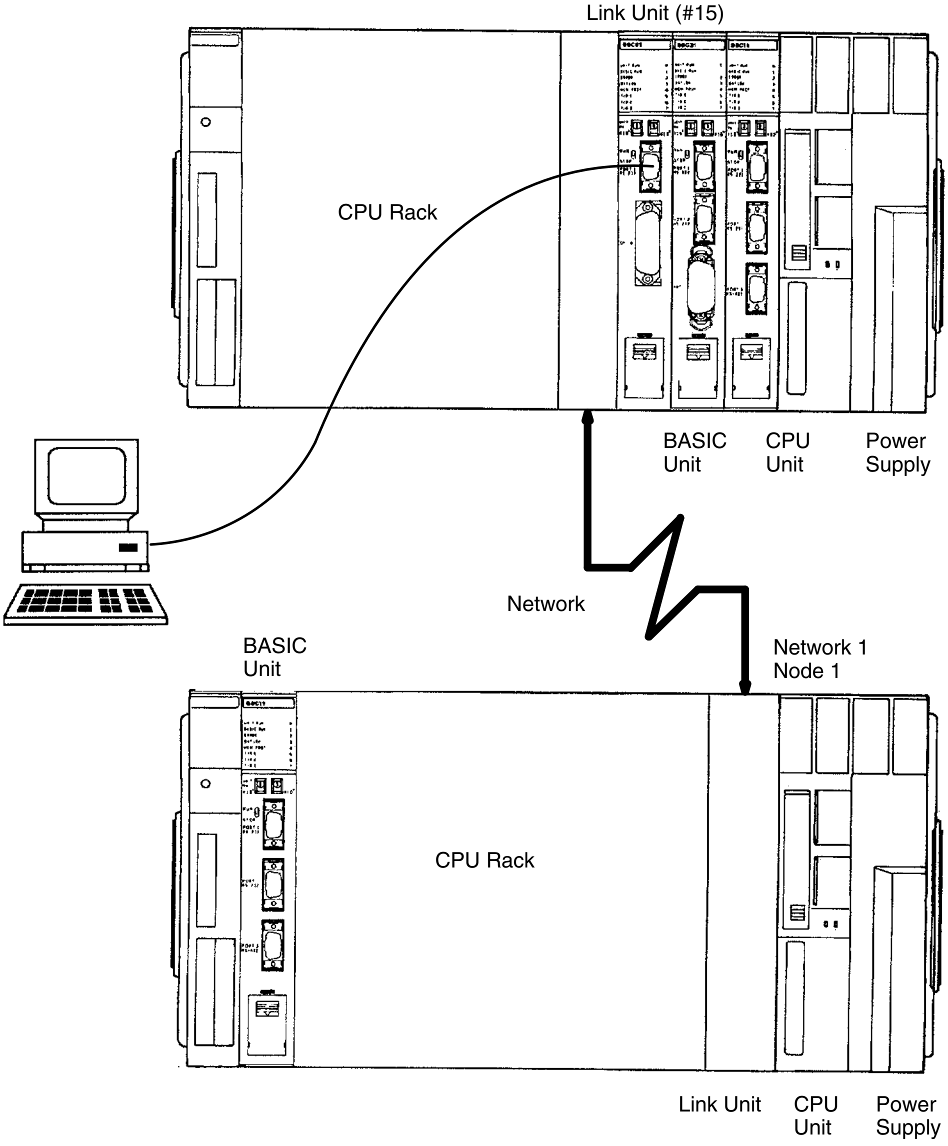
Note Memory switch bit 13 must be ON before using the Kanji printer (KI or KO). Refer to *3-3 Memory Switches*.

PC Communications

Operation

Writes or reads data to or from the PC connected through a network.
Writes data to the memory of the node 1 PC in network 1.
The CPU Unit checks whether data has been written, and sends back the data as is. The BASIC Unit reads the data sent from the CPU Unit by means of an interrupt.

Configuration



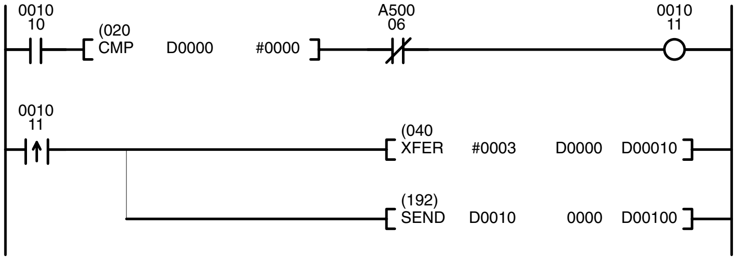
Example Program

```
10 '************************************************************
20 '* Data is written to CPU Unit and same data is sent back to Basic   *
30 '* Unit, read in a interrupt, and compared with original data.       *
40 '************************************************************
50 PARACT 0
60 DIM DM(3),R(3)
70 A=1 : B=&H10 : C=&H100
80 PC WRITE "#1.1,@D,0,3,3H4";A,B,C . ............... Writes data memory of network 1 and
                                                      node 1
90 ON PC (1) GOSUB *RCV
100 PC (1) ON
```

```
110 PAUSE
120 PC READ ”#1.1,@D,0,3,S3H4”;DM(0) . .............. Reads data memory
130 FOR I=0 TO 2
140        IF DM(I) <> R(I) THEN PRINT ”Comparison error”;I
                                                      Comparison of receive data
150 NEXT I
160 END
170 ’
180 *RCV
190 PC READ ”S3H4”;R(0)  . ........................... Reads data transferred from PC
200 RETURN
210 ’
220 END PARACT
```

CPU Unit Ladder Diagram

In the following program, the SEND(192) instruction is executed if D00000 contains anything but all-zeros, i.e., if data has been sent from the BASIC Unit. A differentiated condition is used to execute SEND(192) so that it is executed only once.



The following work bits and flags are used in this program: CIO 001010 enables operation; CIO 001011 is used to signal when the content of D00000 is non-zero; and A50006 is the Equals Flag.

The control data for SEND(192) must be set in advance as follows:

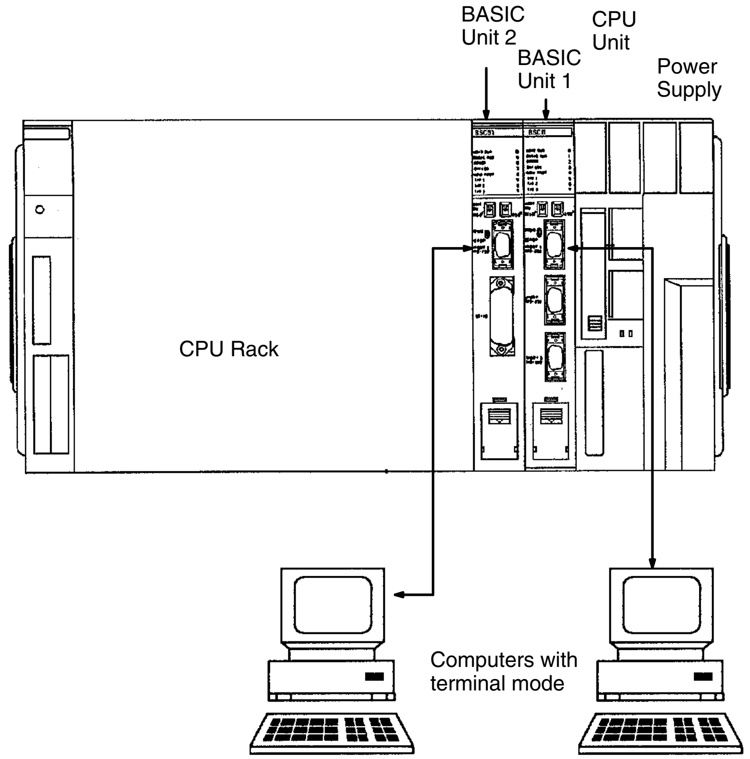
D00100	0003
D00101	0100
D00102	001F
D00103	0000
D00104	0000

Communicating Between BASIC Units

Operation

Communication is performed between two BASIC Units mounted on the same PC. Data is sent from Unit 0 and Unit 1 processes the data.

Configuration



Example Program

```
10 ′***********************************
20 ′*Communication between BASIC Units*
30 ′*Execute this Program on Unit 0   *
40 ′***********************************
50 PARACT 0
60 ′
70 OPEN ″FINS:00.00.17″ AS #1 . ..................... Open BASIC Unit of network 0, node
                                                      0, and Unit 0
80 PRINT #1, ″Please return this data.″
90 INPUT #1,REVERSE$
100 PRINT ″Returned data is  ″;REVERSE$
110 CLOSE #1
120 END PARACT
```

```
10 ′***********************************************
20 ′*Program for communication between BASIC Units*
30 ′*Execute this program on Unit 1               *
40 ′************************************************
50 PARACT 0
60 ′
70 OPEN ″FINS:00.00.16″ AS #1 . ..................... Open BASIC Unit of network 0, node 0,
                                                      and Unit 0
80 ′
90 ON FINS GOSUB *RCV
100 FINS ON
110 PAUSE
120 CLOSE #1
```

```
130 END
140 ′
150 *RCV
170 INPUT #1,RCVD$
180 PRINT ″Received:  ″;RCVD$
190 PRINT #1,RCVD$
200 RETURN
210 ′
220 END PARACT
```

Note Start the program for Unit #1 first

File Input/Output

Operations

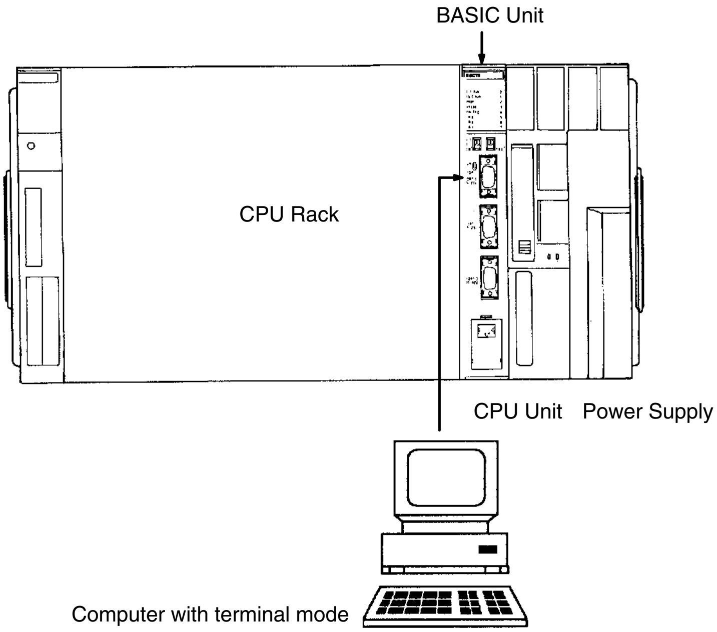
Sequential File:

1, 2, 3...
1. Opens a file in the Memory Card of the CPU Unit.
2. Using the keyboard, sequentially reads and writes data to the file. To end, `999` is input.
3. Reads the written sequential file and displays the data.

Random-access File:

1, 2, 3...
1. Opens a file in the memory card of the CPU Unit.
2. Identifies whether data input from the keyboard is to be read from or written to the file, and writes to or reads from a specified record number.

Configuration



Example Program

```
10 ′****************
20 ′*Sequential file*
30 ′*              *
40 ′****************
50 PARACT 0
60 DIM E$50,F$50,G$50
70 OPEN ″1:DATA2″ FOR OUTPUT AS #1 ................ Open new sequential file to be output on
                                                    data memory of CPU Unit
```

```
80    A$="  OMRON  "
90    B$="  CV500  "
100   C$="VERSION 1"
110   D$="BASIC UNIT"
120   WRITE #1,A$,B$ . ........................... Output data to sequential file (data com-
                                                   pression)
130   PRINT #1,USING "&       & &        &";C$,D$  Output data to sequential file with format
140   GOSUB *WRT
150 CLOSE . ...................................... Close opened file
160 OPEN "0:DATA2" FOR INPUT AS #1 . ............. Open sequential file to be input
170   PRINT "Contents of data file are as follows:"
180   LINE INPUT #1,F$ . ......................... Read one entire line to character vari-
                                                   able (F$)
190   PRINT F$
200   LINE INPUT #1,F$
210   PRINT F$
220   GOSUB *RD
230 CLOSE
240 END
250 '
260 *WRT . ....................................... Processing to output data to sequential
                                                   file
270 INPUT "Input data (to end writing, input 999)";E$
280 IF E$="999" THEN RETURN
290 PRINT #1,E$ . ................................ Output data to sequential file
300 GOTO *WRT
310 '
320 *RD . ........................................ Processing to input data from sequential
                                                   file
330 IF EOF(1) THEN RETURN . ...................... Branch if data has run out
340 INPUT #1,G$ . ................................ Read data
350 PRINT G$
360 GOTO *RD
370 END PARACT

10 '************
20 '*Random file*
30 '*          *
40 '************
50 OPTION LENGTH 100 . ........................... Set default character length of 100
60 PARACT 0
70 ON ERROR GOTO *OCCR
80 OPEN "0:DATA3" AS #1 . ........................ Open random file on memory card
90     FIELD #1,50 AS A$ . ....................... Assign variable area
100    PRINT "Input [W] to write file"
110    PRINT "Input [R] to read file"
120    PRINT "Input [E] to end"
130    B$=INPUT$(1) . ............................ Input condition from buffer to character
                                                   string
140    IF B$="w" OR B$="W" THEN GOSUB *WRT
150    IF B$="r" OR B$="R" THEN GOSUB *RD
160    IF B$="e" OR B$="E" THEN GOTO *E
170    GOTO 100
180    *E
190    PRINT "The size of data file is ";LOF(1) . . Size depends on file record no.
200 CLOSE #1 . ................................... Closing the file
210 END
220 '
230 *WRT . ....................................... Write subroutine
240 INPUT "Specify record number (1-999):",REC%
250 IF REC%>999 THEN ERROR 1 . ................... Set error generation number (ERR=1)
260 IF REC%<1 THEN ERROR 2 . ..................... Set error generation number (ERR=2)
```

```
270 LINE INPUT ”Data: ”;C$
280 PRINT ”Write the data? (Y/[ELSE])”
290 D$=INKEY$
300 IF D$=””THEN GOTO 290
310 IF D$<>”Y” AND D$<>”y” THEN RETURN
320 LSET A$=C$ ....................................... Set data in buffer
330 PUT #1,REC% ...................................... Write buffer data
340 RETURN ........................................... End of write subroutine
350 ’
360 *RD .............................................. Read subroutine
370 INPUT ”Specify record number (1-999):”;REC%
380 IF REC%>999 THEN ERROR 1
390 IF REC%<1 THEN ERROR 2
400 GET #1,REC% ...................................... Read data to buffer
410 PRINT A$
420 RETURN ........................................... End of read subroutine
430 ’
440 *OCCR ............................................ Error processing subroutine
450 IF ERR=1 THEN PRINT ”Record number is too large”
460 IF ERR=2 THEN PRINT ”Record number is too small”
470 ENUM=ERL
480 IF ENUM=400 THEN PRINT ”No data exists in specified record number”
490 RESUME 100
500 END PARACT
```

Reserved Words

ABS

ACOS

ALARM ON / OFF / STOP

ASC

ASIN

ATN

AUTO

BITON / BITOFF

BREAK

CALL

CDBL

CHR$

CINT

CLOSE

CLS

CMD DELIM

CMD PPR

CMD TIMEOUT

COM ON / OFF / STOP

CONT

COS

CSNG

CVI / CVS / CVD

DATA

DATE$

DEF FN

DEF USR

DEFINT / DEFSNG/ DEFDBL / DEFSTR

DEG SEG

DELETE

DIM

EDIT

END

END PARACT

EOF

ERL/ERR

ERROR

EXIT

EXP

FIELD

FILES / LFILES

FINS ON / OFF / STOP

FIX

FOR... TO... STEP...

FRE

GET

GOSUB / RETURN

GOTO

HEX$

IEEE(0)

IEEE(1)

IEEE(2)

IEEE(4)

IEEE(5)

IEEE(6)

IEEE(7)

IF... GOTO... ELSE...

INKEY$

INPUT

INPUT #

INPUT @

INPUT$

INSTR

INT

INTRB

INTRL

INTRR

IRESET REN

ISET IFC

ISET REN

ISET SRQ

```
KEY ON / OFF / STOP
KILL
LEFT$
LEN
LET
LINE INPUT
LINE INPUT #
LINE INPUT @
LIST / LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPRINT
LPRINT USING
LSET/RSET
MERGE
MESSAGE
MID$
MKI$ / MKS$ / MKD$
MON
MSET
NAME
NEW
NEXT
OCT$
ON ALARM GOSUB
ON COM GOSUB
ON ERROR GOTO
ON FINS GOSUB
ON GOSUB
ON GOTO
ON KEY GOSUB
ON PC GOSUB
ON SIGNAL GOSUB
ON SRQ GOSUB
ON TIME$ GOSUB
ON TIMER GOSUB
OPEN
OPTION BASE
OPTION ERASE
OPTION LENGTH
PARACT
PAUSE
PC ON / OFF / STOP
PC READ
PC WRITE
PEEK
PGEN
PINF
PNAME
POKE
POLL
PPOLL
PRINT #
PRINT # USING
PRINT / ?
PRINT @
PRINT USING
PUT
RANDOMIZE
RBYTE
RDIM
READ
RECEIVE
REM
RENUM
RESTORE
RESUME
RIGHT$
RND
ROMLOAD
ROMSAVE
ROMVERIFY
RUN
RUNr
SAVE
SEARCH
SEND
SENDSIG
SGN
SIGNAL ON / OFF / STOP
SIN
SPACE$
SPC
SQR
SRQ ON/OFF/STOP
STATUS
STEP
STOP
STR$
STRING$
SWAP
TAB
TAN
TASK
TIME$
TIME$ ON / OFF / STOP
TIMER ON / OFF / STOP
TROFF
TRON
TWAIT
USR
VAL
VARPTR
VERIFY
VLOAD
VSAVE
WBYTE
WHILE/WEND
WRITE
WRITE #
```

Appendix E
BASIC Instructions

The instructions of the BASIC Unit are broadly classified into commands, statements, functions, and GP-IB instructions.

Commands can be typed in and executed directly from the console in edit or debug mode. Some commands can also be used as statements.

Statements are used in BASIC programs to do most of the program's work and to control the program's execution.

Functions perform a specified calculation and return the result of the calculation to the program. Many functions require one or more arguments.

GP-IB instructions, which control the GP-IB interface, are sub-divided into statements and functions. The GP-IB instructions can be used with the CV500-BSC51 and CV500-BSC61 only.

How to Use this Table

Instruction: This column lists the names of the commands, statements, and functions in alphabetical order.

Syntax: This column describes the form(s) in which the instruction appears in a program, using the following notation:

- Words and symbols in `typewriter font` should be entered exactly as written.
- Items in square brackets ([ ]) may be omitted.
- Items in curly brackets ({ }) indicate choices; alternatives are delimited from each other with the vertical bar character (|). Select one of the alternatives.
- An asterisk (*) indicates that the preceding item or items may be repeated.
- ␣ indicates a required space. (Spaces can also be used between words and symbols to increase program readability.)
- Words in *italics* are English descriptions of the element that should be substituted. For example, *line-no.* should be replaced with an actual line number.

Purpose: This column presents a brief description of the instruction.

Command List

These instructions may be used in EDIT or DEBUG mode. Instructions marked with a diamond (♦) may also be used as statements in programs.

Instruction	Syntax	Purpose
@	@ [task-number]	Selects a task to be debugged.
AUTO	AUTO [start-line-no.] [, increment]	Automatically generates line numbers when a program is typed in.
BREAK	BREAK [{DELETE {ALL | line-no. [, line-no.]*} | line-no. [, line-no.]*}]	Sets, deletes, or lists breakpoints.
CLS♦	CLS	Clears screen.
CONT	CONT	Resumes execution of program.
DELETE	DELETE [start-line-no.] [–[end-line-no.]]	Deletes program lines.
EDIT	EDIT [line-no.]	Edits one line of program.
FILES / LFILES	FILES [drive-no.]	Displays names and size of files in drive.
	LFILES [drive-no.]	Prints names and sizes of files in drive.
KILL♦	KILL "file-name"	Deletes file.
LET♦	[LET] variable-name = expression	Stores value of *expression* in variable.
LIST / LLIST	LIST [start-line-no.] [– [end-line-no.]]	Displays all or part of program.
	LLIST [start-line-no.] [– [end-line-no.]]	Prints all or part of program.

Instruction	Syntax	Purpose
LOAD	LOAD *"file-name"*	Reads BASIC program into current program area.
MERGE	MERGE *"file-name"*	Reads BASIC program to current program area. Program is merged with any existing program.
MON	MON	Sets monitor mode.
MSET	MSET [*address*]	Sets upper limit of BASIC program area to allocate machine language program area.
NAME♦	NAME *"old-file-name"* AS *"new-file-name"*	Changes file name.
NEW	NEW	Deletes program and variables.
PGEN	PGEN [*program-no.*]	Selects current program area.
PINF	PINF	Displays information on program area.
PNAME	PNAME *"program-name"*	Registers or deletes name of current program area.
PRINT♦	PRINT [*expression*] [{,|;|␣} [*expression*]]*	Displays value of expression.
LPRINT♦	LPRINT [*expression*] [{,|;|␣} [*expression*]]*	Prints value of expression.
PRINT USING♦	PRINT USING *format* ; *expression* [{,|;|␣} [*expression*]]*	Displays value of expression in specified format.
LPRINT USING♦	LPRINT USING *format* ; *expression* [{,|;|␣} [*expression*]]*	Prints value of expression in specified format.
RENUM	RENUM [*new-line-no.*] [, [*old-line-no.*] [, *increment*]]	Re-numbers program lines.
ROMLOAD	ROMLOAD	Reads information in EEPROM to user program area.
ROMSAVE	ROMSAVE	Writes information in user program area to EEPROM.
ROMVERIFY	ROMVERIFY	Verifies between EEPROM and user program area.
RUN♦	RUN [*"file-name"*] [, ERASE]	Starts program execution.
SAVE	SAVE *"file-name"*	Saves BASIC program to file.
STEP	STEP	Executes program one step at a time.
TROFF♦	TROFF [{*task-no.*|ALL}]	Stops output of line number trace.
TRON♦	TRON [{*task-no.*|ALL}]	Starts output of line number trace.
VERIFY	VERIFY *"file-name"*	Verifies program.
VLOAD♦	VLOAD *"file-name"*	Reads contents of non-volatile variable from file.
VSAVE♦	VSAVE *"file-name"*	Saves contents of non-volatile variable to file.
WRITE♦	WRITE *expression* [{,|;|␣}*expression*]*	Displays value of expression.

♦The command can also be used as a statement in a program.

Statement List

Instruction	Syntax	Purpose
ALARM ON / OFF / STOP	ALARM {ON | OFF | STOP}	Enables, disables, or stops time interrupt.
BITON / BITOFF	{BITON | BITOFF} *integer-variable*, *bit-position*	Turns ON (1) or OFF (0) the specified bit of an integer variable.
CALL	CALL *name* [(*argument* [, *argument*]*)]	Calls a machine language program (subroutine) stored in memory.
CLOSE	CLOSE [#*file-no.* [, #*file-no.*]*]	Closes file.
CLS	CLS	Clears screen.
COM ON / STOP (OFF is same as STOP)	COM [(*port-no.*)] {ON | STOP}	Enables or stops interrupt from communication line.

Instruction	Syntax	Purpose
DATA	DATA constant [, constant]*	Stores numeric and character constants for use by READ statements.
DEF FN	DEF FNfunction-name [(argument [, argument]*)] = function-definition-expression	Defines function.
DEG SEG	DEF SEG = segment-address	Declares segment address.
DEF USR	DEF USR [no.] = start-address	Defines execution start address of machine language USR function.
DEFINT/DEFSNG/ DEFDBL/DEFSTR	{DEFINT | DEFSNG | DEFDBL | DEFSTR} {variable-name | character–character} [, {variable-name | character–character}]*	Declares variable type.
DIM	DIM variable-name [(subscript [, subscript]*)] [maximum-number-of-characters] [, variable-name [(subscript [, subscript]*)] [maximum-number-of-characters]]*	Declares an array variable or fixed-length string.
END	END	Terminates task.
END PARACT	END PARACT	Declares the end of a task.
ERROR	ERROR error-no.	Simulates generation of error.
EXIT	EXIT task-no.	Terminates specified task.
FIELD	FIELD #file-no., width AS character-string-variable [, width AS character-string-variable]*	Assigns field variable to random file buffer.
FINS ON / STOP (OFF is same as STOP)	FINS {ON | STOP}	Enables or stops interrupts from network.
FOR... TO... STEP... NEXT...	FOR variable = initial-value TO final-value [STEP increment] NEXT [variable [, variable]*]	Repeatedly execute group of statements enclosed by FOR and NEXT statements.
GET	GET #file-no. [, record-no.]	Reads data from random file.

GOSUB / RETURN	GOSUB {line-no. | label}
	RETURN
		Calls subroutine / returns from subroutine.

GOTO	GOTO {line-no. | label}	Branches to specified line or label.

IF... THEN...
ELSE...
IF... GOTO...
ELSE...
	IF conditional-expression THEN {statement | line-no. | label} [ELSE {statement | line-no. | label}]
	IF conditional-expression GOTO {line-no. | label} [ELSE {statement | line-no. | label}]
		Selects statement to be executed according to result of *conditional-expression*.

INPUT	INPUT [WAIT expression,] ["prompt" {, | ;}] variable [, variable]*	Inputs data to specified variable.
INPUT #	INPUT #file-number, variable [, variable]*	Reads data from file into specified variable.
KEY ON / OFF / STOP	KEY (key-no.) {ON | OFF | STOP}	Enables, disables, or stops interrupts from console numeric keys.
KILL	KILL "file-name"	Deletes file.
LET	[LET] variable-name = expression	Assigns the value of an expression to a variable
LINE INPUT	LINE INPUT [WAIT expression,] ["prompt" {, | ;}] character-variable	Inputs a whole line to a character string variable.
LINE INPUT #	LINE INPUT #file-no., character-variable	Reads one line from a file into a character string variable.
LOCATE	LOCATE horizontal-position, vertical-position	Moves cursor on screen.
LSET/RSET	LSET character-variable = character-expression RSET character-variable = character-expression	Substitutes data into field variable.
LPRINT	LPRINT [expression] [{, | ; | ␣} [expression]]*	Prints value of expression.

Instruction	Syntax	Purpose
LPRINT USING	LPRINT USING format ; expression [{, | ; | ⎵} [expression]]*	Output value of expression using specified format.
MESSAGE	MESSAGE function, message-no.	Allocates and releases message numbers.
MID$	MID$(character-expression, expression [, expression]) [= character-expression]	Returns or replaces part of character string variable.
NAME	NAME "old-file-name" AS "new-file-name"	Changes file name.
ON ALARM GOSUB	ON ALARM time GOSUB {line-no. | label}	Specifies interrupt time and defines interrupt routine.
ON COM GOSUB	ON COM [(port-no.)] GOSUB {line-no. | label}	Defines subroutine to process interrupts from communication line.
ON ERROR GOTO	ON ERROR GOTO {0 | line-no. | label}	Defines error processing routine and starts error trap.
ON FINS GOSUB	ON FINS GOSUB {line-no. | label}	Defines subroutine to process interrupts from network.
ON GOSUB	ON expression GOSUB {line-no. | label} [, {line-no. | label}]*	Selects and calls one of several subroutines based on the value of *expression.*
ON GOTO	ON expression GOTO {line-no. | label} [, {line-no. | label}]*	Selects and branches to one of several locations based on the value of *expression.*
ON KEY GOSUB	ON KEY (key-no.) GOSUB {line-no. | label}	Defines subroutine to process numeric key interrupts.
ON PC GOSUB	ON PC (interrupt-no.) GOSUB {line-no. | label}	Defines subroutine to process interrupts from CPU Unit.
ON SIGNAL GOSUB	ON SIGNAL (signal-no.) GOSUB {line-no. | label}	Defines interrupt subroutine for user-defined or system signal.
ON TIME$ GOSUB	ON TIME$ = "time" GOSUB {line-no. | label}	Defines subroutine to be executed at a certain time.
ON TIMER GOSUB	ON TIMER = interval GOSUB {line-no. | label}	Specifies subroutine to be executed after a certain interval
OPEN	OPEN "file-name" [FOR {INPUT | OUTPUT | APPEND}] AS #file-no.	Opens file.
OPTION BASE	OPTION BASE {0 | 1}	Declares subscript of first array element.
OPTION ERASE	OPTION ERASE	Declares initialization of non-volatile variables.
OPTION LENGTH	OPTION LENGTH no.-of-characters	Declares default length for fixed character strings.
PARACT	PARACT task-no. [WORK no.-of-bytes]	Declares beginning of task.
PAUSE	PAUSE	Stops execution of task until interrupt occurs.
PC ON / STOP (OFF is same as STOP)	PC (interrupt-no.) {ON | STOP}	Enables or stops interrupt from CPU Unit.
PC READ	PC READ [WAIT time,] "[[#network, node,] source-area, start-word, no.-of-words,] format [, format]*"; variable [, variable]*	Reads data from CPU Unit into *variable.*
PC WRITE	PC WRITE [WAIT time,] "[[#network, node,] destination-area, start-word, no.-of-words,] format [, format]*"; variable [, variable]*	Writes value of *variable* to CPU Unit.
POKE	POKE address, expression	Writes data to specified address of memory.
PRINT / ?	{PRINT | ?} [expression] [{, | ; | ⎵} [expression]]*	Displays value of expression.
PRINT #	PRINT #file-no., [expression] [{, | ; | ⎵} [expression]]*	Outputs value of expression to a file.
PRINT USING	PRINT USING format ; expression [{, | ; | ⎵} [expression]]*	Output value of expression in specified format.
PRINT # USING	PRINT #file-no., USING format ; expression [{, | ; | ⎵} [expression]]*	Output value of expression in specified format to a file.

Instruction	Syntax	Purpose
PUT	PUT #*file-no.* [, *record-no.*]	Writes data to random file.
RANDOMIZE	RANDOMIZE [*expression*]	Initializes random series.
RDIM	RDIM *variable-name* [(*subscript* [, *subscript*]*)] [*maximum-number-of-characters*] [, *variable-name* [(*subscript* [, *subscript*]*)] [*maximum-number-of-characters*]]*	Declares non-volatile variables.
READ	READ *variable* [, *variable*]*	Reads data from DATA statement and stores it in *variable*.
RECEIVE	RECEIVE *message-no.*, *character-variable*	Receives message.
REM	REM [*comment-text*]	Causes the BASIC Unit to ignore the *comment-text*.
RESTORE	RESTORE [{*line-no.* | *label*}]	Specifies re-use of values in a DATA statement
RESUME	RESUME [{0 | *line-no.* | *label* | NEXT}]	Exits from error processing routine.
RUN	RUN ["*file-name*"] [, ERASE]	Starts program execution.
SEND	SEND *message-no.*, *character-expression*	Sends message.
SENDSIG	SENDSIG *signal-no.*, *task-no.*	Generates signal.
SIGNAL ON / OFF / STOP	SIGNAL *signal-no.* {ON | OFF | STOP}	Enables, disables, or stops signal interrupt.
STOP	STOP	Stops program execution.
SWAP	SWAP *variable-name*, *variable-name*	Swaps values of two variables.
TASK	TASK *task-no.*	Starts terminated task.
TIME$ ON / OFF / STOP	TIME$ {ON | OFF | STOP}	Enables, disables, or stops time interrupt.
TIMER ON / OFF / STOP	TIMER {ON | OFF | STOP}	Enables, disables, or stops timer interrupt.
TROFF	TROFF [{*task-no.* | ALL}]	Stops output of line number trace.
TRON	TRON [{*task-no.* | ALL}]	Starts output of line number trace.
TWAIT	TWAIT *task-no.*	Waits for termination of task.
VLOAD	VLOAD "*file-name*"	Reads contents of non-volatile variable from file.
VSAVE	VSAVE "*file-name*"	Saves contents of non-volatile variable to file.
WHILE/WEND	WHILE *conditional-expression* WEND	Repeatedly execute series of statements while condition is satisfied.
WRITE	WRITE *expression* [{, | ; | ␣} [*expression*]]*	Outputs value of expression.
WRITE #	WRITE #*file-no.*, *expression* [{, | ; | ␣} [*expression*]]*	Outputs value of expression to a file.

Function List

Instruction	Syntax	Purpose
ABS	ABS(*expression*)	Calculates the absolute value of the *expression*.
ACOS	ACOS(*expression*)	Calculates arc cosine of the *expression*.
ASC	ASC(*character-expression*)	Returns the ASCII code of the first character of *character-expression.*
ASIN	ASIN(*expression*)	Calculates the arc sine of the *expression*.
ATN	ATN(*expression*)	Calculates the arc tangent of the *expression*.
CDBL	CDBL(*expression*)	Converts *expression* into a double-precision real number.
CHR$	CHR$(*expression*)	Converts *expression* into characters.
CINT	CINT(*expression*)	Rounds any fractional part of *expression*
COS	COS(*expression*)	Returns cosine of *expression*.

Instruction	Syntax	Purpose
CSNG	CSNG(*expression*)	Converts *expression* into single-precision real number.

CVI / CVS / CVD	CVI(*2-character-string*)
	CVS(*4-character-string*)
	CVD(*8-character-string*)
		Converts character string into numeric value.

DATE$	DATE$ [= "*year*/*month*/*day*"]	Returns date of internal clock, or sets date.
EOF	EOF(*file-no.*)	Returns true (–1) if *file-no.* has reached end of file; false (0) otherwise.
ERL/ERR	ERL ERR	Return line on which error has occurred (ERL) and error code (ERC).
EXP	EXP(*expression*)	Calculates exponential function of *expression* ($e^{expression}$)
FIX	FIX(*expression*)	Truncates any fractional part of *expression*.
FRE	FRE(*expression*)	Returns size of unused memory area.
HEX$	HEX$(*expression*)	Returns a character string with the value of *expression* expressed as a hexadecimal number.
INKEY$	INKEY$	Returns next character in keyboard buffer.
INPUT$	INPUT$(*expression* [, #*file-no.*])	Reads character string of specified length from specified file.
INSTR	INSTR([*expression*,] *character-string*, *key-string*)	Searches for *key-string* in *character-string* and returns its position.
INT	INT(*expression*)	Returns the largest integer which does not exceed *expression.*
INTRB INTRL INTRR	INTRB INTRL INTRR	Variables containing information on an interrupt that has occurred.
LEFT$	LEFT$(*character-expression*, *expression*)	Returns the leftmost *expression* characters from *character-expression.*
LEN	LEN(*character-expression*)	Returns length of *character-expression.*
LOC	LOC(*file-no.*)	Returns current logical position in file.
LOF	LOF(*file-no.*)	Returns size of file.
LOG	LOG(*expression*)	Calculates natural logarithm of *expression*
MID$	MID$(*character-expression*, *length* [, *position*])	Returns *length* characters from *character-expression* starting from *position.*
MKI$ / MKS$ / MKD$	MKI$(*integer-value*) MKS$(*single-precision-value*) MKD$(*double-precision-value*)	Converts numeric value into character string.
OCT$	OCT$(*expression*)	Returns a character string with the value of *expression* expressed as an octal number.
PEEK	PEEK(*address*)	Returns contents of the specified address.
RIGHT$	RIGHT$(*character-expression*, *expression*)	Returns the rightmost *expression* characters from *character-expression*
RND	RND(*expression*)	Returns random number.
SEARCH	SEARCH(*integer-array* [, *expression*] [, *start-element*] [, *increment*])	Searches for first occurrence of the integer value *expression* in *integer-array* and returns element number.
SGN	SGN(*expression*)	Returns –1, 0, or 1 depending on whether *expression* is negative, zero, or positive.
SIN	SIN(*expression*)	Calculates sine of *expression.*
SPACE$	SPACE$(*expression*)	Returns a character string containing *expression* spaces.
SPC	SPC(*expression*)	Outputs *expression* spaces.
SQR	SQR(*expression*)	Calculates the square root of *expression*.

Instruction	Syntax	Purpose
STR$	STR$(*expression*)	Returns a character string with the value of *expression* expressed as a decimal number
STRING$	STRING$(*expression*, {*character-string* | *character-code*})	Returns a string with *expression* copies of the first character of *character-expression* or *character-code*.
TAB	TAB(*expression*)	Moves cursor to specified column.
TAN	TAN(*expression*)	Calculates tangent of *expression*.
TIME$	TIME$ [= "*hour*:*minute*:*second*"]	Returns time of internal clock, or sets time.
USR	USR[*func-no.*](*argument*)	Calls a machine language function
VAL	VAL(*character-expression*)	Converts *character-expression* into a numeric value.
VARPTR	VARPTR(*variable-name*) [, *feature*]	Returns memory address of variable.

GP-IB Instruction List

Statements

Instruction	Syntax	Purpose
CMD DELIM	CMD DELIM = *delimiter-code*	Specifies delimiter.
CMD PPR	CMD PPR = *mode*	Selects PPR mode.
CMD TIMEOUT	CMD TIMEOUT = *timeout-parameter*	Specifies limit value for timeout check.
INPUT @	INPUT@ [*talker-address* [, *listener-address* [, *listener-address*]*]]; *variable* [, *variable*]*	Receives data sent from specified talker and stores it in *variable*.
IRESET REN	IRESET REN	Makes REN (remote enable) false.
ISET IFC	ISET IFC [, *integer*]	Transmits IFC (interface clear).
ISET REN	ISET REN	Makes REN (remote enable) true.
ISET SRQ	ISET SRQ [@] [N]	Transmits SRQ (service request).
LINE INPUT @	LINE INPUT@ [*talker-address* [, *listener-address* [, *listener-address*]*]]; *character-string-variable*	Receives string data sent from specified talker and substitutes it into character string variable.
ON SRQ GOSUB	ON SRQ GOSUB {*line-no.* | *label*}	Specifies first line of SRQ subroutine.
POLL	POLL *talker-address*, *numeric-variable* [; *talker-address*, *numeric-variable*]*	Performs serial polling.
PPOLL	PPOLL [PPU] [, *listener-address*, *integer*]*	Assigns response output line for parallel polling.
PRINT @	PRINT@ [*listener-address* [, *listener-address*]*]; [*data* [, *data*]*] [@]	Transmits data as ASCII character string.
RBYTE	RBYTE [*command*] [, *command*]*; [*numeric-variable* [, *numeric-variable*]*	Receives binary data after transmitting multi-line message.
SRQ ON/OFF/STOP	SRQ {ON | OFF | STOP}	Controls reception of SRQ.
WBYTE	WBYTE [c*ommand*] [, *command*]*]; [*data* [, *data*]*] [@]	Transmits multi-line message and binary data.

Functions

Instruction	Syntax	Purpose
`IEEE(0)`	`IEEE(0)`	Checks the delimiter.
`IEEE(1)`	`IEEE(1)`	Checks the initialized status of GP-IB interface.
`IEEE(2)`	`IEEE(2)`	Checks the talker and listener status, and received interface message.
`IEEE(4)`	`IEEE(4)`	Stores the device status of the device that transmits the service request during serial polling.
`IEEE(5)`	`IEEE(5)`	Stores the talker address of the device that transmits the service request during serial polling.
`IEEE(6)`	`IEEE(6)`	Stores the talker address of the device that does not respond to the serial polling.
`IEEE(7)`	`IEEE(7)`	Stores the data byte obtained as a result of parallel polling.
`STATUS`	`STATUS`	Stores device status.

Appendix F
Machine Language Commands

Each of the machine language monitor commands is described in detail on the following pages. In the description, the following syntax is used:

- Items in brackets [ ] may be omitted.
- An item followed by an asterisk (*) may be repeated.
- Words in *italics* are English descriptions of the value that must be supplied.

Note 1. All commands must be entered in upper case.

2. DS0 is generally used for address calculation. The target address is the specified address (offset) + DS0.
3. If any start address is greater than an end address, an error will occur.
4. Addresses (offsets) must be entered as numbers of 4 digits or less. (A 5-digit address will cause an error.) Leading 0's may be omitted.
5. Data must be entered as numbers of 1 or 2 digits. (3-digit data values will cause an error.) Leading 0's may be omitted.
6. If the monitor detects an input error, it will display a question mark (`?`).
7. The program counter (PC) and program segment (PS) are used for the `G`, `T`, and `B` commands.
8. The Backspace Key can be used to correct inputs until the carriage return is input.

Command: D

Function

Displays the contents of memory in hexadecimal notation.

Syntax

`D` [*start-address*][`.` *end-address*]

Explanation

Display the contents of the memory from a specified start address to end address (example 1). If the end address is omitted, only the byte at the start address is displayed (example 2) If both the start and end addresses are omitted, 8 bytes are displayed, starting from the address after that displayed previously (example 3). If the start address is omitted, the memory contents from the address after that displayed previously and ending at the specified end address (example 4).

To suspend the display, press `CTRL+S`. To resume, press `CTRL+Q`.

Examples

1, 2, 3...

1.
```
*D4001.4005↵
4001-20 30 40 50 60
```
2.
```
*D4010↵
4010-23
```
3.
```
*D↵
4011-34 56 78 9A BC DE F0 12
```
4.
```
*D.4021↵
4019-31 32 33 34 35 36 37 38
4021-39
```

Command: W

Function

Writes new values into memory.

Syntax

`W` *start-address*`:` *data*[`.` *data*]*

Explanation

Stores the specified *data* in memory beginning at *start-address* (example 1).

Up to 80 values can be stored with one `W` command.

Example

```
*W4000:12.34.56.78.9A↵
*D4000.4004↵
4000-12 34 56 78 9A
```

Command: M

Function

Moves a specified block of memory to another place in memory.

Syntax

`M` *destination-address* < *block-start* . *block-end*

Explanation

The source block and destination address must be in the same segment.

Note Make sure the source and destination areas do not overlap.

Example

```
*M000<4000.403F↵
```

Command: C

Function

Compares the contents of two blocks of memory.

Syntax

`C` *block-2-start* < *block-1-start* . *block-1-end*

Explanation

The contents of blocks 1 and 2 are compared and any differences are displayed. If no differences are found, only the next prompt is displayed.

Example

This command will compare memory locations from &H4000 to &H401A to locations from &H5000 to &H501A.

```
*C5000<4000.401A↵
4009-FF(FB)
4013-56(34)
*
```

The lines before the next `*` prompt show that differences were found at &H4009 (which contains &HFF) and &H5009 (which contains &HFB), and at &H4013 (&H56) and &H5013 (&H34).

Command: A

Function

Start assembling mnemonic codes.

Syntax

`A`

The prompt will change to an exclamation point (`!`). Enter the mnemonic codes:

`!`[*address*`:`]*mnemonic-code*↵*

Enter `X` to exit from the mnemonic assembler.

Explanation

The CPU’s mnemonics and operands are described in the NEC V25 (µPD70322) manual. (Some mnemonics and operands are slightly different. See the list on page 185 for details.)

The assembler’s location counter is updated appropriately each time a line has been entered, so it is not necessary to enter a new *address* in order to assemble into consecutive memory locations.

Example

```
*A↵
!3000:MOV IX,IY↵
3000-89 FE .. .. .. .. .. MOV IX,IY
!MOV AW,BW↵
3002-89 D8 .. .. .. .. .. MOV AW,BW
!X↵
*
```

Command: I

Function

Disassembles and displays the machine language program at a specified address.

Syntax

I[*start-address*][. *end-address*]

Explanation

Disassembles and displays the memory contents between *start-address* and *end-address* (example 1). If *end-address* is omitted, 20 instructions are displayed (starting at *start-address*) (example 2). If both *start-* and *end-address* are omitted, 20 instructions are displayed starting from the address after the one displayed immediately before (example 3). To display only one instruction, specify the same *start-address* and *end-address* (example 4).

Example

1, 2, 3...

1.
```
*I3000.3003↵
3000-89FE            MOV          IX,IY
3002-89D8            MOV          AW,BW
```
2.
```
*I3000↵
3000-89FE            MOV          IX,IY
3002-89D8            MOV          AW,BW
3004-1000            ADDC         [BW+IX],AL
3006-86E0            XCH          AL,AH
3008-26              DS1:
3009-8905            MOV          [IY],AW
300B-2438            AND          AL,38
300D-C0E803          SHR          AL,03
3010-B409            MOV          AH,09
3012-F6E4            MULU         AH
```
Displays 20 instructions

3.
```
*I↵
3025-50              PUSH         AW
3026-8CC8            MOV          AW,PS
3028-8ED8            MOV          DS0,AW
```
Displays 20 instructions

4.
```
*I3000.3000↵
3000-89FE            MOV          IX,IY
*
```

Command: S

Function

Saves (writes) the contents of the specified address range to a file on a memory card or to a port connected to a terminal. The format of the saved file can also be specified.

Syntax

SR *format start-address . end-address*
SFH *start-address . end-address . file-name*

Explanation

The save destination is indicated by the character after S: R is the terminal port; F is the CPU Unit memory card.

Format can be S (indicating Motorola S-records) or H (indicating Intel Hex Format). If the destination is the memory card, only the Intel Hex Format can be used.

File-name does not include the 3-character extension (.XXX)
Transfer is started immediately after the command has been entered.

Examples

1, 2, 3...

1.
```
*SRS5000.52FF
*
```

2. `*SFH4000.41FF.FILE2↵`
   `*`

Note To save the contents of memory by specifying save destination `R`, the CVSS is necessary. If the CVSS is not installed, the data are only displayed, and *not* saved.

Command: L

Function

Loads (reads) a section of memory from a file on the memory card or from the port to which a terminal is connected.

Syntax

`LR` *format* [*offset*]
`LFH`[*offset*]`.` *file-name*

Explanation

The load source is specified by the second letter of the command:
`R`: terminal connected to port
`F`: CPU Unit's memory card
If the source is the terminal, *format* can be either `S` (for Motorola S-records) or `H` (for Intel Hex Format). If the source is the memory card, only Intel Hex Format can be used.

The address to which the file contents is to be loaded is the specified address + segment value of DS0 + *offset*.

File-name does not include the extension (`.XXX`).

To abort this command, press `CTRL+Z` twice.

Examples

1, 2, 3...

1. `*LFH.FILE3↵`
2. `*LRH↵`
3. `*LRH1000↵`

Note To load the machine language program from a Terminal connected to the port, the CVSS is necessary. Memory contents cannot be directly loaded from a terminal other than those with CVSS.

Command: V

Function

Verifies the memory block transferred from the port to which a terminal is connected or the memory card of the CPU Unit against the contents of the BASIC Unit's memory.

Syntax

`VR` *format* [*offset*]
`VFH`[*offset*]`.` *file-name*

Explanation

The source of data to verify is specified by the second letter of the command:
`R`: terminal connected to port
`F`: CPU Unit's memory card
If the source is the terminal, *format* can be either `S` (for Motorola S-records) or `H` (for Intel Hex Format). If the source is the memory card, only Intel Hex Format can be used.

The address to be verified is the specified transfer address + segment value of DS0 + *offset*.

The result of the verification can be checked by the `X` command (refer to the description of the `X` command).

To abort this command, press `CTRL+Z` twice.

Examples

1, 2, 3...

1. `*VRS↵`
2. `*VFH1000.FILE4↵`

Command: X

Function

Displays the result of the previous `S`, `L`, or `V` command. (The `S`, `L`, and `V` commands do not display an error code even if an error has occurred while these commands are executed.) The results of executing these commands therefore must be checked by this command.

Syntax

`X[`*`command`*`]`

Explanation

The results of executing the S, L, and V commands are recorded and may be displayed by this command, as follows:

Command	Normal completion	Abnormal completion
`S`	`SAVE COMPLETE`	`SAVE ERROR`
`L`	`LOAD COMPLETE`	`LOAD ERROR`
`V`	`VERIFY OK`	Mis-matched addresses and data are displayed.

Example

```
*XS↵
SAVE COMPLETE
*
```

Command: B

Function

Sets a break point at the specified address, or displays currently set break points. Up to two break points can be set.

Syntax

`B` [*address*]

Explanation

Sets a break point at *address*. Only the two most recently set break points are valid. If no break points are set, 0000 is displayed. A break point cannot be set at address 0000.

If *address* is omitted, currently set break point addresses are displayed.

The PS (program segment) is used and the target address is the specified address plus PS.

Examples

1, 2, 3...

1. `*B3000↵`

2.
   ```
   *B↵
   B=3000 0000
   ```

3.
   ```
   *B5000↵
   *B↵
   B=3000 5000
   ```

Command: N

Function

Cancels all break points.

Syntax

`N`

Explanation

Cancels both break points at once.

Command: G

Function

Begin executing the machine language program at the specified start address.

Syntax

`G`[*start-address*]

Explanation

Usually, the program is executed with break points set in advance. When the program execution has stopped at a break point, the break point is cleared and the current contents of the registers are displayed.

If *start-address* is omitted, the program is executed starting from the current address indicated by the program counter.

The PS (program segment) and program counter (PC) are used and the target address is the specified address plus PS.

The initial values of the stack pointers (SP, SS) for `MON` are set according to `MSET`.

Example

```
*G4000r
R2 R1 R0 V   D I  B  S  Z  F1 A  F0 P  IB C
–  –  –  *  –  –  –  –  –  –  –  –  –  –  –
AW-FEDC BW-0000 CW-0000 DW-0000  SP-0000 BP-0000
IX-0000 IY-0000 PS-0000 DS0-0000 SS-0000 DS1-0000 PC-1234
```

For the flags, `*` indicates 1 (set) and – indicates 0 (reset)

Command: T

Function Executes one step (one instruction) of the machine language program.

Syntax `T`[*address*]

Explanation Executes one instruction at the specified address, and, after execution, disassembles and displays the instruction. Also displays the current contents of the registers.

If *address* is omitted, the instruction at the address currently indicated by the program counter is executed.

The PS (program segment) and program counter (PC) are used and the target address is the specified address plus PS.

Example

```
*T4020↵
```

Command: R

Function Changes the contents of a register or flag, or displays the current contents of all the registers and flags.

Syntax

`R`
`R` *register-name* = *data*
`R` *flag-name* = *flag-state*

Explanation *Register-name* must be one of these names:
`AW`, `BW`, `CW`, `DW`, `SP`, `BP`, `IX`, `IY`, `PS`, `DS0`, `SS`, `DS1`, or `PC`

Data should be a hexadecimal number.

Flag-name must be one of these names:
`R2`, `R1`, `R0`, `V`, `D`, `I`, `B`, `S`, `Z`, `F1`, `A`, `F0`, `P`, `IB`, or `C`

Flag-state should be 0 (reset) or 1 (set).

The PS (program segment) and program counter (PC) are used and the target address is the specified address plus PS.

Examples

1, 2, 3...

1. `*RAW=FEDC↵`
2. `*RV=1↵`
3. `*RS=0↵`
4. `*R↵`

```
R2 R1 R0 V   D I  B  S  Z  F1 A  F0 P  IB C
–  –  –  *  –  –  –  –  –  –  –  –  –  –  –
AW-FEDC BW-0000 CW-0000 DW-0000  SP-0000 BP-0000
IX-0000 IY-0000 PS-0000 DS0-0000 SS-0000 DS1-0000
PC-1234
```

Command: K

Function

Performs addition or subtraction on 4-digit hexadecimal data.

Syntax

`K` *value* + *value*
`K` *value* – *value*

Explanation

Calculates the sum or difference of the two values. Any carry or borrow is ignored.

Example

```
*K1234+5678↵
1234+5678=68AC
```

Command: ESW

Function

Displays or sets the contents of the memory switches in the BASIC Unit, or reads or writes the memory switches of the CPU Unit.

Syntax

`ESW` *switch-no.* . . . . Displays switch settings
`ESW` *switch-no.* = *data* . . . . Sets memory switches
`ESW – R` . . . . Displays memory switches from CPU Unit
`ESW – W` . . . . Write memory switches to CPU Unit

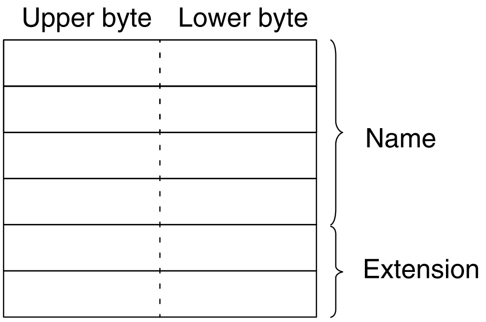
Explanation

- Display

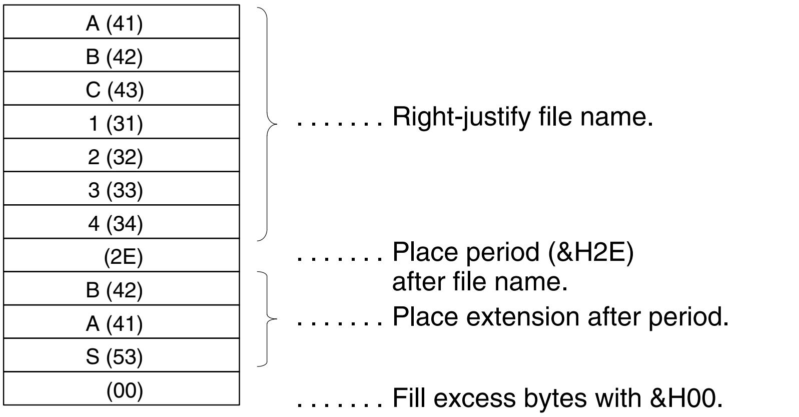
  The switch numbers are as follows:

  1 : System parameters (ESW1)
  2 : Automatic transfer file name (ESW2)
  3 : Terminal, printer ports (ESW3)
  4 : Baud rate for each port (ESW4)
  5 : Terminal specifications (ESW5)
  6 : Cyclic area settings (ESW6) (Groups 1 to 12 displayed.)
  7 : GP-IB setting (ESW7)

- The file name consists of up to eight ASCII characters, a period, and a 3-character extension. The file name must start with an alphanumeric character. The file extension is `BAS`.
- Do not leave any blank characters between the file name and period or period and file extension.



Example: File name `ABC1234.BAS`



.... Right-justify file name.

.... Place period (&H2E) after file name.

.... Place extension after period.

.... Fill excess bytes with &H00. When setting this area with machine language monitor command `ESW2`, excess bytes are automatically filled.

- Setting

  Set switches 1, 3, 4, 5, and 7 as follows:
  `ESW`*n*=*4-digit-hexadecimal-number*
  Set switch 2 as follows:
  `ESW2=`*file-name*`.`*file-extension*
  Set switch 6 as follows:
  `ESW6`–*m* =*dddd–dddd–dddd–dddd*
  (*m* : group no. 1 to 12, *d* : decimal digit)
  Only the contents of the memory in the BASIC Unit are changed. To change the contents of the CPU Unit's memory, write the settings to the CPU Unit (`ESW – W`).

Reading from CPU Unit

The current contents of the memory switches in the CPU Unit are read to the BASIC Unit. The messages displayed at this time are as shown in the table below.

Status	Message
Normal completion	`MEMORY SWITCH READ COMPLETE`
Memory switch information error	`MEMORY SWITCH ERROR`
Memory switch information missing	`MEMORY SWITCH NONE`
Read error	`MEMORY SWITCH FINS ERROR`
Read timeout	`TIMEOUT ERROR`
Error during error logging	`ERROR LOG WRITE ERROR`

Writing to CPU Unit

The contents of the memory switches in the BASIC Unit are written to the CPU Unit. The messages displayed at this time are as shown in the following table:

Status	Message
Normal completion	`MEMORY SWITCH COPY COMPLETE`
Write data error	`MEMORY SWITCH ERROR`
Write error	`MEMORY SWITCH FINS ERROR`
Write timeout	`TIMEOUT ERROR`

Examples

1, 2, 3...

1. `*ESW1=007F↵`
2. `*ESW1↵`
   `007F`
3. `*ESW2=ABCDE123.BAS↵`

4. `*ESW2↵`
   ```
   ABCDE123.BAS
   ```
5. `*ESW3↵`
   ```
   0000
   ```
6. `*ESW6-7=0080-0032-0000-0005↵`
7. `*ESW6↵`

```
0080 1500 0000 0015  0000 0000 0000 0000  0000 0000 0000 0000
0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000
0080 0032 0000 0005  0000 0000 0000 0000  0000 0000 0000 0000
0000 0000 0000 0000  0000 0000 0000 0000  0000 0000 0000 0000
```

Machine Language Mnemonics

The mnemonics accepted by the machine language monitor's assembler conform to those of the BASIC Unit's CPU, the V25 (NEC µPD70322), with slight differences.

Instructions that Cannot be Used

The following instructions are not used because their functions or operands are meaningless for the BASIC Unit, or because they are represented by a different methods:

```
MOVBK              LDM                OUTM
CMPBK              STM                FPO1
CMPM               INM                FPO2
```

Operand Description Restrictions

The operands related to processing outside a segment cannot be used because the capacity of the user program area of the BASIC Unit is limited.

`CALL` *far-proc*
`CALL` *memptr32*
`RETF` *pop-value*
`BR` *far-label*
`BR` *memptr32*
`BR` *short-label*

The operands cannot be abbreviated. The describe the full name of an operand.

`MUL` *reg, imm8*
`MUL` *reg,imm16*

An instruction that can be assembled in more than one way is always assembled in only one way.

`MOV` *reg, reg* ... (pattern of instruction *reg, reg*)
→1000 100W 11reg2 reg1 with direction flag d = 0

Only one instruction can be entered on each line. Use separate lines for multi-opcode instructions. For example, the following entries are illegal:

```
REPC CMPBKW
MOV DS1:[BW],AW
```

Operand Representation Rules

Immediate Values

xxxx: 4-digit hexadecimal number
Words and bytes are identified by the values.

Memory Addressing Modes

```
[BW+IX]     [BW+IY]     [BP+IX]          [BP+IY]
[IX+BW]     [IY+BW]     [IX+BP]          [IY+BP]
[IX]        [IY]        [BW]
```

`[0000]` to `[FFFF]` .... Direct address
`1234[BW+IX]` .... Enter displacement on the left

Examples

```
MOV   [BW+IX],AW
MOV   FF12[BW+IY],AW
MOV   [4321],AW
```

Processing Unit Specifications (Word/Byte)

To specify whether an instruction should operate on a byte or word, use the `BYTE` or `WORD` qualifiers:

Examples

```
TEST1 BYTE [IX],CL
TEST1 WORD [IY],CL
```

Branch Instructions

Labels cannot be used; specify branch addresses.

Examples

```
2000  7502  BNE   2004
2002  8B04  MOV   AW,[IX}
2004  E90900      2010
```

Appendix G
Reserved Words

A

ABS
ACOS
ALARM
AND
APPEND
AS
ASC
ASIN
AUTO

B

BASE
BITOFF
BITON
BREAK

C

CALL
CDBL
CHDIR
CHR $
CINT
CLOSE
CLS
CMD
COM
CONT
COS
CSNG
CVD
CVI
CVS

D

DATA
DATE $
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DIM
DELIM
DELETE

E

EDIT
END
EOF
EQV
ERASE
ERL
ERR
ERR2
ERR3
ERROR
EXIT
EXP

F

FIELD
FINS
FILES
FIX
FN
FOR
FRE

G

GET
GO
GOSUB
GOTO

H

HEX $

I

IEEE
IF
IFC
IMP
INKEY $
INPUT
INPUT $
INPUT @
INSTR
INT
INTR
INTRB
INTRL
INTRR
IRESET
ISET

K

KEY
KILL
KYBD

L

LEFT $
LEN
LENGTH
LET
LFILES
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPRINT
LPRT
LSET

M

MERGE
MESSAGE
MID $
MKD $
MKI $
MKS $
MOD
MON
MSET

N

NAME
NEW
NEXT
NOT

O

OCT $
OFF
ON
OPEN
OPTION
OR

P

PARACT
PAUSE
PC
PEEK
PGEN
PINF
PNAME
POKE
POLL
PPOLL
PPR
PRINT
PRINT @
PUT

R

RANDOMIZE
REN
RBYTE
RDIM
READ
RECEIVE
REM
REN
RENUM
RESTORE
RESUME
RETURN
RIGHT $
RND
ROMLOAD
ROMSAVE
ROMVERIFY
RSET
RUN

S

SAVE
SCRN
SEARCH
SEG
SEND
SENDSIG
SGN
SIGNAL
SIN
SPACE $
SPC
SQR
SRQ
STATUS
STEP
STOP
STR $
STRING $
SWAP

T

TAB
TAN
TASK
THEN
TIME $
TIMER
TO
TROFF
TRON
TWAIT
TIMEOUT

U

USING
USR

V

VAL
VARPTR
VERIFY
VLOAD
VSAVE

W

WAIT
WBYTE
WEND
WHILE
WRITE

X

XOR

Appendix H
Controlling RS-232C Communications Lines

RS-232C communications lines are controlled using the `OPEN` statement as follows:

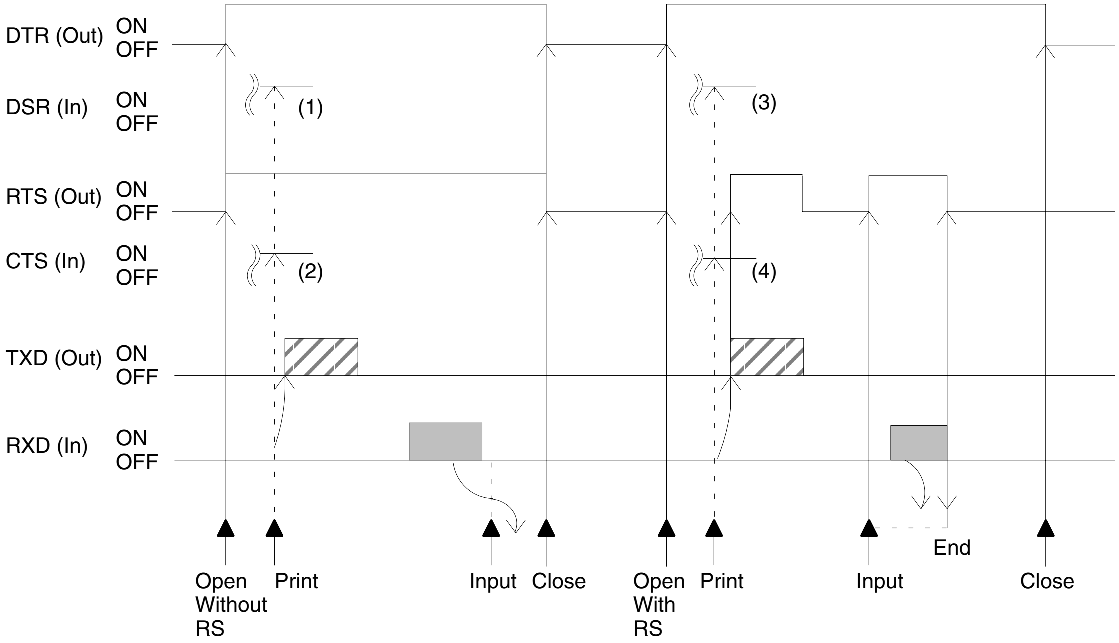
`OPEN" COMn: [`*speed*`] [,` *parity*`] [,` *data_length*`] [,` *stop_bits*`]`
`[, XON/XOFF] [, RS] [, CS`*ml*`] [, DS0] [, LF]" AS#`*file-no.*

RS If RS control is designated, the RTS signal will be turned ON when the I/O command is executed and will be turned OFF otherwise. If RS control is not designated, the RTS signal will remain ON.

CS If CS0 or nothing is designated, there will be no limit to the wait for the ON CTS signal or end of transmission. If a value between CS100 and CS30000 is designated, a wait will be for 100 ms to 30 s maximum.

DS0 If DS0 is designated, the DSR signal will not be checked. If nothing is designated, the DSR signal will be checked.

XON/XOFF If XON/XOFF or nothing is designated, XON/XOFF control will be performed. If XN is designated, XON/XOFF control will not be performed.



(1) and (2) Not checked if DS0 is designated. Checked if nothing is designated (i.e., a RS-232C not ready error will occur if OFF when checked).

(3) and (4) If CS0 or nothing is designated, the signal will be turned ON and an indefinite wait will be made until printing has ended. If a value between CS100 and CS30000 is designated, the signal will be turned ON and a wait of 100 ms to 30 s will be made until printing ends. If the signal turns ON during printing or the time expires, a wait of 60 s will be made. If the 60 s also expire, an I/O timeout error will occur.

Note Communications control using RTS/DTR signals is not possible for the ports set as the terminal and printer ports. This point particularly applies to BCS11/12 Units, for which port 1 is default set to terminal port and port 2 is default set to printer port.

Appendix I
Programming with Windows 95 HyperTerminal

Overview

Previously, an FIT10 Terminal Pack or N88-DISK-BASIC was required to program the BASIC Unit. Now, however, it is possible to program using HyperTerminal and other accessories that have been added to the standard Windows 95 package.

When creating programs using HyperTerminal, only the backspace key can be used in operations on the terminal screen. The cursor keys cannot be used.

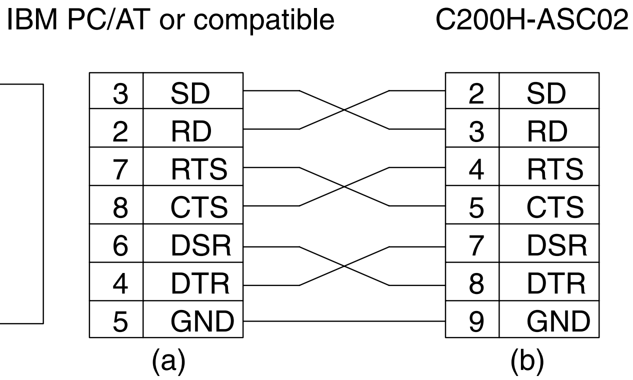
Setup

Connections

Provide a connecting cable for connecting the BASIC Unit to the computer. Connector specifications and the connection configuration are shown below.

Connector

(a) D-sub 9-pin female	Hood:	XM2S-0913
	Connector:	XM2D-0901
(b) D-sub 9-pin male	Hood:	XM2S-0911
	Connector:	XM2A-0901



IBM PC/AT or compatible

3	SD
2	RD
7	RTS
8	CTS
6	DSR
4	DTR
5	GND

(a)

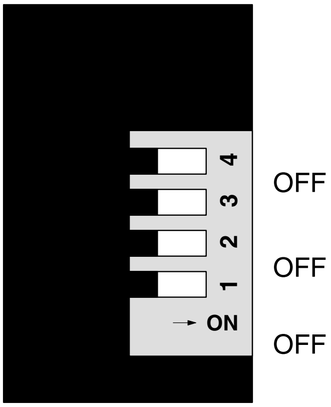
C200H-ASC02

2	SD
3	RD
4	RTS
5	CTS
7	DSR
8	DTR
9	GND

(b)

DIP Switch Settings

Make the settings shown below using the DIP switch in the lower part of the front of the Unit.
Pin 1: Memory protect
Pin 2: Memory switch disabled
Pin 3: ---
Pin 4: ---



HyperTerminal Startup

- Start up HyperTerminal via ***Start/Programs/Accessories***.
- After starting up HyperTerminal, make the settings shown below.

Location Information

Area code: Enter the area code and select ***OK***.

HyperTerminal

A message prompting you to install a modem will be displayed. Select ***No***.

Connection Description

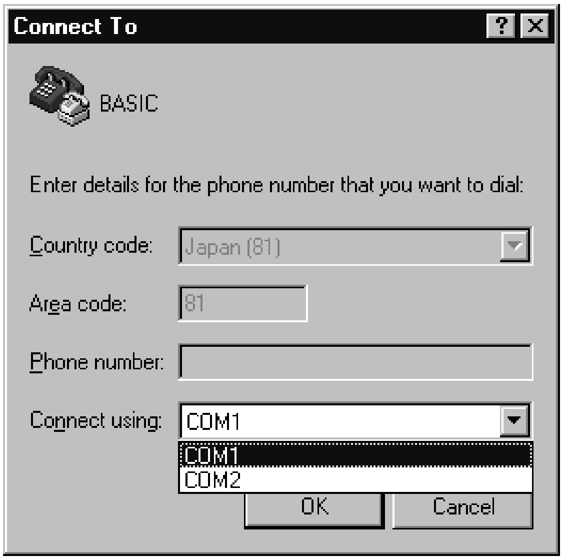
Name: Enter the desired name and select ***OK***.

Connect To

Connect using: Select ***COM1*** and ***OK***.

COM1 Properties

Bits per second: Set to 9,600.
Data bits: Set to 8.
Parity: Set to "`None`".
Stop bits: Set to 1.
Flow control: Set to "`Xon/Xoff`".
Select ***OK***.


Connect To

BASIC

Enter details for the phone number that you want to dial:

Country code: Japan (81)

Area code: 81

Phone number:

Connect using: COM1

- Default settings can be used for all the other settings.
- These settings do not have to be repeated each time you use HyperTerminal. Simply select the icon with the required name.
- If the modem settings have already been made for the computer you are using, only the settings from *Connection Description* onwards are required.

Confirming Connection

Key in Ctrl + X at the computer. The following message will be displayed indicating that connection is complete.

```
BASIC UNIT  Version 1.18 1994/03/25
(C) Copyright OMRON Corporation 1991
ok
```

Memory Switch Settings for BASIC Unit

Set the control method for terminal connection. The backspace key will be enabled by this.

```
Ok
MON↵                          ← Moves to monitor mode
*ESW5=1200↵                   ← VT-100 insert mode
*ESW-W↵                       ← Writes settings to EEPROM
MEMORY SWITCH COPY COMPLETE
*Q↵                           ← Exits MONITOR mode
Ok
■
```

With IBM PC/AT or compatible machines, turn OFF the Scroll Lock key.
This completes the setup.

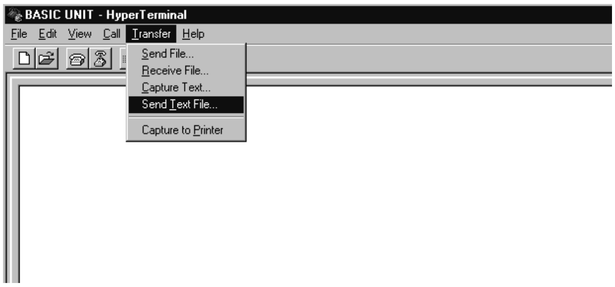
Operation

Creating Programs

Programs are created using text editors, such as Notepad, and are saved as text.

Transferring Programs from the Computer

1, 2, 3...
1. Delete the program currently in the BASIC Unit memory using the NEW command.
2. Transfer the program saved by selecting ***Send Text File...*** from the ***Transfer*** menu as shown below.

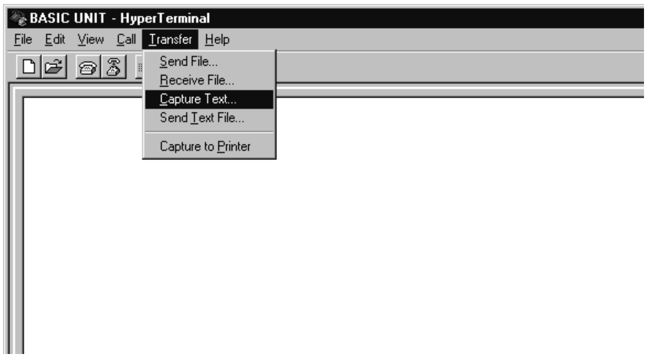


Transferring Programs to the Computer

1, 2, 3...
1. Input the following on the terminal screen. (Do not press the **Enter** Key yet.)

```
SAVE "COM1:"
```

2. Select ***Capture Text*** from the ***Transfer*** menu, and specify the name of the file for saving the program.
3. Press the **Enter** Key.
4. When program transfer has finished, select ***Stop*** in ***Transfer/Capture Text***.



Appendix J
Setting Memory Switches

With BASIC Units, serial port settings and other settings are performed using memory switches. This appendix explains how to make memory switch settings.

The following two methods can be used to set memory switches. Explanations for both methods are given below.

1. From the Terminal

2. Using Support Software (e.g., CVS, SSS)

1. Setting Memory Switches from the Terminal

After the BASIC Unit is connected to the terminal, go into monitor mode as shown below.

```
Ok
MON↵                                - - - - Goes into monitor mode
*
```

Next, make the memory switch settings as shown below.

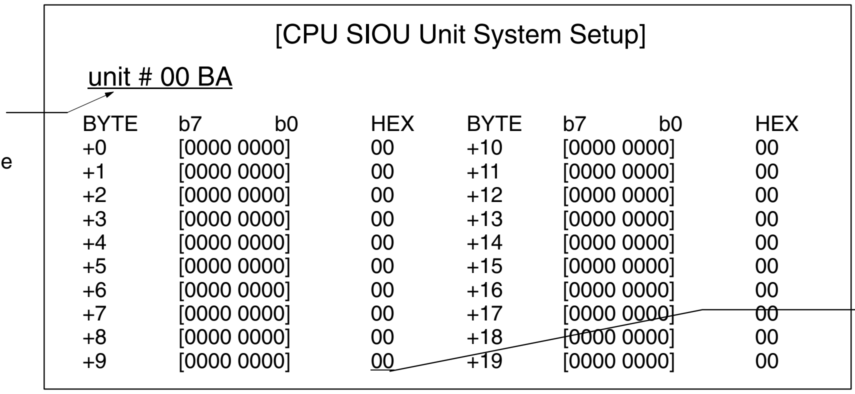
```
*ESW1=1101↵                         - - - - Sets ESW1 to 1101
*ESW7=0125↵                         - - - - Sets ESW7 to 0125
```

When the settings have been completed, write the settings to the Unit and exit monitor mode.

```
*ESW-W↵                             - - - - Writes the memory switch settings
MEMORY SWITCH COPY COMPLETE
*Q↵                                 - - - - Exits monitor mode
```

2. Setting Memory Switches Using Support Software

1. After the Support Software has been connected online to the PC, select "T: CPU SIOU Unit System Setup" from the Communications and CPU Bus Unit Setup Menu. The following screen will be displayed.



```
                 [CPU SIOU Unit System Setup]
  unit # 00 BA
  BYTE   b7        b0    HEX   BYTE   b7        b0    HEX
  +0     [0000 0000]     00    +10    [0000 0000]     00
  +1     [0000 0000]     00    +11    [0000 0000]     00
  +2     [0000 0000]     00    +12    [0000 0000]     00
  +3     [0000 0000]     00    +13    [0000 0000]     00
  +4     [0000 0000]     00    +14    [0000 0000]     00
  +5     [0000 0000]     00    +15    [0000 0000]     00
  +6     [0000 0000]     00    +16    [0000 0000]     00
  +7     [0000 0000]     00    +17    [0000 0000]     00
  +8     [0000 0000]     00    +18    [0000 0000]     00
  +9     [0000 0000]     00    +19    [0000 0000]     00
```

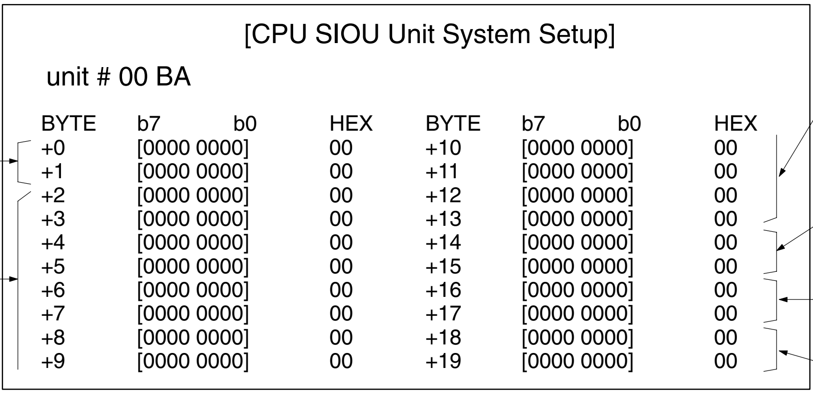
Use the PageUp and PageDown Keys to select the Unit to be set.

Use the PageUp, Page-Down, and Cursor Keys to select the address to to be set.

The setting will be written directly to the CPU Unit.

Press the Shift+Right Cursor Keys to make input in hexadecimal. Press the Shift+Left Cursor Keys to return to decimal input.

2. Perform the memory switch settings from this screen. (For details of the settings, refer to *3-3 Memory Switches.*)



```
                 [CPU SIOU Unit System Setup]
  unit # 00 BA
  BYTE   b7        b0    HEX   BYTE   b7        b0    HEX
  +0     [0000 0000]     00    +10    [0000 0000]     00
  +1     [0000 0000]     00    +11    [0000 0000]     00
  +2     [0000 0000]     00    +12    [0000 0000]     00
  +3     [0000 0000]     00    +13    [0000 0000]     00
  +4     [0000 0000]     00    +14    [0000 0000]     00
  +5     [0000 0000]     00    +15    [0000 0000]     00
  +6     [0000 0000]     00    +16    [0000 0000]     00
  +7     [0000 0000]     00    +17    [0000 0000]     00
  +8     [0000 0000]     00    +18    [0000 0000]     00
  +9     [0000 0000]     00    +19    [0000 0000]     00
```

ESW1 (System parameters)

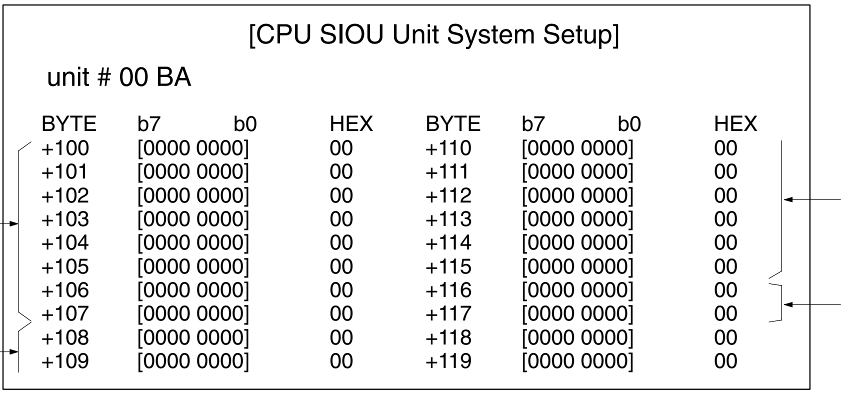
ESW2 (Automatic transfer file name)

ESW2 (Automatic transfer file name)

ESW3 (Selecting printer/ terminal ports)

ESW4 (Baud rate)

ESW5 (Terminal specifications)



```
                 [CPU SIOU Unit System Setup]
  unit # 00 BA
  BYTE   b7        b0    HEX   BYTE   b7        b0    HEX
  +100   [0000 0000]     00    +110   [0000 0000]     00
  +101   [0000 0000]     00    +111   [0000 0000]     00
  +102   [0000 0000]     00    +112   [0000 0000]     00
  +103   [0000 0000]     00    +113   [0000 0000]     00
  +104   [0000 0000]     00    +114   [0000 0000]     00
  +105   [0000 0000]     00    +115   [0000 0000]     00
  +106   [0000 0000]     00    +116   [0000 0000]     00
  +107   [0000 0000]     00    +117   [0000 0000]     00
  +108   [0000 0000]     00    +118   [0000 0000]     00
  +109   [0000 0000]     00    +119   [0000 0000]     00
```

ESW6-11 (Cyclic area settings, input area 5)

ESW6-12 (Cyclic area settings, input area 6)

ESW6-12 (Cyclic area settings, input area 6)

ESW7 (GP-IB setting)

Saving Memory Switch Settings

- The memory switch settings cannot be saved to a file from the CPU SIOU Unit System Setup Screen. If a memory card is installed in the CPU Unit, save the settings to the memory card using the online memory card operations, and then save them to a computer.
- The memory switch settings are saved to the CPU Unit's EEPROM. For this reason, if the CPU Unit is replaced, it is necessary to make the memory switch settings again.

Glossary

active controller The device on a general-purpose interface bus that is currently controlling communications on the bus.

address A number used to identify the location of data or programming instructions in memory or to identify the location of a network or a unit in a network.

address command A command sent to a specific address on a general-purpose interface bus.

advanced instruction An instruction input with a function code that handles data processing operations within ladder diagrams, as opposed to a basic instruction, which makes up the fundamental portion of a ladder diagram.

allocation The process by which the PC assigns certain bits or words in memory for various functions. This includes pairing I/O bits to I/O points on Units.

alphanumeric character An upper- or lower-case letter, digit, or underscore (_). The underscore is considered to be a letter.

analog Something that represents or can process a continuous range of values as opposed to values that can be represented in distinct increments. Something that represents or can process values represented in distinct increments is called digital.

Analog I/O Unit I/O Units that convert I/O between analog and digital values. An Analog Input Unit converts an analog input to a digital value for processing by the PC. An Analog Output Unit converts a digital value to an analog output.

AND A logic operation whereby the result is true if and only if both premises are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.

area See *data area* and *memory area*.

area prefix A one or two letter prefix used to identify a memory area in the PC. All memory areas except the CIO area require prefixes to identify addresses in them.

argument A value passed to a function when the function is called.

arithmetic operator A character indicating to the BASIC Unit that it should perform some sort of calculation; for instance, “+” indicates addition, and “*” indicates multiplication.

array element One part of an array variable. An array element can be another array (for multi-dimensional arrays) or a simple variable (an integer, floating-point, string, etc.)

array subscript An integer expression used to designate an array element for some operation.

array variable A variable which consists of a collection of parts called array elements. Each element can be another array (for multi-dimensional arrays) or a simple variable (an integer, floating-point, string, etc.)

ASCII Short for American Standard Code for Information Interchange. ASCII is used to code characters for output to printers and other external devices.

assembler A program which converts machine-language mnemonics to machine instructions.

asynchronous execution Execution of programs and servicing operations in which program execution and servicing are not synchronized with each other.

Auxiliary Area A PC data area allocated to flags and control bits.

auxiliary bit A bit in the Auxiliary Area.

back-up A copy made of existing data to ensure that the data will not be lost even if the original data is corrupted or erased.

BASIC A common programming language. BASIC Units are programmed in BASIC.

basic instruction A fundamental instruction used in a ladder diagram. See *advanced instruction*.

BASIC Unit A CPU Bus Unit used to run programs in BASIC.

baud rate The data transmission speed between two devices in a system measured in bits per second.

BCD Short for binary-coded decimal.

binary A number system where all numbers are expressed in base 2, i.e., numbers are written using only 0's and 1's. Each group of four binary bits is equivalent to one hexadecimal digit. Binary data in memory is thus often expressed in hexadecimal for convenience.

binary-coded decimal A system used to represent numbers so that every four binary bits is numerically equivalent to one decimal digit.

bit The smallest piece of information that can be represented on a computer. A bit has the value of either zero or one, corresponding to the electrical signals ON and OFF. A bit represents one binary digit. Some bits at particular addresses are allocated to special purposes, such as holding the status of input from external devices, while other bits are available for general use in programming.

bit address The location in memory where a bit of data is stored. A bit address specifies the data area and word that is being addressed as well as the number of the bit within the word.

breakpoint Used during program debugging to mark places where the BASIC Unit should stop executing the program and allow the programmer to check the state of the program's variables.

buffer A temporary storage space for data in a computerized device.

building-block PC A PC that is constructed from individual components, or "building blocks." With building-block PCs, there is no one Unit that is independently identifiable as a PC. The PC is rather a functional assembly of Units.

bus A communications path used to pass data between any of the Units connected to it.

bus link A data link that passed data between two Units across a bus.

byte A unit of data equivalent to 8 bits, i.e., half a word.

central processing unit A device that is capable of storing programs and data, and executing the instructions contained in the programs. In a PC System, the central processing unit executes the program, processes I/O signals, communicates with external devices, etc. The Unit containing the CPU is called the CPU Unit.

channel See *word*.

character code A numeric (usually binary) code used to represent an alphanumeric character.

character constant A character expression which contains no string variables.

character expression An expression involving only character strings, string variables, functions returning character strings, and the "+" operator.

character string A sequence of characters delimited by double quotes (`"`).

checksum A sum transmitted with a data pack in communications. The checksum can be recalculated from the received data to confirm that the data in the transmission has not been corrupted.

CIO Area A memory area used to control I/O and to store and manipulate data. CIO Area addresses do not require prefixes.

command A BASIC Unit instruction which is usually used in immediate mode (e.g. `LIST`, `RUN`, or `NEW`).

command format The syntax required for use in a command and specifying what data is required in what order.

comment statement A statement which is ignored by the BASIC Unit. They may be included in a program to describe the program or to explain how it is supposed to work. Lines beginning with the `REM` instruction are comments, and the single quote character (`'`) begins a comment which extends to the end of the current line.

communications port interrupt An interrupt that occurs when a character is received by one of the communications ports.

constant An input for an operand in which the actual numeric value is specified. Constants can be input for certain operands in place of memory area addresses. Some operands must be input as constants.

control bit A bit in a memory area that is set either through the program or via a Programming Device to achieve a specific purpose, e.g., a Restart Bit is turned ON and OFF to restart a Unit.

control signal A signal sent from the PC to effect the operation of the controlled system.

Control System All of the hardware and software components used to control other devices. A Control System includes the PC System, the PC programs, and all I/O devices that are used to control or obtain feedback from the controlled system.

controlled system The devices that are being controlled by a PC System.

controller A device on a general-purpose interface bus that is capable of controlling communications.

CPU See *central processing unit*.

CPU Bus Unit
A special Unit used with CV-series PCs that mounts to the CPU bus. This connection to the CPU bus enables special data links, data transfers, and processing.

CPU Rack
The main Rack in a building-block PC, the CPU Rack contains the CPU, a Power Supply, and other Units. The CPU Rack, along with the Expansion CPU Rack, provides both an I/O bus and a CPU bus.

C-series PC
Any of the following PCs: C2000H, C1000H, C500, C200H, C40H, C28H, C20H, C60K, C60P, C40K, C40P, C28K, C28P, C20K, C20P, C120, or C20.

CTS signal
A signal used in communications between electronic devices to indicate that the receiver is ready to accept incoming data.

CV Support Software
A programming package run on an IBM PC/AT or compatible to serve as a Programming Device for CV-series PCs.

CV-series PC
Any of the following PCs: CV500, CV1000, CV2000, or CVM1

CVSS
See *CV Support Software*.

cycle
One unit of processing performed by the CPU, including SFC/ladder program execution, peripheral servicing, I/O refreshing, etc. The cycle is called the scan with C-series PCs.

cycle time
The time required to complete one cycle of CPU processing.

cyclic (data) transfer
A transfer of data that occurs at a specific interval.

data area
An area in the PC's memory that is designed to hold a specific type of data.

data link
An automatic data transmission operation that allows PCs or Units within PC to pass data back and forth via common data areas.

data register
A storage location in memory used to hold data. In CV-series PCs, data registers are used with or without index registers to hold data used in indirect addressing.

data transfer
Moving data from one memory location to another, either within the same device or between different devices connected via a communications line or network.

debug
A process by which a draft program is corrected until it operates as intended. Debugging includes both the removal of syntax errors, as well as the fine-tuning of timing and coordination of control operations.

decimal
A number system where numbers are expressed to the base 10. In a PC all data is ultimately stored in binary form, four binary bits are often used to represent one decimal digit, via a system called binary-coded decimal.

decimal integer constant
An integer constant expressed in decimal notation. Such a constant uses only the numerals `0` through `9`.

declarator
A special character added to a variable to specify the type of variable, e.g., a character, a single-precision real number, etc.

decrement
Decreasing a numeric value, usually by 1.

default
A value automatically set by the PC when the user does not specifically set another value. Many devices will assume such default conditions upon the application of power.

destination The location where an instruction places the data on which it is operating, as opposed to the location from which data is taken for use in the instruction. The location from which data is taken is called the source.

destination line The target of a `GOTO` or `GOSUB` statement.

destination variable The variable which is to receive the results of a calculation or operation (the variable in which the results are to be stored).

digit A unit of storage in memory that consists of four bits.

DIP switch Dual in-line package switch, an array of pins in a signal package that is mounted to a circuit board and is used to set operating parameters.

distributed control A automation concept in which control of each portion of an automated system is located near the devices actually being controlled, i.e., control is decentralized and ‘distributed’ over the system. Distributed control is a concept basic to PC Systems.

DM Area A data area used to hold only word data. Words in the DM area cannot be accessed bit by bit.

DM word A word in the DM Area.

double-precision constant A floating-point constant which has at least one of these properties: a trailing hash mark (e.g. `123.45#`); an exponent declared with `D` or `d` instead of `E` or `e` (e.g. `1.2345D2`); or more than 15 digits in the mantissa (e.g. `123.450000000000`).

double-precision variable A variable which can hold a double-precision value.

downloading The process of transferring a program or data from a higher-level or host computer to a lower-level or slave computer. If a Programming Device is involved, the Programming Device is considered the host computer.

DSR signal Data Set Ready signal; sent by a modem to indicate that it is functional.

EEPROM Electrically erasable programmable read-only memory; a type of ROM in which stored data can be erased and reprogrammed. This is accomplished using a special control lead connected to the EEPROM chip and can be done without having to remove the EEPROM chip from the device in which it is mounted.

elapsed-time interrupt An interrupt which occurs after a specified period of time.

electrical noise Random variations of one or more electrical characteristics such as voltage, current, and data, which might interfere with the normal operation of a device.

EM Area Extended Data Memory Area; an area that can be optionally added to certain PCs to enable greater data storage. Functionally, the EM Area operates like the DM Area. Area addresses are prefixes with E and only words can be accessed. The EM Area is separated into multiple banks.

EPROM Erasable programmable read-only memory; a type of ROM in which stored data can be erased, by ultraviolet light or other means, and reprogrammed.

error code A numeric code generated to indicate that an error exists, and something about the nature of the error. Some error codes are generated by the system; others are defined in the program by the operator.

error generation number A number used to identify an error generated by a program.

event (data) transfer A data transfer that is performed in response to an event, e.g., an interrupt signal.

event processing Processing that is performed in response to an event, e.g., an interrupt signal.

executable statement A statement which causes the BASIC Unit to perform some operation, rather than one which changes the way the BASIC Unit interprets the program. (For example, `PRINT` is an executable statement, but `REM` is not.)

Expansion CPU Rack A Rack connected to the CPU Rack to increase the virtual size of the CPU Rack. Units that may be mounted to the CPU Backplane may also be mounted to the Expansion CPU Backplane.

Expansion I/O Rack A Rack used to increase the I/O capacity of a PC. In CV-Series PC, either one Expansion I/O Rack can be connected directly to the CPU or Expansion CPU Rack or multiple Expansion I/O Racks can be connected by using an I/O Control and I/O Interface Units.

expression The translation of a mathematical formula into BASIC notation. For example, the formula for the area of a circle is: $A=\pi r^2$; the BASIC expression to calculate the area of a circle is: `AREA=3.1415*RADIUS^2`.

FA Factory automation.

factory computer A general-purpose computer, usually quite similar to a business computer, that is used in automated factory control.

fatal error An error that stops PC operation and requires correction before operation can continue.

FINS See *CV-mode*.

flag A dedicated bit in memory that is set by the system to indicate some type of operating status. Some flags, such as the carry flag, can also be set by the operator or via the program.

floating-point decimal A decimal number expressed as a number (the mantissa) multiplied by a power of 10, e.g., 0.538 x 10^{-5}.

floating-point format The layout of a single- or double-precision value in memory.

floating-point constant A numeric constant which has a fractional or exponential part.

force reset The process of forcibly turning OFF a bit via a programming device. Bits are usually turned OFF as a result of program execution.

force set The process of forcibly turning ON a bit via a programming device. Bits are usually turned ON as a result of program execution.

frame checksum The results of exclusive ORing all data within a specified calculation range. The frame checksum can be calculated on both the sending and receiving end of a data transfer to confirm that data was transmitted correctly.

function A BASIC Unit instruction which calculates a value based on its arguments and returns the value to the program. The programmer can define new functions with the `DEF FN` statement.

general-purpose interface bus A bus used to connect various devices to a computer.

generation line The line in a program that generates an event, e.g., an interrupt.

global variable A variable which can be accessed from any of the tasks in a program.

GPC An acronym for Graphic Programming Console.

GP-IB An acronym for general-purpose interface bus.

Graphic Programming Console A programming device with advanced programming and debugging capabilities to facilitate PC operation. A Graphic Programming Console is provided with a large display onto which ladder-diagram programs can be written directly in ladder-diagram symbols for input into the PC without conversion to mnemonic form.

handshake line A line in a program or a physical connection between devices used for handshaking.

handshaking The process whereby two devices exchange basic signals to coordinate communications between them.

hexadecimal A number system where all numbers are expressed to the base 16. In a PC all data is ultimately stored in binary form, however, displays and inputs on Programming Devices are often expressed in hexadecimal to simplify operation. Each group of four binary bits is numerically equivalent to one hexadecimal digit.

hexadecimal constant An integer constant expressed in hexadecimal notation. Hexadecimal constants must begin with the characters `&H` or `&h` and contain only hexadecimal digits (numerals `0` through `9` and letters `a` through `f` or `A` through `F`).

host interface An interface that allows communications with a host computer.

Host Link System A system with one or more host computers connected to one or more PCs via Host Link Units or host interfaces so that the host computer can be used to transfer data to and from the PC(s). Host Link Systems enable centralized management and control of PC Systems.

Host Link Unit An interface used to connect a C-series PC to a host computer in a Host Link System.

I/O allocation The process by which the PC assigns certain bits in memory for various functions. This includes pairing I/O bits to I/O points on Units.

I/O Block Either an Input Block or an Output Block. I/O Blocks provide mounting positions for replaceable relays.

I/O Control Unit A Unit mounted to the CPU Rack to monitor and control I/O points on Expansion CPU Racks or Expansion I/O Racks.

I/O delay The delay in time from when a signal is sent to an output to when the status of the output is actually in effect or the delay in time from when the status of an input changes until the signal indicating the change in the status is received.

I/O device A device connected to the I/O terminals on I/O Units, Special I/O Units, etc. I/O devices may be either part of the Control System, if they function to help control other devices, or they may be part of the controlled system.

I/O Interface Unit

A Unit mounted to an Expansion CPU Rack or Expansion I/O Rack to interface the Rack to the CPU Rack.

I/O point

The place at which an input signal enters the PC System, or at which an output signal leaves the PC System. In physical terms, I/O points correspond to terminals or connector pins on a Unit; in terms of programming, an I/O points correspond to I/O bits in the IR area.

I/O refreshing

The process of updating output status sent to external devices so that it agrees with the status of output bits held in memory and of updating input bits in memory so that they agree with the status of inputs from external devices.

I/O response time

The time required for an output signal to be sent from the PC in response to an input signal received from an external device.

I/O Terminal

A Remote I/O Unit connected in a Wired Remote I/O System to provide a limited number of I/O points at one location. There are several types of I/O Terminals.

I/O Unit

The most basic type of Unit mounted to a Backplane. I/O Units include Input Units and Output Units, each of which is available in a range of specifications. I/O Units do not include Special I/O Units, Link Units, etc.

I/O verification error

A error generated by a disagreement between the Units registered in the I/O table and the Units actually mounted to the PC.

I/O word

A word in the CIO area that is allocated to a Unit in the PC System and is used to hold I/O status for that Unit.

IBM PC/AT or compatible

A computer that has similar architecture to, that is logically compatible with, and that can run software designed for an IBM PC/AT computer.

initialize

Part of the startup process whereby some memory areas are cleared, system setup is checked, and default values are set.

input

The signal coming from an external device into the PC. The term input is often used abstractly or collectively to refer to incoming signals.

input bit

A bit in the CIO area that is allocated to hold the status of an input.

Input Block

A Unit used in combination with a Remote Interface to create an I/O Terminal. An Input Block provides mounting positions for replaceable relays. Each relay can be selected according to specific input requirements.

input device

An external device that sends signals into the PC System.

input point

The point at which an input enters the PC System. Input points correspond physically to terminals or connector pins.

input signal

A change in the status of a connection entering the PC. Generally an input signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.

Input Terminal

An I/O Terminal that provides input points.

instruction

A direction given in the program that tells the PC of the action to be carried out, and the data to be used in carrying out the action. Instructions can be used to simply turn a bit ON or OFF, or they can perform much more complex actions, such as converting and/or transferring large blocks of data.

integer constant A numeric value which has a percent sign (%) appended, or an expression containing only integer constants.

integer variable A variable that can hold an integer value.

Intel HEX record Hexadecimal data recorded according to the Intel standard.

Intelligent Signal Processor A control-panel interface used to access and control signals. The Processor is capable of processing the signals according to specifications, and thus the name.

interface An interface is the conceptual boundary between systems or devices and usually involves changes in the way the communicated data is represented. Interface devices such as NSBs perform operations like changing the coding, format, or speed of the data.

interrupt (signal) A signal that stops normal program execution and causes a subroutine to be run or other processing to take place.

Interrupt Input Unit A Rack-mounting Unit used to input external interrupts into a PC System.

interrupt service routine A BASIC subroutine which is called in response to an interrupt.

inter-task communication Communication (transfer of data or status information) between two tasks in a BASIC Unit program.

interval interrupt An interrupt which occurs each time a certain amount of time has elapsed.

IOIF An acronym for I/O Interface Unit.

IOM (Area) A collective memory area containing all of the memory areas that can be accessed by bit, including timer and counter Completion Flags. The IOM Area includes all memory area memory addresses between 0000 and 0FFF.

JIS An acronym for Japanese Industrial Standards.

jump A type of programming where execution moves directly from one point in a program to another, without sequentially executing any instructions in between. Jumps in ladder diagrams are usually conditional on an execution condition; jumps in SFC programs are conditional on the step status and transition condition status before the jump.

keyword A word that has special meaning to the BASIC Unit. Programs cannot use keywords for variable or label names.

label A name attached to a program line for use in `GOTO` and `GOSUB` statements.

least-significant (bit/word) See *rightmost (bit/word)*.

LED Acronym for light-emitting diode; a device used as for indicators or displays.

leftmost (bit/word) The highest numbered bits of a group of bits, generally of an entire word, or the highest numbered words of a group of words. These bits/words are often called most-significant bits/words.

line number An integer which uniquely identifies a line within a program. Line numbers may be used in `GOTO` and `GOSUB` statements.

line One portion of a BASIC program. A line consists of a line number and one or more statements.

link A hardware or software connection formed between two Units. “Link” can refer either to a part of the physical connection between two Units or a software connection created to data existing at another location (i.e., data links).

Link System A system used to connect remote I/O or to connect multiple PCs in a network. Link Systems include the following: SYSMAC BUS Remote I/O Systems, SYSMAC BUS/2 Remote I/O Systems, SYSMAC LINK Systems, Host Link Systems, and SYSMAC NET Link Systems.

Link Unit Any of the Units used to connect a PC to a Link System. These include Remote I/O Units, SYSMAC LINK Units, and SYSMAC NET Link Units.

listener A device on a general-purpose interface bus that is receiving data from another device on the bus.

listener address The addresses on a general-purpose interface bus of a device that is receiving data from another device on the bus.

load The processes of copying data either from an external device or from a storage area to an active portion of the system such as a display buffer. Also, an output device connected to the PC is called a load.

local variable A variable which can only be accessed by the task in which it is declared.

logical expression An expression made up of one or more logical operations, which has “TRUE” or “FALSE” as its value.

logical operation An operation on one or more “TRUE” or “FALSE” values (a Boolean operation), or an operation which returns a “TRUE” or “FALSE” indication.

logical operator A keyword or symbol which instructs the BASIC Unit to perform some calculation that returns a “TRUE” or “FALSE” value.

loop A group of instructions that can be executed more than once in succession (i.e., repeated) depending on an execution condition or bit status.

LSI An acronym for large scale integration.

machine code The binary program code that is actual executed by a CPU.

machine language A programming language in which the program is written directly into machine code.

MCR Unit Magnetic Card Reader Unit.

megabyte A unit of storage equal to one million bytes.

memory area Any of the areas in the PC used to hold data or programs.

memory switch A bit or bits in memory that are used to set operating parameters similar to the way a hardware switch would be.

most-significant (bit/word) See *leftmost (bit/word).*

Motorola S-record A format standardized by the Motorola company to store programs.

MS-DOS An operating system in common use on smaller computers.

multi-dimensional array An array in which more than one subscript is required to access an element.

multidrop configuration A bus configuration in which all devices are connected in series, but across, not through, each device.

multitasked program A program which consists of two or more sub-programs or “tasks” executing concurrently.

multitasking Describes a computer which can run more than one program at a time, or which can give the illusion that several programs are running simultaneously.

my-address The address of a device on a general-purpose interface bus.

nesting Programming one loop within another loop, programming a call to a subroutine within another subroutine, or programming an IF–ELSE programming section within another IF–ELSE section.

network interrupt An interrupt that occurs when data is received on the network interface.

Network Service Board A device with an interface to connect devices other than PCs to a SYSMAC NET Link System.

Network Service Unit A Unit that provides two interfaces to connect peripheral devices to a SYSMAC NET Link System.

noise interference Disturbances in signals caused by electrical noise.

non-executable statement A statement that changes the way the BASIC Unit processes the program, but does not cause the Unit to perform any particular operation. For example, the `REM` statement causes the Unit to ignore the rest of the line.

nonfatal error A hardware or software error that produces a warning but does not stop the PC from operating.

non-volatile variable A variable that is stored in battery-backed memory. Non-volatile variables retain their values even if power to the Unit is turned off.

NOT A logic operation which inverts the status of the operand. For example, AND NOT indicates an AND operation with the opposite of the actual status of the operand bit.

null string A string containing no characters `("")`.

numeric constant A number (integer or floating-point) or a numeric expression containing no variables or function calls.

numeric expression A sequence of numbers, variables, and arithmetic operators that instructs the BASIC Unit to calculate a numeric value.

numeric key interrupt An interrupt that occurs when the user presses one of the numeric keypad keys.

numeric variable A variable that can hold a numeric value.

object code The code that a program is converted to before actual execution. See *source code*.

octal
A number system where all numbers are expressed in base 8, i.e., numbers are written using only numerals 0 through 7.

octal constant
An integer constant expressed in octal notation. Octal constants must begin with `&`, `&O`, or `&o` and contain only octal digits (numerals `0` through `7).`

OFF
The status of an input or output when a signal is said not to be present. The OFF state is generally represented by a low voltage or by non-conductivity, but can be defined as the opposite of either.

OFF delay
The delay between the time when a signal is switched OFF (e.g., by an input device or PC) and the time when the signal reaches a state readable as an OFF signal (i.e., as no signal) by a receiving party (e.g., output device or PC).

offset
A positive or negative value added to a base value such as an address to specify a desired value.

ON
The status of an input or output when a signal is said to be present. The ON state is generally represented by a high voltage or by conductivity, but can be defined as the opposite of either.

ON delay
The delay between the time when an ON signal is initiated (e.g., by an input device or PC) and the time when the signal reaches a state readable as an ON signal by a receiving party (e.g., output device or PC).

operand
The values designated as the data to be used for an instruction. An operand can be input as a constant expressing the actual numeric value to be used or as an address to express the location in memory of the data to be used.

operating error
An error that occurs during actual PC operation as opposed to an initialization error, which occurs before actual operations can begin.

operator
A character that instructs the BASIC Unit to perform some calculation. For example, the "+" character indicates that the BASIC Unit should add two numeric values (or concatenate two strings).

operator priority
Controls the order of evaluation for sub-expressions in a numeric expression. For example, `2+3*4` is interpreted as `2+(3*4)` or 14 (and not `(2+3)*4` or 20), because the operator priority for `*` is higher than that for `+`. Parentheses may be used to change the order in which sub-expressions are evaluated.

OR
A logic operation whereby the result is true if either of two premises is true, or if both are true. In ladder-diagram programming the premises are usually ON/OFF states of bits or the logical combination of such states called execution conditions.

OS
Operating system; the basic software the drives a computer and on which all other software is executed.

output
The signal sent from the PC to an external device. The term output is often used abstractly or collectively to refer to outgoing signals.

Output Block
A Unit used in combination with a Remote Interface to create an I/O Terminal. An Output Block provides mounting positions for replaceable relays. Each relay can be selected according to specific output requirements.

output device
An external device that receives signals from the PC System.

output point The point at which an output leaves the PC System. Output points correspond physically to terminals or connector pins.

output signal A signal being sent to an external device. Generally an output signal is said to exist when, for example, a connection point goes from low to high voltage or from a nonconductive to a conductive state.

Output Terminal An I/O Terminal that provides output points.

overflow The state where the capacity of a data storage location has been exceeded.

overwrite Changing the content of a memory location so that the previous content is lost.

pad byte An extra byte added at the end of a string to make the total number of characters in the string even.

parallel polling A polling method in which all devices in a system are polled at the same time.

parity Adjustment of the number of ON bits in a word or other unit of data so that the total is always an even number or always an odd number. Parity is generally used to check the accuracy of data after being transmitted by confirming that the number of ON bits is still even or still odd.

parity check Checking parity to ensure that transmitted data has not been corrupted.

PC An acronym for Programmable Controller.

PC configuration The arrangement and interconnections of the Units that are put together to form a functional PC.

PC System With building-block PCs, all of the Racks and independent Units connected directly to them up to, but not including the I/O devices. The boundaries of a PC System are the PC and the program in its CPU at the upper end; and the I/O Units, Special I/O Units, Optical I/O Units, Remote Terminals, etc., at the lower end.

PCB An acronym for printed circuit board.

PC Setup A group of operating parameters set in the PC from a Programming Device to control PC operation.

Peripheral Device Devices connected to a PC System to aid in system operation. Peripheral devices include printers, programming devices, external storage media, etc.

peripheral servicing Processing signals to and from peripheral devices, including refreshing, communications processing, interrupts, etc.

PID Unit A Unit designed for PID control.

placeholder A zero that is required to indicate the place value of other digits in a numeral, e.g., the zeros to the right of the decimal point in the following number: 0.0045.

pointer A variable or register which contains the address of some object in memory.

present value The current value registered in a device at any instant during its operation. Present value is abbreviated as PV. The use of this term is generally restricted to timers and counters.

printed circuit board A board onto which electrical circuits are printed for mounting into a computer or electrical device.

program code
The representation of a program used internally by the BASIC Unit.

Programmable Controller
A computerized device that can accept inputs from external devices and generate outputs to external devices according to a program held in memory. Programmable Controllers are used to automate control of external devices. Although single-unit Programmable Controllers are available, building-block Programmable Controllers are constructed from separate components. Such Programmable Controllers are formed only when enough of these separate components are assembled to form a functional assembly, i.e., there is no one individual Unit called a PC.

Programming Console
The simplest form or programming device available for a PC. Programming Consoles are available both as hand-held models and as CPU-mounting models.

Programming Device
A Peripheral Device used to input a program into a PC or to alter or monitor a program already held in the PC. There are dedicated programming devices, such as Programming Consoles, and there are non-dedicated devices, such as a host computer.

PROM
Programmable read-only memory; a type of ROM into which the program or data may be written after manufacture, by a customer, but which is fixed from that time on.

PROM Writer
A peripheral device used to write programs and other data into a ROM for permanent storage and application.

prompt
A message or symbol that appears on a display to request input from the operator.

protocol
The parameters and procedures that are standardized to enable two devices to communicate or to enable a programmer or operator to communicate with a device.

PV
See *present value*.

Rack
An assembly that forms a functional unit in a Rack PC System. A Rack consists of a Backplane and the Units mounted to it. These Units include the Power Supply, CPU, and I/O Units. Racks include CPU Racks, Expansion I/O Racks, and I/O Racks. The CPU Rack is the Rack with the CPU mounted to it. An Expansion I/O Rack is an additional Rack that holds extra I/O Units. An I/O Rack is used in the C2000H Duplex System, because there is no room for any I/O Units on the CPU Rack in this System.

rack number
A number assigned to a Rack according to the order that it is connected to the CPU Rack, with the CPU Rack generally being rack number 0.

Rack PC
A PC that is composed of Units mounted to one or more Racks. This configuration is the most flexible, and most large PCs are Rack PCs. A Rack PC is the opposite of a Package-type PC, which has all of the basic I/O, storage, and control functions built into a single package.

RAM
Random access memory; a data storage media. RAM will not retain data when power is disconnected.

random access file
A file that can be accessed at any desired point, and not only sequentially.

RAS
An acronym for reliability, assurance, safety.

record One block or unit of data in a sequential access file.

refresh The process of updating output status sent to external devices so that it agrees with the status of output bits held in memory and of updating input bits in memory so that they agree with the status of inputs from external devices.

register A special memory location inside the BASIC Unit's CPU.

relative expression A logical expression concerning the magnitudes of two numeric or string expressions (for example, A>B is a relative expression which is TRUE if the value of A is greater than the value of B, and FALSE otherwise).

relative operator A character (e.g. >, <, =) or pair of characters (e.g. >=, <=) used in a relative expression.

relay-based control The forerunner of PCs. In relay-based control, groups of relays are interconnected to form control circuits. In a PC, these are replaced by programmable circuits.

reserved bit A bit that is not available for user application.

reserved word A word in memory that is reserved for a special purpose and cannot be accessed by the user.

reset The process of turning a bit or signal OFF or of changing the present value of a timer or counter to its set value or to zero.

Restart Bit A bit used to restart a Unit mounted to a PC.

restart continuation A process which allows memory and program execution status to be maintained so that PC operation can be restarted from the state it was in when operation was stopped by a power interruption.

retrieve The processes of copying data either from an external device or from a storage area to an active portion of the system such as a display buffer. Also, an output device connected to the PC is called a load.

retry The process whereby a device will re-transmit data which has resulted in an error message from the receiving device.

rightmost (bit/word) The lowest numbered bits of a group of bits, generally of an entire word, or the lowest numbered words of a group of words. These bits/words are often called least-significant bits/words.

rising edge The point where a signal actually changes from an OFF to an ON status.

ROM Read only memory; a type of digital storage that cannot be written to. A ROM chip is manufactured with its program or data already stored in it and can never be changed. However, the program or data can be read as many times as desired.

round-robin In order, completing one item before moving on to the next.

routine A section of a program; often one which may be called by other parts of the program as a subroutine.

row-major form Describes the layout of the elements of an array variable in memory.

RS-232C interface An industry standard for serial communications.

RS-422 interface An industry standard for serial communications.

RTS signal Request To Send: the BASIC Unit can be programmed to assert this signal when it wishes to send data through a communications port.

scan The process used to execute a ladder-diagram program. The program is examined sequentially from start to finish and each instruction is executed in turn based on execution conditions. The scan also includes peripheral processing, I/O refreshing, etc. The scan is called the cycle with CV-series PCs.

scan time The time required for a single scan of a ladder-diagram program.

secondary command A command sent with a listener address to specify the address of another listener or talker.

segment A 64K-byte block of memory beginning on a 16-byte boundary. The BASIC Unit's CPU has several registers that can hold the address of the beginning of a segment.

self diagnosis A process whereby the system checks its own operation and generates a warning or error if an abnormality is discovered.

sequential access file A file that can be read or written only sequential from the beginning to the end.

serial polling A polling method in which each device being polled is polled one at a time in sequence.

series A wiring method in which Units are wired consecutively in a string. In Link Systems wired through Link Adapters, the Units are still functionally wired in series, even though Units are placed on branch lines.

service request A signal from a device requesting that some sort of processing occur.

servicing The process whereby the PC provides data to or receives data from external devices or remote I/O Units, or otherwise handles data transactions for Link Systems.

set The process of turning a bit or signal ON.

set value The value from which a decrementing counter starts counting down or to which an incrementing counter counts up (i.e., the maximum count), or the time from which or for which a timer starts timing. Set value is abbreviated SV.

signal interrupt An interrupt caused by another task activating a `SIGNAL` instruction.

simple variable A non-array variable. Simple variables have only one value and cannot be subscripted.

single-precision constant Any number which is not specifically designated as an integer or double-precision floating point value, or which *is* designated as a single-precision value by a trailing exclamation point (`!`), or a numeric expression containing only integer and single-precision constants.

single-precision variable A variable that can hold a single-precision floating point value.

software error An error that originates in a software program.

software protect A means of protecting data from being changed that uses software as opposed to a physical switch or other hardware setting.

software switch See *memory switch*.

source code The code in which a program is written, e.g., ASCII. Source code must be converted to object code before execution.

Special I/O Unit A Unit that is designed for a specific purpose. Special I/O Units include Position Control Units, High-speed Counter Units, Analog I/O Units, etc.

SRAM Static random access memory; a data storage media.

SRQ See *service request*.

stack A data structure in memory which is maintained automatically by the BASIC Unit's CPU. The stack is used in `GOSUB` and `RETURN` instructions, as well as during interrupts.

statement The smallest complete unit of a BASIC program.

suboperand See *operand*.

subroutine A group of instructions placed separate from the main program and executed only when called from the main program or activated by an interrupt.

subscript An integer expression that designates an element of an array variable.

substitution statement A statement that uses the "=" operator to substitute the value of a second variable for that of the first variable.

SV Abbreviation for set value.

synchronous execution Execution of programs and servicing operations in which program execution and servicing are synchronized so that all servicing operations are executed each time the programs are executed.

syntax The form of a program statement (as opposed to its meaning). For example, the two statements, `LET A=B+B` and `LET A=B*2` use different syntaxes, but have the same meaning.

syntax error An error in the way in which a program is written. Syntax errors can include 'spelling' mistakes (i.e., a function code that does not exist), mistakes in specifying operands within acceptable parameters (e.g., specifying read-only bits as a destination), and mistakes in actual application of instructions (e.g., a call to a subroutine that does not exist).

system configuration The arrangement in which Units in a System are connected. This term refers to the conceptual arrangement and wiring together of all the devices needed to comprise the System. In OMRON terminology, system configuration is used to describe the arrangement and connection of the Units comprising a Control System that includes one or more PCs.

system error An error generated by the system, as opposed to one resulting from execution of an instruction designed to generate an error.

system error message An error message generated by the system, as opposed to one resulting from execution of an instruction designed to generate a message.

system variable A variable that contains information about the system (e.g. the current date and time, or the line number on which the last error occurred).

talker A device on a general-purpose interface bus that is sending data to other devices on the bus.

talker address The addresses on a general-purpose interface bus of a device that is sending data to other devices on the bus.

task A complete sub-unit within a BASIC program. Each task has its own variables, stack, and so on, and is completely independent of any other tasks in the program, although it may use inter-task communication to exchange data with these other tasks. The BASIC Unit can execute several tasks simultaneously.

task block Each task is delimited the `TASK` and `END TASK` statements; all statements between these statements are part of the task block.

task program A program written to perform a task.

terminator The code comprising an asterisk and a carriage return (* CR) which indicates the end of a block of data in communications between devices. Frames within a multi-frame block are separated by delimiters. Also a Unit in a Link System designated as the last Unit on the communications line.

three-line handshaking A handshaking method that uses three communications lines to perform handshaking.

timer A location in memory accessed through a TC bit and used to time down from the timer's set value. Timers are turned ON and reset according to their execution conditions.

timer interrupt An interrupt caused by the BASIC Unit's timer.

TR Area A data area used to store execution conditions so that they can be reloaded later for use with other instructions.

TR bit A bit in the TR Area.

transfer The process of moving data from one location to another within the PC, or between the PC and external devices. When data is transferred, generally a copy of the data is sent to the destination, i.e., the content of the source of the transfer is not changed.

transmission distance The distance that a signal can be transmitted.

UM area The memory area used to hold the active program, i.e., the program that is being currently executed.

uni-line message A message transferred on the control bus using only one signal line.

Unit In OMRON PC terminology, the word Unit is capitalized to indicate any product sold for a PC System. Though most of the names of these products end with the word Unit, not all do, e.g., a Remote Terminal is referred to in a collective sense as a Unit. Context generally makes any limitations of this word clear.

unit address A number used to control network communications. Unit addresses are computed for Units in various ways, e.g., 10 hex is added to the unit number to determine the unit address for a CPU Bus Unit.

unit number A number assigned to some Link Units, Special I/O Units, and CPU Bus Units to facilitate identification when assigning words or other operating parameters.

universal command A command sent to all devices on a general-purpose interface bus.

uploading The process of transferring a program or data from a lower-level or slave computer to a higher-level or host computer. If a Programming Devices is involved, the Programming Device is considered the host computer.

user indicator Indicators on a device that can be controlled by a user, e.g., from a user program being run on the device.

user program A program written by the user as opposed to programs provided with a product.

variable An area of memory in which a value can be stored; also refers to the name used in the program to designate that memory area.

variable-length character string A character string variable which can hold a string of any length (up to a system-defined maximum length).

volatile variable A variable which is not stored in battery-backed memory. Volatile variables lose their contents whenever power to the Unit is turned off.

watchdog timer A timer within the system that ensures that the scan time stays within specified limits. When limits are reached, either warnings are given or PC operation is stopped depending on the particular limit that is reached.

WDT See *watchdog timer*.

wildcard A special character used in a filename or extension to indicate zero or more possible characters.

wire communications A communications method in which signals are sent over wire cable. Although noise resistance and transmission distance can sometimes be a problem with wire communications, they are still the cheapest and the most common, and perfectly adequate for many applications.

word A unit of data storage in memory that consists of 16 bits. All data areas consists of words. Some data areas can be accessed only by words; others, by either words or bits.

word address The location in memory where a word of data is stored. A word address must specify (sometimes by default) the data area and the number of the word that is being addressed.

word allocation The process of assigning I/O words and bits in memory to I/O Units and terminals in a PC System to create an I/O Table.

work area A part of memory containing work words/bits.

work bit A bit in a work word.

work word A word that can be used for data calculation or other manipulation in programming, i.e., a ‘work space’ in memory. A large portion of the IR area is always reserved for work words. Parts of other areas not required for special purposes may also be used as work words.

write protect switch A switch used to write-protect the contents of a storage device, e.g., a floppy disk. If the hole on the upper left of a floppy disk is open, the information on this floppy disk cannot be altered.

write-protect A state in which the contents of a storage device can be read but cannot be altered.

Index

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. W206-E1-04

Revision code

The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	June 1992	Original production
1A	November 1992	**Page 10:** Details of configuration between $18000 and $3FFFF corrected. **Page 25:** Information on cyclic transfers corrected. **Page 27:** Description of manual starting under *Starting Mode Bit (b0)* changed. **Page 57:** Information was added to end of *4-3-1 Preparations*. **Page 179:** Note added to introduction to Appendix. **Pages 179 and 180:** `W` command description rewritten.
2	July 1993	Minor corrections to add CV2000 and CVM1, precautions subsection added to *Section 1*, FINS commands added to *Section 6*, appendix added on controlling RS-232C communications lines, and appendix of reserved words added. **Page 3:** Note added on FINS commands. **Page 10:** Information added. Non-volatile variable and variable areas description changed. **Page 14:** DIP switch settings corrected. **Page 18:** Note added. **Page 21:** Information added to Battery Error Flag, Error Code, and Fatal Error Flag. **Page 26:** CPU Bus Link Transfers fixed; information added to Memory Switches. **Page 30:** Note and caution added and mode definitions added to Terminal Model. **Page 31:** Information added at top of page. **Page 33:** Step 5 removed. **Page 37:** Character variable classification corrected in the top chart. Note was added. **Page 38:** Single-precision data range corrected. **Page 55:** Information added on merging programs. **Page 59:** Note expanded. **Page 61:** Section added on saving/loading PC programs. **Page 64:** Range of numeric data for single-precision real numbers corrected. **Page 73:** Caution added. **Page 81:** Information on interrupt-related instructions rewritten. **Page 82:** Information added on communications port interrupts. **Page 83:** Information added on network and PC interrupts. **Page 84:** Example program altered. **Page 94:** Limit to the number of possible message numbers stated. **Page 96:** Information on PS added to top of page. **Page 97:** `HALT` changed in first two programs; “300B” changed to “300F” under *Run the Program.* **Page 98:** Information added on Memory Cards and EEPROM. **Page 100:** Note added; line 20 modified; and first and next to last machine line modified. **Page 101:** Lines 40, 50, and 70 altered and machine language program altered. **Page 107:** Notes added and “B” line in table corrected. **Page 113:** Information on CTS (transmission monitor) corrected and added. **Page 122:** “IEEE(8)” corrected to “IEEE(7)” in first line under *Functions*. **Pages 129 to 132:** Corrections made to 7, 62, 64, 68, 70, 111, 129, 200, and “Compiler error.” **Page 139:** RAM memory specifications corrected. **Page 144:** Host interface example for CV-series PCs added. **Page 149:** Lower right portion of top diagram corrected. **Page 153:** “Outer connection” part of diagram corrected. **Pages 159 and 160:** Corrections made in lines 120, 170, and 250. **Page 163:** Line 290 corrected. **Pages 165 and 166:** Lines 70, 80, 90, 100, 120, and 130 corrected and ladder diagram added. **Page 167:** Line 160 removed; note added, and 1st step in *File Input/ Output* corrected. **Page 168:** Lines 130 and 160 corrected. **Page 170:** Reserved words added. **Pages 173 to 175:** Information for `COM`, `FINS`, and `PC` corrected. **Page 180:** Example for `A` corrected. **Page 181:** Syntax of `S` corrected. **Page 182:** Syntax of `L` and V and examples for `S` and `L` corrected. **Page 183:** Syntax of `X` corrected; addition made on PS to `B` and `G`, on stack pointers to `G`; example corrected. **Page 184:** PS information added to `T`. **Page 185:** Syntax of `ESW` corrected. **Page 186:** Examples #6 and #7 corrected.
2A	December 1996	*Precautions* added before Section 1. **Page 13:** Send and receive buffer information added to the end of *Execution*. **Page 105:** Point added to the end of *Common Programming Mistakes*.
3	May 2000	Changes were made on the following pages. **All pages:** “PC” and “CPU” changed to “CPU Unit” where appropriate. **Page v:** Changes to symbols and minor changes in wording. **Pages xii, xiii:** Major changes to safety operation. **Pages 21, 121, 201:** Information on RTS/DTR signals added. **Page 30:** Sentence added to define “CPU Bus Unit System Setup.” **Pages 31 to 36:** Graphics/tables added/changed in several places. **Page 87:** Information on interrupts added. **Page 110:** Information on mantissa changed; information added to graphics. **Page 121:** Information on processing time added. **Page 139:** Information on FINS error response codes added. **Page 148:** One line added to second table. **Page 153:** Information on termination resistance added. **Page 196:** Top graphic changed. *Appendix I* and *Appendix J* added**.**

Revision code	Date	Revised content
04	August 2003	Changes were made on the following pages. **Page xii:** Added information on safety precautions for external circuits. **Page xiii:** Added ”Power Supply Units” to application precaution. **Page 24:** Added information on specifying cyclic areas and reading and writing from cyclic areas using the PC READ and PC WRITE instructions. Information on output words also added. **Page 25:** Added information on input words. **Pages 32, 33:** Changed ”+0” to ”+1” in diagram. **Page 34:** Changed ”+14” to ”+15” in top diagram and changed ”+15” to ”+14” in bottom diagram. **Page 35:** Changed ”+16” to ”+17” in top diagram and changed ”+17” to ”+16” in bottom diagram. **Page 36:** Changed ”+18” to ”+19” in top diagram and changed ”+19” to ”+18” in bottom diagram. **Page 39:** Changed ”+116” to ”+117” in top diagram and changed ”+117” to ”+116” in bottom diagram.

OMRON Corporation
FA Systems Division H.Q.
66 Matsumoto
Mishima-city, Shizuoka 411-8511
Japan
Tel: (81)55-977-9181/Fax: (81)55-977-9045

Regional Headquarters

OMRON EUROPE B.V.
Wegalaan 67-69, NL-2132 JD Hoofddorp
The Netherlands
Tel: (31)2356-81-300/Fax: (31)2356-81-388

OMRON ELECTRONICS LLC
1 East Commerce Drive, Schaumburg, IL 60173
U.S.A.
Tel: (1)847-843-7900/Fax: (1)847-843-8568

OMRON ASIA PACIFIC PTE. LTD.
83 Clemenceau Avenue,
#11-01, UE Square,
Singapore 239920
Tel: (65)6835-3011/Fax: (65)6835-2711

OMRON

Authorized Distributor:

Note: Specifications subject to change without notice.

Printed in Japan

Cat. No. W206-E1-04 CV500-BSC11/21/31/41/51/61 BASIC Units OPERATION MANUAL OMRON