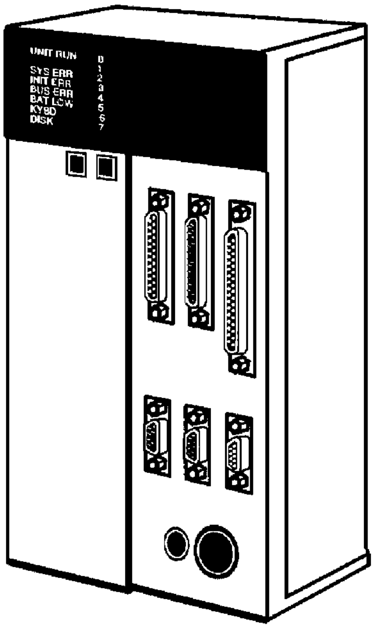
CV500-VP213/217/223/227-E Personal Computer Unit

Technical Manual

Revised October 1995



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 the product.

⚠ **DANGER!** Indicates information that, if not heeded, is likely to result in loss of life or serious injury.

⚠ **WARNING** Indicates information that, if not heeded, could possibly result in loss of life or serious injury.

⚠ **Caution** Indicates information that, if not heeded, could result in relatively serious or minor injury, damage to the product, or faulty operation.

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.

Trademarks

Microsoft C, MS-DOS, Quick BASIC, and Quick C are all registered trademarks of Microsoft Corporation.

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

SECTION 1
Communications and Control Functions 1

1-1 Features 2
1-2 Communications/Control Services 2
1-3 Selecting a Service 4
1-4 Communications/Control Service Details 6

SECTION 2
Software Configuration and Driver Installation 13

2-1 Outline 14
2-2 General Procedures 14
2-3 Program Disk Configuration 15
2-4 Changing the Quick Library Version 16
2-5 Copying Files 16
2-6 Installing Device Drivers 16

SECTION 3
CPU Bus Library 19

3-1 Communication with the PC 20
3-2 Before Using BASIC 24
3-3 BASIC Functions 25
3-4 Before Using C 45
3-5 C Functions 45

SECTION 4
CPU Bus Driver 77

4-1 Introduction 78
4-2 The FINS Format 78
4-3 Using the CPU Bus Driver 82
4-4 CPU Bus Driver Operations List 85
4-5 CPU Bus Driver Operations 85
4-6 FINS Commands Serviced by Drivers 98
4-7 Sample Programs 104
4-8 Measuring CPU Bus Access Performance 117

SECTION 5
FINS Library 121

5-1 Introduction 122
5-2 BASIC Program and FINS Library Structure 122
5-3 Processing Flow 123
5-4 Using the FINS Library 124
5-5 FINS Library Operations 126
5-6 Sample Programs 132

SECTION 6
FINS Driver 137

6-1 Introduction 138
6-2 Processing Flow 138
6-3 Using the FINS Driver 141
6-4 FINS Driver Operations 142
6-5 Sample Programs 147

SECTION 7
RS-232C Communications 155

7-1 RS-232C Port Specifications 156

TABLE OF CONTENTS

7-2 Serial Communications Library 156
7-3 Communications Control Functions 157
7-4 RS-232C Baud Rate Limitations 163
7-5 RS-232C Response Errors 164

SECTION 8
Controlling User Indicators 169

8-1 The User Indicators 170
8-2 Sample Program 171

SECTION 9
Using Error Logs 173

9-1 Error Logs 174
9-2 Writing Error Records 174
9-3 Reading Error Records 176
9-4 Errors at BIOS Startup 177

SECTION 10
Precautions 179

10-1 Operating Environment 180
10-2 Differences from the CV500-VP111/121-E Units 180
10-3 Other Precautions 181

Appendices

A Memory Configuration 183
B I/O Configuration 185
C Interrupts 189
D Error Codes 191
E FINS Commands to the CV500-VP2jj-E 193
F FINS Commands to CV-series PCs 195
G Troubleshooting with FINS Response Codes 197
H PC Memory Configuration 203
I Hard Disk Interface Board Settings 205

Glossary 207

Index 211

Revision History 215

About this Manual:

This manual describes the programming of the CV500-VP2□□-E Personal Computer Units including the installation of system files and the designing of application software, and consists of the sections described below.

Please read this manual completely and be sure you understand the information provided before attempting to program and operate a CV500-VP2□□-E Personal Computer Unit. You must also read the*Personal Computer Unit Operation Manual* and be sure you understand the content provided within it before attempting to program and/or operate a Personal Computer Unit.

Section 1 introduces the communications and control functions that can be used with the Personal Computer Unit and provides instructions on how to use each of the functions.

Section 2 provides general information on the software and explains how to install the CPU Bus Driver and FINS Driver.

Section 3 describes the BASIC and C commands that are used to communicate between the Personal Computer Unit and the local or remote Programmable Controllers.

Section 4 introduces the CPU Bus Driver and describes the FINS commands used in the CPU Bus Driver.

Section 5 describes the FINS library.

Section 6 describes the FINS driver.

Section 7 describes the RS-232C communications functions and how to use them.

Section 8 describes how to use the error log, which records errors that occur during operation of the Unit.

Section 9 describes some important differences between the CV500-VP1□1-E and CV500-VP2□□-E versions of Personal Computer Unit and general precautions when setting up and using the Units.

The **Appendices** provides information on memory configuration, I/O configuration, interrupts, error codes, FINS commands, troubleshooting with FINS response codes, PC memory configuration, and Hard Disk Interface Board settings.

⚠**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.

SECTION 1
Communications and Control Functions

This section introduces the communications and control functions that can be used with the Personal Computer Unit, and provides instructions on how to use each of the functions. Use this section to determine the functions needed for your system configuration.

1-1 Features 2
1-2 Communications/Control Services 2
1-3 Selecting a Service 4
1-3-1 Required Functions 4
1-3-2 Libraries, Drivers, and Programming Languages 5
1-4 Communications/Control Service Details 6
1-4-1 Cyclic Service 6
1-4-2 Event Service 7
1-4-3 CPU Bus Link Service 11

1-1 Features

The Personal Computer Unit's communications/control functions have the following features.

High-speed Communication

The Personal Computer Unit communicates with the PC through the CPU bus, providing high-speed communications compared to an RS-232C or other interfaces. Almost 8K-bytes of data can be transferred at one time depending on the type of communications servicing being used.

Communications with other Hierarchies

The Personal Computer Unit can communicate through the network up to three hierarchies.

Communications with all CPU Bus Units

The Personal Computer Unit can communicate easily with other Personal Computer Units, BASIC Units, and any other CPU Bus Unit within a three-hierarchy network.

Simplified Programs

The CPU Bus Library executes all of the complicated CPU bus communications procedures. All the user needs to do is set the parameters in the function when it is loaded in the program, and the CPU bus can be used automatically.

BASIC and C Libraries are provided. Furthermore, the CPU Bus Driver, FINS Driver, and FINS Library are supported for users who want to use other programming languages or require more precise control of communications.

1-2 Communications/Control Services

The following table shows the three communications/control services that can be used through the CPU bus interface.

No.	Service	Explanation
1	Event service	Communication possible with Programmable Controller and other Units in the local node, as well as with other Programmable Controllers and Units through the network.
2	Cyclic service	Communication possible with the Programmable Controller in the local node.
3	CPU Bus Link service	Communication possible with Programmable Controller and other CPU Bus Units in the local node.

These three services are described briefly below. Refer to *1-4 Communications/ Control Service Details* for more details.

Event Service

Event service communication is accomplished with OMRON's FINS command/ response protocol, and can be used to control other Units and obtain data from other Units. Refer to the *FINS Command Reference Manual (W227)* for details.

Event servicing is the only communications/control service that can be used with the CPU Bus Library. Users programming with only the BASIC or C CPU Bus Library don't need to know the details of the three services. Proceed directly to *Section 2 Software Configuration and Driver Installation* after reading *1-3 Selecting a Service*.

Cyclic Service

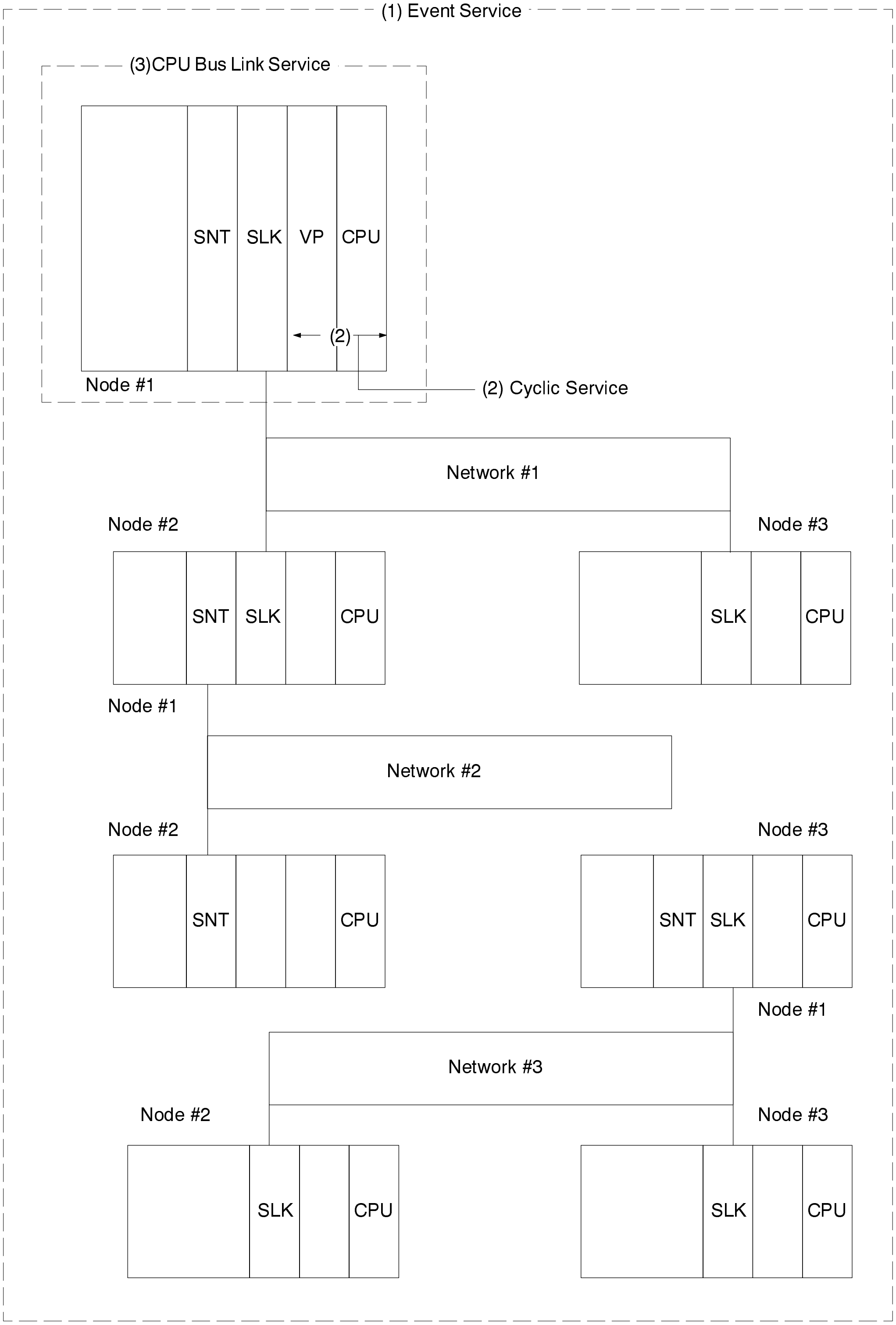
Use the cyclic service for communications (similar to a data link) between the Personal Computer Unit and the local-node PC.

CPU Bus Link Service

This service periodically reads data from other CPU Bus Link Units in the local node; it is useful for monitoring the most recent data or operating status of the CPU Bus Link Units in the local node. The CPU Bus Driver must be used.

Effective Service Range

The effective ranges of the three types of communications and control service are as follows:

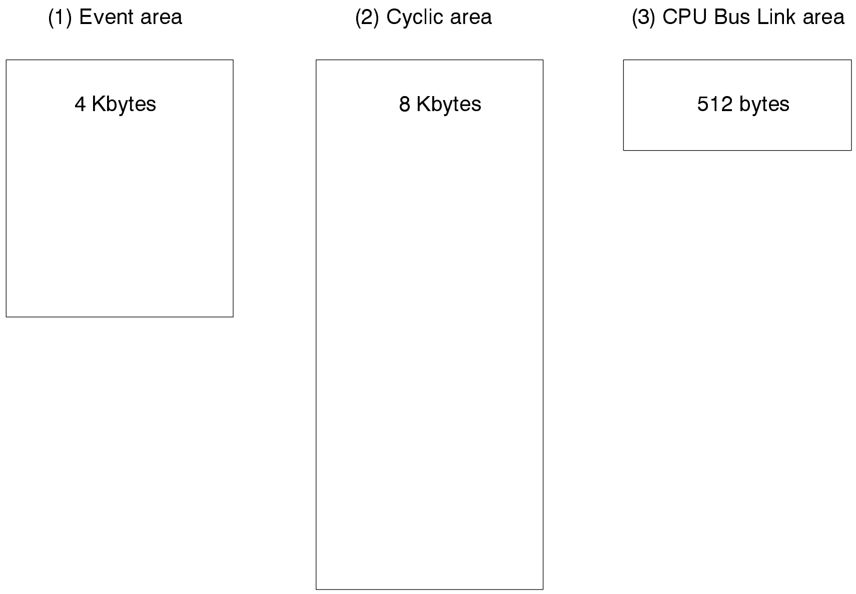


Each of these three types of service uses a dedicated data area within the Personal Computer Unit. Therefore the amount of data that can be processed at one time for any of the services depends on the size of its respective data area.

The cyclic area is the largest, followed in order by the event area and the CPU bus link area. (Refer to the illustration below.) For example, when exchanging large amounts of data with a Programmable Controller in the same node, efficient execution can be achieved by using cyclic service. Likewise, when exchanging data with a CPU Bus Link Unit in the same node, a greater volume of data can be handled by the event service than by the CPU Bus Link service.

The area used by the system is included in these areas. Refer to *1-4 Communications/Control Service Details* for information on the maximum amount of each data area that can be used at one time.

The following diagram shows the relative sizes of each service-dedicated data area of the Personal Computer Unit.



1-3 Selecting a Service

This section provides the information necessary to select the type of communications service most suitable for your system configuration and describes the libraries, drivers, and programming language to used with the selected service.

1-3-1 Required Functions

The following table shows the special features of each service. Select the service that has the most functions needed in your system configuration.

Service	Inter-network communication	Compatible Units	Amount of data/transfer
Event	Up to three hierarchies	All CPU Bus Units	2013 bytes (including FINS header)
Cyclic	Not possible	Only the CPU	3987 words
CPU Bus Link	Not possible	All CPU Bus Units	8 words

Event Service

Other special features of the event service are listed below.

1, 2, 3...

1. This is the only service that can communicate with other networks.
2. Programming is simple with the provided BASIC and C libraries.
3. High-level communications and control can be performed using the CPU Bus Driver, FINS Driver, and FINS Library.

Cyclic Service

Other special features of the cyclic service are listed below.

1, 2, 3...

1. This service allows a simple data-link type of communication between the local CPU and the Personal Computer Unit.
2. Useful for exchanging a large amount of data with the CPU at one time.
3. Communication is not possible with Units other than the CPU.

CPU Bus Link Service

Other special features of the CPU Bus Link service are listed below.

1, 2, 3...

1. Data can be exchanged periodically (every 10 ms) with other CPU Bus Link Units in the same node.
2. Useful for periodically exchanging data between Personal Computer Units.
3. Useful when periodically reading the CPU's system data.
4. Transfers a relatively small amount of data compared to the other services.

CPU Bus Link Service

This service periodically reads data from other CPU Bus Link Units in the local node; it is useful for monitoring the most recent data or operating status of the CPU Bus Link Units in the local node. The CPU Bus Driver must be used.

1-3-2 Libraries, Drivers, and Programming Languages

The following table shows the libraries and drivers required to use each service, and a reference for more details. Refer to *1-4 Communications/Control Service Details* for more details on each service.

Service	Library (language) and Driver	Reference
Event (See note.)	CPU Bus Library (BASIC, C)	Section 3
	CPU Bus Driver	Section 4
	FINS Library (BASIC)	Section 5
	FINS Driver	Section 6
Cyclic	CPU Bus Driver	Section 4
CPU Bus Link	CPU Bus Driver	Section 4

Note The CPU Bus Driver must be installed in the system even when the CPU Bus Library, FINS Library, and FINS Driver are being used, because communications take place through the CPU Bus Driver.

Programming Languages

The following table shows the programming languages that are supported.

Operating System	BASIC	C
DOS	Quick BASIC 4.5 (Microsoft)	MS-C 6.0, MS-C 7.0 (Microsoft)
		Quick C 2.5 (Microsoft)

Use the CPU Bus Driver or FINS Driver when using a programming language other than the ones listed above. Choose a language that can use MS-DOS function calls and 8086 software interrupts.

Services and Libraries for BASIC Users

Users who wish to program in BASIC should use event servicing. Programs can be created easily when the CPU Bus Driver is used.

If the FINS Library is used, programs can be created that use FINS commands for communication and control.

Library or Driver?

If a CPU Bus Library is used programs can be created and communications/control operations can be performed easily. If relatively simple operations are being performed, such as transferring data to and from the CPU, the time required to create programs can be reduced if the CPU Bus Library is used.

On the other hand, the Drivers and FINS Library allow precise and complicated control operations that can't be achieved with the CPU Bus Library.

The CPU Bus Driver and FINS Driver

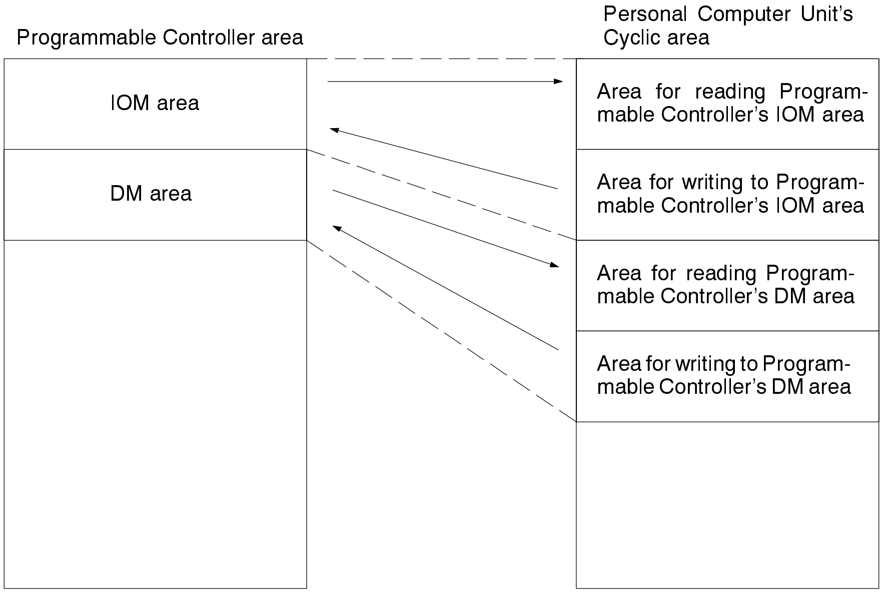
When using event servicing, these two drivers basically perform the same function. The difference is in the headers used for data transmission and reception. When the FINS Driver is used, it isn't necessary to know the data format of the header, so it is easy to use the service through the CPU bus; however, the event service is the only service that can be used when the FINS Driver is used.

Use the information provided in this section to select the service, library, driver, and programming language most suitable for your system, refer to the appropriate sections for more details on these items, and proceed with programming.

1-4 Communications/Control Service Details

1-4-1 Cyclic Service

Cyclic service provides a data link-type connection between the Personal Computer Unit and the local-node Programmable Controller. By means of this service, data from a Programmable Controller in the same node can be handled as if it were data being processed in the Personal Computer Unit. Therefore this service is effective for monitoring and controlling data in the Programmable Controller (such as the contents of data areas).


Programmable Controller area
Personal Computer Unit's Cyclic area
IOM area
DM area
Area for reading Programmable Controller's IOM area
Area for writing to Programmable Controller's IOM area
Area for reading Programmable Controller's DM area
Area for writing to Programmable Controller's DM area

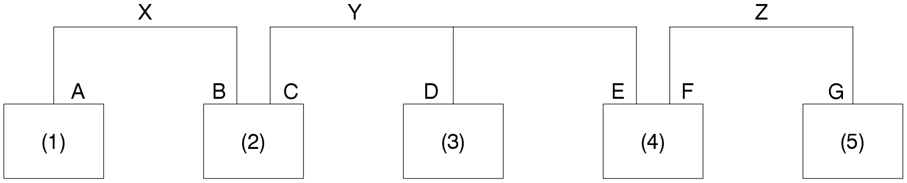
This service can either read Programmable Controller data (in word units) into the cyclic area of the Personal Computer Unit, or write data from the cyclic area of the Personal Computer Unit into the Programmable Controller. Therefore, when using this service, it is necessary to specify the data area address of the Programmable Controller involved in the data transfer, as well as the direction of transfer. (The direction will be either reading or writing, as seen from the Personal Computer Unit.)

Regardless of the direction of data transfer (i.e., Programmable Controller to Personal Computer Unit or Personal Computer Unit to Programmable Controller), the maximum number of data transfer locations that can be specified at one time with this service is six, and the maximum total length of data that can be specified for transfer is 4,096 words. Within that total, 109 words are used by the system, so the maximum number of words that can be utilized by the user is 3,987.

1-4-2 Event Service

Event service can execute communications with Programmable Controllers and other devices both within and outside of the node. It can control those devices and obtain information from them.

By means of setting the routing table, communications can be conducted through the network, and it is possible to control and exchange data with devices in other networks (up to a maximum of three hierarchical levels).



X to Z: Network address
A to G: Network node address

Given that the Personal Computer Unit's node is (1), communications will be possible with devices in node (5) via networks X, Y, and Z. (Communications will also be possible with devices in nodes (1) through (4).

In addition, even if the routing table is not set, communications can still be conducted within the same node. It is therefore possible, within the node, to reference present status such as connection information and to access the Programmable Controller's variable areas.

In general, when event service is used, devices are controlled and information is referenced in the following way. The event service executes a command with a request with respect to the device that it wants to control or reference information from. The response to the request (e.g., the referenced information, or whether the control was properly executed) is then received from the device.

In other words, the event service communications procedure consists of the following steps:

1, 2, 3...

1. Service request command is transmitted to a device that offers a service.
2. Response to the request is received from the device that offers the service.

This communications procedure is the same for other devices as well, so service request commands can also be transmitted from other devices to the Personal Computer Unit. In such cases, the Personal Computer Unit will transmit a response to the request to the device that sent the service request command.

Note Refer to *Appendix H PC Memory Configuration* for details on the PC data areas that can be specified by the user.

Service Request Command/Response

These conform to Programmable Controller FINS commands, and the contents of a command will vary according to the contents of the service requested. In addition, the contents of the response will vary according to the command. Refer to the *FINS Command Reference Manual (W227)* for details.

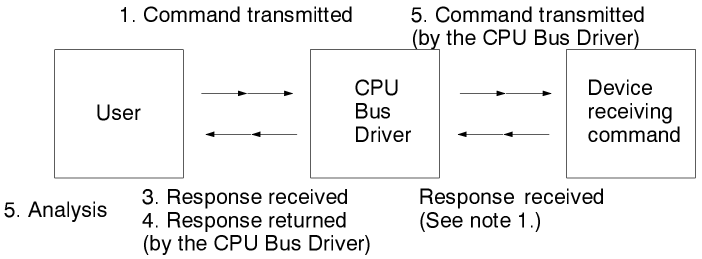
The system, however, will return responses with respect to the commands listed below. Thus there is no need to create responses for these commands.

Commands Eliciting System Response

No.	Command
1	Read Controller Information
2	Read Time Information
3	Write Time Information
4	Loopback Test
5	Read Error Log
6	Clear Error Log

We will now consider this communications processing procedure in more concrete terms.

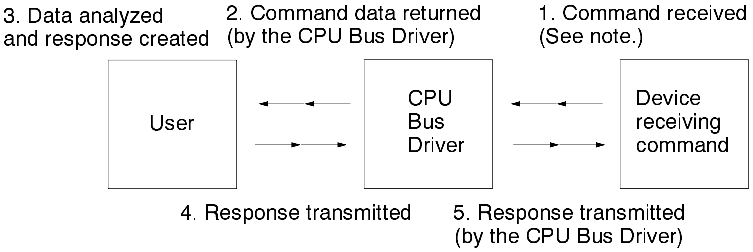
Command from Personal Computer Unit


1. Command transmitted
5. Command transmitted (by the CPU Bus Driver)
User
CPU Bus Driver
Device receiving command
5. Analysis
3. Response received
4. Response returned (by the CPU Bus Driver)
Response received (See note 1.)

1, 2, 3...
1. The user creates a request command corresponding to the service requested, specifies the device offering the service, and requests the CPU Bus Driver to transmit the command.
   At this point, control is returned to the program.
2. The CPU Bus Driver receives the “transmit command” request from the user, and transmits the specified request command to the specified device.
3. In order to receive the response to the command that was transmitted, the user requests “receive response” of the CPU Bus Driver.
4. The CPU Bus Driver returns to the user the response data received from the device to which the command had been transmitted.
5. The user analyzes the contents of the response that is received.

Note
1. The response from the destination device is received in the CPU Bus Driver’s reception buffer whether or not the user receives the response.
2. Steps 1, 3, and 5 are performed by the program.

Command from Another Device


3. Data analyzed and response created
2. Command data returned (by the CPU Bus Driver)
1. Command received (See note.)
User
CPU Bus Driver
Device receiving command
4. Response transmitted
5. Response transmitted (by the CPU Bus Driver)

1, 2, 3...
1. In order to receive the request command from the other device, the user requests “receive command” of the CPU Bus Driver.
2. In response to the “receive command” requested by the user, the CPU Bus Driver returns to the user the command data it has received from the other device.
3. The user analyzes the command data that is received, and creates corresponding response data.
4. Along with creating the response data, the user specifies the device which is the source of the request command, and requests “transmit response” of the CPU Bus Driver.
5. The CPU Bus Driver receives the “transmit response” from the user, and transmits the specified response data to the specified device.

   Note The command reception from the device is executed regardless of the user request. This service request command/response can handle a maximum of 2,048 bytes of data. The system uses 36 bytes, so the maximum amount of data that can be utilized by the user is

2,012 bytes. The beginning of the 2,012 bytes is used for ICF data. Refer to *4-2 The FINS Format* for details.

Event service will be processed as outlined above, and the object of communications can be any device connected through the networks. In other words, the device that is to be the object of communications is determined by the user of the service.

The question, then, will be how to specify the location of that device. This is done by means of the three types of addresses explained below. (The address of the Personal Computer Unit is indicated in the same way.)

Network Address

The network address is the address of the network to which the device belongs. (Network address $00 indicates the same network.)

`$00`	Same network address

Node Address

Within a given network, each device also has a node address. The following codes have special meanings.

`$00`	Same node address
`$FF`	Broadcast to all nodes on specified network (See note 2.)

Unit Address

Within a given node, each device has a unit address, specified by the absolute address.

Example: CPU Bus Unit #0 will have an address of $10.

The following codes have special meanings.

`$00`	Unit address of Programmable Controller
`$10 to $2F`	CPU Bus Units
`$FD`	Peripheral Tools (e.g., FIT)
`$FE`	Communications Units (See note 2.)

Note
1. The actual way in which these addresses are specified will vary according to the drivers and libraries that are used. Please refer to the sections covering drivers and libraries.
   (e.g., SYSMAC NET, SYSMAC LINK)
2. These codes can be used to specify a destination device, but can't be used to specify the same device, i.e., the device can't use these codes to specify itself.

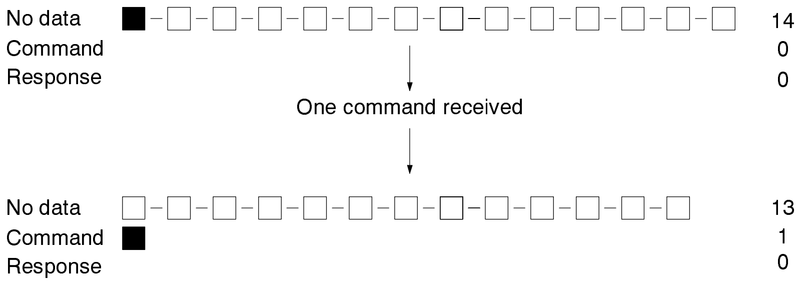
Reception Processing

Reception of commands and responses is executed automatically by the CPU Bus Driver.

Number of Reception Buffers
The CPU Bus Driver has 14 internal reception buffers. (Fourteen is the default, but the number of reception buffers can be specified in the device driver options. Refer to *Section 2 Software Configuration and Driver Installation* for details.)

Among these reception buffers, there is no distinction in terms of whether they are used for commands or responses. Data is received and stored in one of the buffers. For example, if the data received is a command (as shown in the illustration below), the buffer will be used as a command reception buffer.

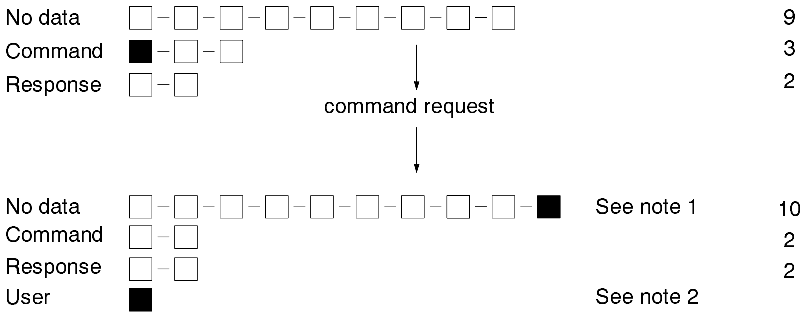
The same will apply if the data received happens to be a response. Therefore, a total of up to 14 commands and responses can be saved internally. If a command or response is received after that point, it cannot be saved and an error will occur. The CPU Bus ERR indicator will light, and the error will be logged.



Managing Reception Buffers

The CPU Bus Driver classifies received data into commands and responses, and saves the data internally until a request is executed for either "receive command" or "receive response." Thus, for example, when a "receive command" request is executed (as shown in the illustration below), the leading data stored internally as command data will be returned to the user.

When even a portion of the received data is read, that data will be discarded. Then when the next request is executed, the next data will be returned. Therefore, when it is preferable that the received data not be lost, it is recommended that the maximum length reception buffer be secured, and that the maximum number of bytes be requested.

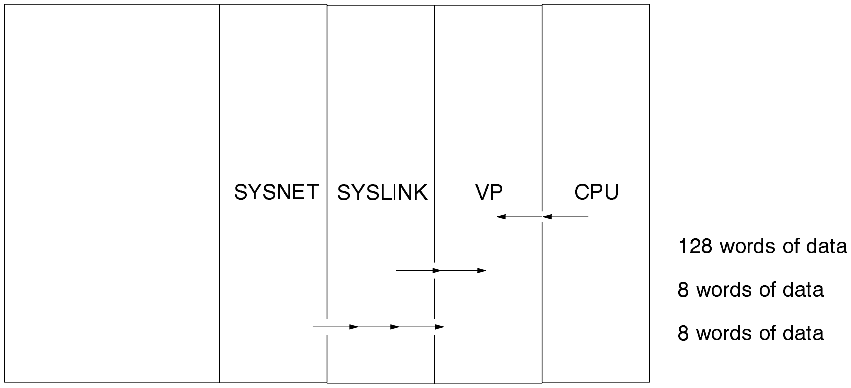


Note

1. After data is copied to the user's specified area, that buffer will be made a "no data" buffer.
2. In response to the command reception request, the leading data of the command will be carried over.

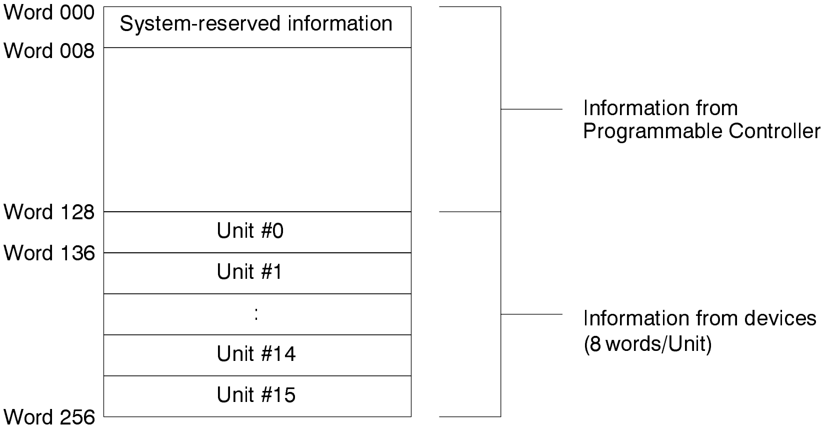
1-4-3 CPU Bus Link Service

CPU bus link service periodically reads data from every device within the same node, so it is effective for regular monitoring of new data and operating status of all the devices in the node. The data read by this service is data from the Programmable Controller and from all the devices within the same node. Altogether, 256 words of data (in word units) can be read.



Within this, a maximum of 128 words (including information reserved for the system) from the Programmable Controller and eight words each for Unit numbers 0 through 15 can be read.

CPU Bus Link Area Contents:


Word 000
Word 008
Word 128
Word 136
Word 256
System-reserved information
Unit #0
Unit #1
:
Unit #14
Unit #15
Information from Programmable Controller
Information from devices (8 words/Unit)

Therefore, when transmitting Personal Computer Unit information to another device, a maximum of eight words can be written into the CPU bus link area and information can be provided.

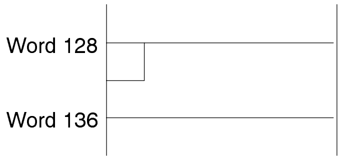
Data can be written only to the portion of the CPU Bus Link Area assigned to the Personal Computer Unit's unit number. Also, the CPU Bus Link must be enabled in the PC Setup (letter I) in order to use this service.

The information obtained from the Programmable Controller (excluding that portion reserved for system use) or from other devices by means of this service depends on the contents of the CPU and devices. Therefore, when using this service to exchange information with other devices, the data contents must be settled in advance.

Regardless of the content of the information from a device, it can be recognized based on that information whether or not that device is currently participating in the CPU bus link service. The leading bit of the data area from each device can be checked, and if the bit is ON it indicates that the device is participating in the CPU bus link service.

The driver will automatically control the status of this flag when the user writes data to the CPU bus link area; the user shouldn't change the flag's status.

Example: Unit #0



If bit D15 of word 128 is ON, then Unit #0 will be in operation.

Information reserved for the system is also received from the Programmable Controller. This information is configured as eight words of data. By reading this data, the user can obtain the status, time, etc., of the Programmable Controller at the time the data is read.

Note Even when the CPU bus link service is stopped, it is still possible to read the system reserved information.

The information reserved for system use is as shown below.

Word 00	PC mode system sw	Refer to Note below.	
Word 01	Minute Second	Minute Second	BCD (0 to 59) BCD (0 to 59)
Word 02	Date Hour	Date Hour	BCD (1 to 31) BCD (0 to 23)
Word 03	Year Month	Year Month	BCD (0 to 99) BCD (1 to 12)
Word 04	– Day	Day	BCD (0 to 6) 0 : Sunday
Word 05	Reserved area 1		
Word 06	Reserved area 2		
Word 07	Reserved area 3		

Note Programmable Controller Mode and System Switch Contents:

15	14	13	12	11	10		8	7	6	5	4	3	2	1	0
15	–	–	12	11		to		7	6	–	4	3	2	1	0

The meaning when each bit is turned ON is as follows:

15:	KEY-SW system protected
12:	Memory card write-protected
11:	UM read-protected
07:	Fatal error occurred
06:	Non-fatal error occurred
04:	PC is operating
03:	RUN mode
02:	MONITOR mode
01:	DEBUG mode
00:	PROGRAM mode

SECTION 2
Software Configuration and Driver Installation

This section provides general information on the software and explains how to install the CPU Bus Driver and FINS Driver.

2-1 Outline 14
2-2 General Procedures 14
2-3 Program Disk Configuration 15
2-4 Changing the Quick Library Version 16
2-5 Copying Files 16
2-6 Installing Device Drivers 16

2-1 Outline

As described in Section 1, the Personal Computer Unit CPU Bus Driver and libraries are composed of the following software.

- CPU Bus Library
- CPU Bus Driver
- FINS Library
- FINS Driver

This section describes the driver installation and file duplication required to use these drivers and libraries, and diagrams the software configuration of the program disk provided with the Personal Computer Unit.

When using Quick Basic 4.2, refer to *2-4 Changing the Quick Library Version* for the procedure required to modify the Quick Library.

The CPU Bus Library, FINS Library, and FINS Driver all communicate through the CPU Bus Driver, so the CPU Bus Driver must be installed even when the other libraries and driver are used.

2-2 General Procedures

The following procedure is generally followed when using Personal Computer Unit's CPU Bus Driver and CPU Bus Library to design application software.

1, 2, 3...

1. Install the CPU Bus Driver and CPU Bus Library.

   Refer to *2-5 Copying Files*.

2. Design the application program.

   The program files to be used will vary according to the language that is to be used for designing the program and the services that are to be employed.

Service/Program file	Language used	Reference
Event service used. CPU Bus Library used.	BASIC or C	*Section 3 CPU Bus Library*
Event service used. FINS Library used.	Design with BASIC.	*Section 5 FINS Library*
Event service used. FINS Driver used.	Design with C or assembler.	*Section 6 FINS Driver*
Event service, Cyclic service, or CPU Bus link service used. CPU driver used.	Design with C or assembler.	*Section 4 CPU Bus Driver*

3. Start the application program.

2-3 Program Disk Configuration

The Personal Computer Unit CPU bus device drivers and libraries are stored on the program disk (3.5-inch 2HD, 1.44 MB format).

For details on the installation disk configuration, refer to *Appendix C* in the *CV Personal Computer Unit Operation Manual (W219)*.

The configuration of the program disk is as follows:

```
Root Directory
│
│  (Device driver)
├─ \DEV ──────┬── RESET.EXE - - - - - - - - - - -   Personal Computer Unit Reset Command
│             ├── SBUS.SYS- - - - - - - - - - - -   CPU Bus Driver
│             └── MEMDSK.386 - - - - - - - - - -    Device driver for MS-Windows
│
│  (RS-232C library directory)
├─ \CSIO ─────────┬── CSIOS.LIB- - - - - - - - - - - - -   Small model library
│                 ├── CSIOC.LIB - - - - - - - - - - - - -  Compact model library
│                 ├── CSIOM.LIB- - - - - - - - - - - - -   Medium model library
│                 └── CSIOL.LIB - - - - - - - - - - - - -  Large model library
│
│  (FINS driver/library directory)
├─ \PRG ──────────┬── DGIOX.COM - - - - - - - - - - - -   FINS driver
│                 ├── PCDGSUB.LIB - - - - - - - - - - -   Quick BASIC library (compiler)
│                 ├── PCDGSUB.QLB - - - - - - - - - - -   Quick BASIC library (interpreter)
│                 ├── TO42.BAT - - - - - - - - - - - - -  Batch file used to convert
│                 │                                        PCDGSUB.QLB to Ver. 4.2.
│                 │     (Sample programs)
│                 └── \SAMPLE
│
│  (CPU Bus Library directory)
└─ \CVLIB ────────┬── \BASIC ──────┬── BASLIB.LIB - - - - - - - - -   Quick BASIC library (compiler)
                  │                ├── BASLIB.QLB - - - - - - - - -   Quick BASIC library (interpreter)
                  │                ├── TO42.BAT - - - - - - - - - -   Batch file used to convert
                  │                │                                   PCDGSUB.QLB to Ver. 4.2.
                  │                │     (Sample programs)
                  │                └── \SAMPLE
                  │
                  └── \C ──────────┬── CLIBS.LIB- - - - - - - - - -   Small model library
                                   ├── CLIBC.LIB - - - - - - - - - -  Compact model library
                                   ├── CLIBM.LIB - - - - - - - - - -  Medium model library
                                   ├── CLIBL.LIB - - - - - - - - - -  Large model library
                                   │
                                   │     (Sample programs)
                                   └── \SAMPLE
```

2-4 Changing the Quick Library Version

The CPU Bus and FINS Quick Libraries are for Quick BASIC 4.5. Use the following commands to modify the libraries when they are used with Quick BASIC 4.2. The batch files used to issue the following commands are contained in each directory.

CPU Bus Library

```
lib baslib *baslib *_pcinit *_pcmsg *_pcrdwr *_subfunc
*intdos * int86 *strcpy *itoa *xtoa *dosret;
link /Q baslib+_pcint+_pcmsg+_pcrdwr+_subfunc+
intdos+int86+strcpy+itoa+xtoa+dosret, baslib.qlb,
nul, bqlb42.lib;
```

CPU Bus Library

```
lib pcdgsub *pcdgsub *int86 *intdos *dosret;
link /Q pcdgsub+int86+intdos+dosret, pcdgsub.qlb, nul,
bqlb42.lib;
```

Precautions

Users who modify the quick libraries should take the following steps:

1, 2, 3...

1. Use Quick BASIC 4.2 for lib, link, and bqlb42.lib.
2. Copy pcdgsub.lib, baslib.lib, and bqlb42.lib into the current directory.

2-5 Copying Files

Before using the Personal Computer Unit's CPU Bus Driver, CPU Bus Library, FINS Driver, and FINS Library, copy the following files from the program disk to the drive that is to be used.

Designing with Quick BASIC

```
PCDGSUB.LIB
PCDGSUB.QLB
BASLIB.LIB
BASLIB.QLB
```

Designing with C or Assembler

Copy the following files when the FINS Driver is used:

```
      DGIOX.COM
      RESET.EXE
```

Copy the following files when CPU Bus Driver is used:

```
      CLIBS.LIB   CLIBM.LIB
      CLIBC.LIB   CLIBL.LIB
```

The CPU Bus Driver (SBUS.SYS) is installed in ROM (Drive E). SBUS.SYS must be copied when SBUS.SYS is to be used in a drive other than Drive E.

Copy the FINS Driver (DGIOX.COM) when the FINS Driver is used. Refer to*2-6 Installing Device Drivers* for details.

2-6 Installing Device Drivers

The CPU Bus Driver (SBUS.SYS) must be installed in the system in order to use the Personal Computer Unit's communications functions. Both the CPU Bus Driver and FINS Driver must be installed when the FINS driver is being used.

Be sure to install the CPU Bus Driver in the system when a system disk is being modified or a new CONFIG.SYS file is being created. Install both the CPU Bus Driver and FINS Driver when the FINS driver is being used.

Installing the CPU Bus Driver

Add the following line to the CONFIG.SYS file to install the CPU Bus Driver.

```
DEVICE=n:\SBUS.SYS /V65 /C3 /B12
```

Drive name

Options

Be sure to insert the SBUS.SYS line after the keyboard driver, otherwise it will be impossible to exit by pressing the ESC Key.

Change the drive name when a SBUS.SYS file other than the one in drive E is used.

Example: When SBUS.SYS is Copied to Drive F and Used.
`DEVICE=F:`<*directory*>`\SBUS.SYS` *option*

Options

The following options can be specified for SBUS.SYS.

`/Vnn` When a direct request is executed, the specification is made with the interrupt signal "`nn`." Specify a number from 60 to 65 for the interrupt signal (interrupt signal 60H to 65H). If a number outside of that range is specified, the direct request processing will not be executed.
Refer to *4-3 Using the PCU Bus Driver* for details on direct requests.
When the FINS Driver, FINS Library, or CPU Bus Library are used, "65" must be specified.

`/Cnn` This option specifies the retry counter for CPU bus I/O area access rights acquisition. For the retry counter, specify a number from 0 to 10. If a number outside of that range is specified, the driver will regard it as "10." If the option is not specified, no retries will be executed.
Increase this setting if communications are stopped by PC Busy errors.

`/Bnn` This option specifies the number of reception buffers (1 to 14) for the event service. The default is 14.
Lowering this number reduces the amount of memory taken up by the CPU Bus Driver.

Installing the FINS Driver

When the FINS Driver is used, the following line must also be added to CONFIG.SYS. (It must be added after the SBUS.SYS specification.)

```
DEVICE=n:\DGIOX.COM
```

(n: Drive name)

Note Be sure to add the FINS Driver after the CPU Bus Driver.

SECTION 3
CPU Bus Library

This section describes the BASIC and C commands that are used to communicate between the Personal Computer Unit and the local or remote Programmable Controllers. These commands enable access to and control of Programmable Controller memory contents, status, and error information.

3-1 Communication with the PC 20

- 3-1-1 CPU Bus Library 20
- 3-1-2 CPU Bus Library Contents 23
- 3-1-3 Disk Configuration 23
- 3-1-4 General Development Procedure 23

3-2 Before Using BASIC 24

- 3-2-1 Functions 24
- 3-2-2 Using Library Functions 24
- 3-2-3 Executing in Interpreter Format 25
- 3-2-4 Executing in Compiler Format 25

3-3 BASIC Functions 25

3-4 Before Using C 45

- 3-4-1 Functions 45
- 3-4-2 Using Library Functions 45

3-5 C Functions 45

3-1 Communication with the PC

The Personal Computer Unit can communicate with Programmable Controllers and CPU Bus Units over Programmable Controller networks by means of the CPU bus interface. A CPU Bus Library, storing function subroutines that can be used with BASIC and C programming languages, is provided for communications via the CPU bus interface.

3-1-1 CPU Bus Library

The features of the CPU Bus Library are described in this section, along with an example of how it can be used.

Features

The CPU Bus Library provides the following features.

High-speed Communications Communications with the Programmable Controller are conducted through the CPU bus. Communications are much faster via the CPU bus than by other means, such as RS-232C. You can transmit approximately 2 KB of information at a time, at a baud rate of 16 Mbps.

Network Communications By specifying network addresses and nodes, you can communicate across three network levels, the any local network and any interconnected network that is not separated by more than one other network.

Easy Programming The complex communications procedures of the CPU bus are all executed by the CPU Bus Library. When using library function in a program, it is only necessary to set the parameters. This makes it simple to use network communications without having specific knowledge of communications procedures.

CPU Bus Unit Communications As long as they are within the three network levels, the Personal Computer Unit can easily communicate not only with other Personal Computer Units but also with BASIC Units and any other CPU Bus Units.

Error Log Access If an error should occur, you can easily access the error log in the Programmable Controller to help troubleshoot the problem.

CPU Bus Communications

The CPU Bus Library has an interface that can be used with BASIC or C. By creating programs with these languages, communications can be easily executed using the CPU bus. An example is shown below of using a program created in BASIC to write data into a Programmable Controller area and then read data back from the same area.

In this program, functions are executed in the following order.

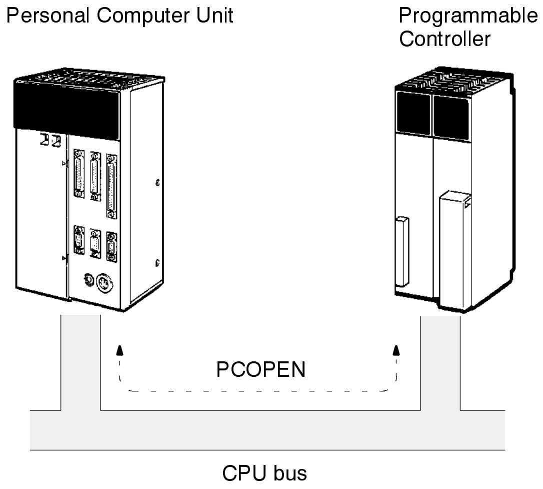
1, 2, 3...

1. **PCOPEN**
   The CPU bus is opened (making it available for use), and preparations are made for communications with the Programmable Controller.

```
PCOPEN (RTN%)
          ↑
```

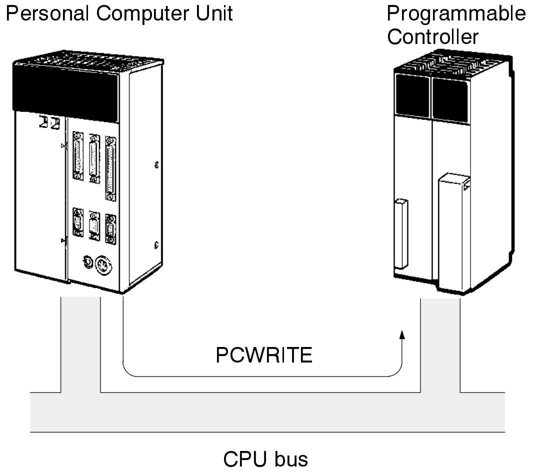
The results of the open are returned.

When using the CPU bus for communications, you must execute this function first.



2. **PCWRITE**
   Data is written into a data area of the Programmable Controller. The data area location into which the data is to be written and the data format can be specified by means of parameters (e.g., SUBFMT $). In this case, two words are written to D00000.

```
NE%=0                         : Network address
NO%=0                         : Node address
VALUE$ (1) =”1001”            : Write data 1
VALUE$ (2) =”2002”            : Write data 2
SUBFMT$ =” @D,0,2,$2I”        : Data area location
                                Size to be written
                                Data format
PCWRITE (NE%,NO%,SUBFMT$,VALUE$ (1),RTN%)
                                     ↑
                              The results of the write are returned.
```

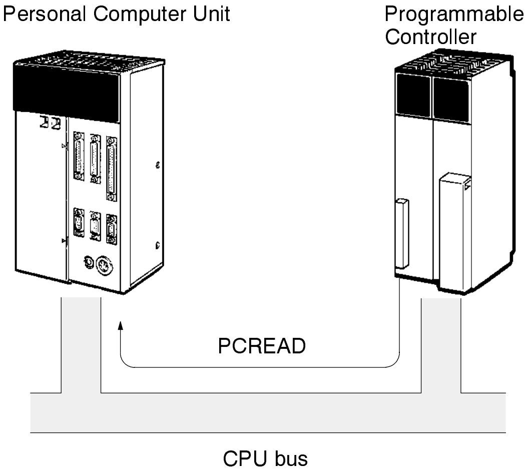


3. **PCREAD**
   Data is read from the data area of the Programmable Controller. The data area location from which the data is to be read and the data format can be specified by means of parameters (e.g., SUBFMT $).

```
NE%=0                             : Network address
NO%=0                             : Node address
VALUE$ (1) ="     "               :
VALUE$ (2) ="     "               :
SUBFMT$ =" @D,0,2,$2I"            : Data area location
                                    Size to be read
                                    Data format
PCREAD (NE%,NO%,SUBFMT$,VALUE$ (1),RTN%)
```

The results of the read processing are returned.

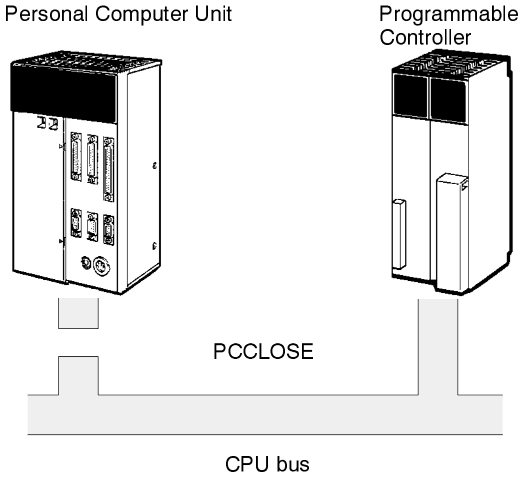
Data is returned.



4. **PCCLOSE**
   The CPU bus is closed. When you have finished using the library, you must execute this function.

```
PCCLOSE (RTN%)
```

The results of the close are returned.



CPU Bus Library Functions

The following functions are included in the CPU Bus Library.

Name	Operation
PCOPEN	Opens the CPU bus.
PCCLOSE	Closes the CPU bus.
PCREAD	Reads data from a Programmable Controller data area.
PCWRITE	Writes data into a Programmable Controller data area.
PCMSRD	Receives a message from another Unit.
PCMSWR	Sends a message to another Unit.
PCSTAT	Accesses and controls Programmable Controller status.
PCMODE	Changes Programmable Controller operating modes.

3-1-2 CPU Bus Library Contents

The CPU Bus Library is configured of the software shown below. Communications with the Programmable Controller can be executed by calling external functions from programs created with these languages.

CPU Bus BASIC Library

The CPU bus BASIC library can be used with Microsoft's Quick BASIC 4.5.

CPU Bus C Library

The CPU bus C library can be used with Microsoft's C compilers (MS-C 6.0, 7.0) There are four types, according to memory model.

3-1-3 Disk Configuration

The CPU Bus Library is stored in directory \CVLIB of PROGRAM DISK. The file configuration is as follows:

`\CVLIB\C`

`CLIBC.LIB`	Library for MS-C (compact model)
`CLIBS.LIB`	Library for MS-C (small model)
`CLIBM.LIB`	Library for MS-C (medium model)
`CLIBL.LIB`	Library for MS-C (large model)

`\CVLIB\BASIC`

`BASLIB.LIB`	Library for Quick BASIC (for compiler)
`BASLIB.QLB`	Library for Quick BASIC (for interpreter)

3-1-4 General Development Procedure

The general procedure for developing a program using the CPU Bus Library is as follows:

1, 2, 3...

1. Copy the necessary file(s).
   The file(s) that is required will vary according to the programming language that is to be used for program development. Use the COPY command to copy the appropriate file(s) to the drive that is to be used for development.

   Developing with MS-C or assembler:
   CLIBC.LIB (compact model)
   CLIBS.LIB (small model)
   CLIBM.LIB (medium model)
   CLIBL.LIB (large model)

   Developing with Quick BASIC:
   BASLIB.LIB (when developing with compiler)
   BASLIB.QLB (when developing with interpreter)
2. Set the CPU Bus Driver into the MS-DOS system.
   The CPU Bus Driver (SBUS.SYS) is already in Drive F in the Personal Computer Unit. When newly creating the MS-DOS system file, CONFIG.SYS,

you must insert the following line to install the CPU Bus Driver (SBUS.SYS). (When starting from the built-in ROM, F:\CONFIG.SYS is used.)

Refer to *2-6 Installing Device Drivers* for details on the parameters.

```
DEVICE = E:\SBUS.SYS /V65
```

The CPU Bus Driver is installed in the Personal Computer Unit's built-in ROM Disk beforehand.

The "/V65" is for system use and must be added.

3. Create the programs.
   For further instructions on using libraries, refer to the remainder of this manual.

3-2 Before Using BASIC

This section will explain how to use the CPU Bus Library for BASIC. The version of BASIC that can use the CPU Bus Library is Quick BASIC.

3-2-1 Functions

The following table lists the available functions.

Name	Operation	Reference
PCOPEN	Opens the CPU bus.	p. 26
PCMSRD	Receives a message from another Unit.	p. 26
PCMSWR	Sends a message to another Unit.	p. 29
PCREAD	Reads data from a Programmable Controller data area.	p. 31
PCWRITE	Writes data to a Programmable Controller data area.	p. 35
PCSTAT	Accesses and controls Programmable Controller status.	p. 40
PCMODE	Changes the Programmable Controller operating modes.	p. 43
PCCLOSE	Closes the CPU bus.	p. 44

3-2-2 Using Library Functions

To use library functions, program according to the following procedure. With Quick BASIC, the library itself cannot be loaded at the source level.

1, 2, 3...

1. Specify a library function. (Program example 1)
   Specify a function with the following statement.
   DECLARE SUB <function name> CDECL ALIAS "<actual function name>"
   You can declare any function name. The actual function name is the function name in the library.
2. Use CALLS to call the library function. (Program example 2)

   Program Examples

```
DECLARE SUB PCOPEN CDECL ALIAS "_b_pcopen"        (1)
DECLARE SUB PCMSRD CDECL ALIAS "_b_pcmsrd"
DECLARE SUB PCMSWR CDECL ALIAS "_b_pcmswr"
DECLARE SUB PCREAD CDECL ALIAS "_b_pcread"
DECLARE SUB PCWRITE CDECL ALIAS "_b_pcwrite"
DECLARE SUB PCSTAT CDECL ALIAS "_b_pcstat"
DECLARE SUB PCMODE CDECL ALIAS "_b_pcmode"
DECLARE SUB PCCLOSE CDECL ALIAS "_b_pcclose"
      .
      .
CALLS PCOPEN (RTN%)                               (2)
      .
      .
```

3-2-3 Executing in Interpreter Format

The procedure for executing an existing program with the BASIC interpreter is as follows:

1, 2, 3...

1. Input `QB/I BASLIB` and then press the Enter Key. Quick BASIC will start BASLIB.
2. Load and execute the program.

Note The QuickBASIC library was created using QuickBASIC 4.2. If you are programming the Personal Computer Unit with any other version, remake baslib.qlb using the following command lines.

```
lib baslib *baslib *_pcinit *_pcmsg *_pcrdwr *_subfunc
*intdos *int86 *strcpy *itoa *xtoa *dosret;

link/Q baslib+_pcinit+_pcmsg+_pcrdwr+_
subfunc+intdos+int86+strcpy+itoa+xtoa+dosret,baslib.qlb,
nul,bqlb45.lib;
```

You will also need to create a new library for each version corresponding to the bqlb45.lib portion. For version 4.2, this file would be bqlb42.lib.

3-2-4 Executing in Compiler Format

The procedure for first compiling and then executing an existing program is as follows:

1, 2, 3...

1. Input `BC <program name>.BAS /O;` and then press the Enter Key. The program will be compiled and an object file will be created. If you add /D after the /O, you can abort program execution in progress by means of Crtl-C.
2. Input `LINK/EX/NOE <program name> + BASLIB.LIB <program name>.EXE,NUL,;` and then press the Enter Key. The BASIC library will be linked to the object file, and an imperative command will be created.
3. Execute EXE, the command that was created.

3-3 BASIC Functions

This section will explain the functions available in the BASIC library.

Headings

Each function will be covered in terms of the following headings.

Purpose An outline of the function's operation will be provided.

Specifiers When functions are defined with Quick BASIC, the specifiers are the names, in the library, of the functions that are to be used.

Format This is the format for functions used in a program. Function names used in the format sections are temporary names defined for the functions. It is not necessary to use these names.

Parameters The meanings of the parameters used in the function's format are given. When the word "input" is enclosed in parentheses next to a given parameter, it will indicate that a value for the parameter is to be input by the user. Likewise, "output" will indicate that the function will return a value for the parameter. A chart will show each parameter's format, range, and type.

Comments Parameter contents and setting precautions will be explained.

Returned Values The meanings of the values returned by the functions will be given. Based on these values you can determine whether functions have executed correctly or whether errors have been generated.

Related Functions This provides the names of other functions related to the function being described. You can refer to the other functions in order to gain a better understanding of the one in question.

Program Examples

One or more examples will be given of programs in Quick BASIC. Program files are kept in directory \CVLIB\BASIC\SAMPLE on the program disk.

PCOPEN CPU BUS OPEN

Purpose

This function opens the CPU bus so that it can be used.

Specifiers

Operation number: 00H (0)
Library function name: _b_pcopen

Format

`PCOPEN (RTN%)`

Parameters

RTN%: Returned value (output)

Parameters	Format	Contents
RTN%	Integer	After execution of the function, the results will be entered as an integer.

Comments

Must be executed first to use the CPU bus.

Returned Values

RTN%	Meaning
0	Ended normally.
1	CPU Bus Driver is not installed.
2	Bus is already open.

Related Functions

PCCLOSE

Program Examples

In this example, the CPU Bus Library is opened and closed.

```
'********************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM               *
'*                  QUICK BASIC                         *
'********************************************************
'
        DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"
        DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

        CALLS pcopen(rtn%)
        IF rtn% <> 0 THEN
               PRINT "pcopen error : rtn = "; rtn%
               GOTO finish
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT "Exit sample program."
        END
```

PCMSRD RECEIVE MESSAGE

Purpose

With this function, messages can be received from other Units.

Specifiers

Operation number: 01H (1)
Library function name: _b_pcmsrd

Format

`PCMSRD (NE%,NO%,UN%,CT%,VALUE$,DSZ%,T%,RTN%)`

Parameters

NE%	:	Source network address (output)
NO%	:	Source node address (output)
UN%	:	Source unit address (output)
CT%	:	Number of actual reception bytes (output)
VALUE$	:	Reception buffer (input/output)
DSZ%	:	Number of bytes requested for reception (input)
T%	:	Timer value (input)
RTN%	:	Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note 1)
UN%	Integer	Absolute address (see note 2)
CT%	Integer	Length of message actually received (see note 1)
VALUE$	String	Space for the number of bytes specified by DSZ%
DSZ%	Integer	Maximum length for message received
T%	Integer	0 to 32767 (Unit: x110 ms)
RTN%	Integer	Status entered as integer after execution of function.

Note

1. The maximum length varies according to the type of network through which the message is transmitted.
2. For CPU Bus Units, the value will be offset by 10 hexadecimal. For example, the UN% for CPU Bus Unit #4 will be &H14.

Comments

When a message is received, the network address, node address, and unit address of the Unit that sent the message will be entered respectively in NE%, NO%, and UN%, and the message length will be entered in CT%.

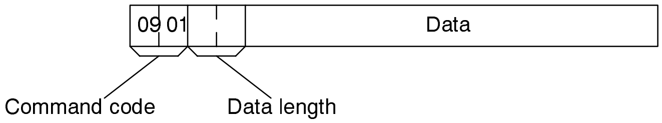
Dummy data (such as spaces) for the number of bytes requested for reception (DSZ%) must be assigned in advance for the variable VALUE$, and the memory area for the received data must be available. If it is not available, an error will be generated. The maximum setting for DSZ% is 538 bytes.

For the variable T%, assign integer values from 0 to 32767 (in units of 110 ms) for the reception waiting time. If “0” is specified, there will be no waiting time.

This function will be ended once a single message of any length has been received, even if no message for the number of bytes requested for reception (DSZ%) is received. In addition, if a message of a size exceeding the DSZ% is received, the DSZ% portion of the message will be returned to the user and the remainder will be discarded.

This function uses command 0901 of the FINS commands. When this function is executed, the Unit will standby for a message that has command code 0901; all other commands will be disregarded until the message with command code 0901 is received.

The following diagram shows the format for FINS command 0901. Use this format when transmitting from outside the Personal Computer Unit.



The following example shows the command format needed to send “ABCDEF.”

09	01	00	06	A	B	C	D	E	F

Return the following response to the message source if the Personal Computer Unit received the message normally.

09	01	00	00

Returned Values

RTN%	Meaning
0	Ended normally.
1	Library is not open.
3	Programmable Controller or Repeater is busy. Try again.
5	There is a parameter error.
6	Message cannot be received from specified Unit.
8	There is an error in the routing tables.

Related Functions

PCMSWR

Program Examples

In this example, a 20-byte message is received from another Unit.

```
’*******************************************************
’*           BASIC LIBRARY SAMPLE PROGRAM              *
’*                  QUICK BASIC                        *
’*******************************************************
’
        DECLARE SUB pcopen CDECL ALIAS ”_b_pcopen”
        DECLARE SUB pcmsrd CDECL ALIAS ”_b_pcmsrd”
        DECLARE SUB pcclose CDECL ALIAS ”_b_pcclose”

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT ”pcopen error : rtn = ”; rtn%
               GOTO finish
        END IF

        dsz% = 20                ’Number of bytes for reception
        time% = 100              ’Timer value
        d$ = STRING$(20, ” ”)  ’Reception buffer
        CALLS pcmsrd(ne%,no%,un%,ct%,d$,dsz%,time%,rtn%)
        IF rtn%<>0 THEN
               PRINT ”pcmsrd error : rtn = ”; rtn%
        ELSE
              PRINT ”Source network address: ”; ne%
              PRINT ”Source node address: ”; no%
               PRINT ”Source unit address: ”; un%
                PRINT ”Number of bytes for reception: ”;
ct%
               PRINT ”Reception message: ”; d$
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT ”Sample program finished”
        END
```

PCMSWR SEND MESSAGE

Purpose

With this function, messages can be transmitted to specified Units.

Specifiers

Operation number: 02H (2)
Library function name: _b_pcmswr

Format

```
PCMSWR (NE%,NO%,UN%,CT%,VALUE$,RTN%)
```

Parameters

NE%	:	Destination network address (input)
NO%	:	Destination node address (input)
UN%	:	Destination unit address (input)
CT%	:	Number of bytes for transmission (input)
VALUE$	:	Reception buffer (input)
RTN%	:	Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note 1)
UN%	Integer	Specifies absolute address (see note 2)
CT%	Integer	Length of transmission message (1 to 538)
VALUE$	String	Transmission string of length specified by CT%
RTN%	Integer	Status entered as integer after execution of function.

Note

1. The maximum length varies according to the type of network through which the message is transmitted.
2. For CPU Bus Units, the unit number will be offset by 10 hexadecimal. For example, the UN% for CPU Bus Unit #4 will be &H14.

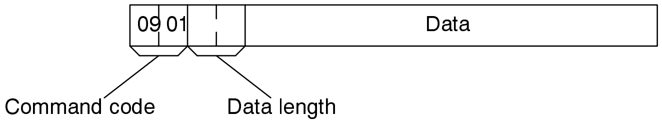
Comments

Data for the number of bytes requested for reception (CT%) will be assigned in advance for the variable VALUE$. If there is no transmission data, an error will be generated.

This function will not be ended until the message has been completely received at the destination (i.e., until a response has been returned from the destination). The function may end, however, due to the Repeater being busy.

This function uses command 0901 of the FINS commands, and messages can be transmitted to BASIC Units.

This function uses command 0901 of the FINS commands. When executing this function, send the FINS message with the format shown in the following diagram.



The following example shows the command format needed to send “ABCDEF.”

09	01	00	06	A	B	C	D	E	F

When the destination Unit isn’t a Personal Computer Unit and the message is received normally, return the following response addressed to the Personal Computer Unit.

09	01	00	00

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	There is a parameter error.
7	Ended abnormally.

Related Functions

PCMSRD

Program Examples

In this example, the message ”`Message sent`” is transmitted to another Unit.

```
'*******************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM              *
'*                  QUICK BASIC                        *
'*******************************************************
'
        DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"
        DECLARE SUB pcmswr CDECL ALIAS "_b_pcmswr"
        DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT "pcopen error : rtn = "; rtn%
               GOTO finish
        END IF
RETRY:
        ne% = 1                     'Network address
        no% = 1                     'Node address
        un% = &H14                  'Unit address (Unit #4)
        ct% = 14                    'No. of transmission bytes
        d$ = "Transmission message:" 'Transmission message
        PRINT "Message sent: "; d$
        CALLS pcmswr(ne%,no%,un%,ct%,d$,rtn%)
        IF rtn%=3 GOTO RETRY
        IF rtn%<>0 THEN
               PRINT "pcmswr error : rtn = "; rtn%
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT "Sample program finished"
        END
```

PCREAD READ AREA

Purpose

This function reads data from a data area of a Programmable Controller.

Specifiers

Operation number:	03H (3)
Library function name:	_b_pcread

Format

```
PCREAD (NE%,NO%,SUBFMT$,VALUE$(1),RTN%)
PCREAD (NE%,NO%,SUBFMT$,VALUE%(1),RTN%)
```

Parameters

NE%	: Source network address (input/output)
NO%	: Source node address (input/output)
SUBFMT$	: Subformat (input)
VALUE$ (1)	: Reception buffer (string) (input/output)
VALUE% (1)	: Reception buffer (numeral) (input/output)
RTN%	: Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
SUBFMT$	String	Refer to the comments below.
VALUE$ (1)	String	Array variable: 1 must be entered in the parentheses.
VALUE% (1)	Integer	Array variable: 1 must be entered in the parentheses.
RTN%	Integer	Status entered as integer after execution of function.

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

If the source node is in the same network, then use 0 for NE%.

If the source node is the same, then use 0 for NE% and NO%.

Use array variables VALUE $ (1) or VALUE % (1) specified with SUBFMT$.

For the variable SUBFMT$, assign strings (subformats) that specify the read area, data conversion method, and so on.

Subformats can be in the following forms:
” [SUB, START, NUM,] FMT [,FMT....]”
The contents inside of the brackets can be omitted. FMT can specify multiple items. Character types ($) and integer types (%), however, cannot be present. In addition, if the contents inside of the brackets are omitted, only one integer type (%) can be specified.

SUB	: Subcommand
START	: Beginning word for read
NUM	: Number of words to be read
FMT	: Storage format

Note Characters for SUBFMT $ must all be in upper case.

For SUB (subcommand), set the Programmable Controller data area.

Subcommands

Subcommand	Specified area	Word/bit addresses	Data unit
@R	CIO Area	0 to 2,555	Word
@A	Auxiliary Area (A) (A256 to A511 are read only.)	0 to 511	Word
@TN	Transition Flags (read only)	0 to 1,023	Bit
@ST	Step Flags (read only)	0 to 1,023	Bit
@TF	Timer Flags (read only)	0 to 1,023	Bit
@T	Timer present values	0 to 1,023	Word
@CF	Counter Flags (read only)	0 to 1,023	Bit
@C	Counter present values	0 to 1,023	Word
@D	DM	0 to 24,575	Word
@E0	Expansion DM (EM) Bank 0	0 to 32,765	Word
@E1	Expansion DM (EM) Bank 1	0 to 32,765	Word
@E2	Expansion DM (EM) Bank 2	0 to 32,765	Word
@E3	Expansion DM (EM) Bank 3	0 to 32,765	Word
@E4	Expansion DM (EM) Bank 4	0 to 32,765	Word
@E5	Expansion DM (EM) Bank 5	0 to 32,765	Word
@E6	Expansion DM (EM) Bank 6	0 to 32,765	Word
@E7	Expansion DM (EM) Bank 7	0 to 32,765	Word
@SG	CPU Bus Link Area (G)	0 to 255	Word

The ranges shown for word and bit numbers are for the CV1000 Programmable Controller. Refer to the *CV-series PC Operation Manual: Ladder Diagrams (W202)* for address ranges for other PCs.

If SUB, START, and NUM are designated, then data will be read from the area specified by the subcommand. The designated number of words will be read, starting with the beginning word for the read.

SUB, START, and NUM must all be designated or all omitted. The user can't designate just one or two of these values.

If SUB, START, and NUM are omitted, the PCREAD function won't read the Programmable Controller's data area; operation will be controlled by the PC's SEND(192) command. When SEND(192) is executed, the read will be executed according to operands specified for SEND(192).

The Personal Computer Unit will standby for the SEND(192) command. (The standby state can be cancelled by pressing the Esc Key.) One integer type (%) can be specified for the storage format. The network and node addresses of the Programmable Controller that executed SEND(192) will be entered in NE% and NO%.

The range for NUM (number of words to be read) is 1 to 256.

FMT (storage format) is a character string that interprets the data from the words that are read, and specifies the conversion method for storing the data in memory.

Data conversion by FMT will be as follows:

a) The contents of the Programmable Controller data will be interpreted according to specifications. (Data that cannot be interpreted according to specifications will be regarded as 0.)

b) The data will be converted to numerals or character strings.

c) It will be stored in array variables.

The FMT form will be:
{% or $} n {I, H, O, or A}
In n, the number of words to be read with this storage format is specified. If n is not specified, it will be regarded as 1.

Storage Format (FMT) Chart

FMT	Interpretation	Data storage form
%nI	Decimal integer	Integer
%nH	Hexadecimal integer	Integer
%nO	Octal integer	Integer
$nI	Decimal integer	String expressing decimals
$nH	Hexadecimal integer	String expressing hexadecimals
$nO	Octal integer	String expressing octals
$nA	2-character ASCII code	Two characters

The following are examples of the storage formats.

1, 2, 3...

1. **I-type (Decimal) Formats** (%nI, $nI)

   Programmable Controller word data: 1234

   Character types: PCREAD (....”...$1I,” VALUE$ (1))
   → VALUE$ (1) =”1234”

   Integer types: PCREAD (....”...%1I,” VALUE% (1))
   → VALUE% (1) =1234

2. **H-type (Hexadecimal) Formats** (%nH, $nH)

   Programmable Controller word data: 89AB

   Character types: PCREAD (....”...$1H,” VALUE$ (1))
   → VALUE$ (1) =”89AB”

   Integer types: PCREAD (....”...%1H,” VALUE% (1))
   → VALUE% (1) =&H89AB = –30293

3. **O-type (Octal) Formats** (%nO, $nO)

   Programmable Controller word data: 1234

   Character types: PCREAD (....”...$1O,” VALUE$ (1))
   → VALUE$ (1) =”1234”

   Integer types: PCREAD (....”...%1O,” VALUE% (1))
   → VALUE% (1) =&O1234 = 668

4. **A-type (ASCII code) Format** ($nA)

   Programmable Controller word data: 5152

   Character types: PCREAD (....”...$1A,” VALUE$ (1))
   → VALUE$ (1) =”QR”

   The ASCII codes for Q and R are &H51 and &H52 respectively.
   When multiple words are read in A-type format, they will all be stored in VALUE$ (1).

When character data is specified for FMT, it is necessary to make an array declaration, for the number of FMT characters, for the character array variable VALUE$ ( ) as a reception buffer. In addition, it is necessary to assign, in advance, dummy data (e.g., spaces) for the FMT contents, and to reserve memory space. The number of bytes for the assigned dummy data will be as follows, with “n” being the number of FMT words read:

I, H, and O types:	4 bytes
A type:	2 x n bytes

Example: PCREAD (NE%, NO%, ”@R, 100, 5, $2I, $3I,” VALUE$ (1))
$2I → Stored in VALUE$ (1) and VALUE$ (2).
$3I → Stored in VALUE$ (3), VALUE$ (4), and VALUE$ (5).
Therefore, five VALUE$ ( ) array elements must be provided.

Example: PCREAD (NE%, NO%, ”@D, 100, 2, $2A,” VALUE$ (1))
$2A → 2 x 2-byte VALUE$ (1) areas are required.

When numeral data is specified for FMT, it is necessary to make an array declaration of an integer array variable VALUE% ( ), the same size as NUM or the total number of FMT words read (n).

Example: PCREAD (NE%, NO%, ”@R, 100, 5, %2I, %3I,” VALUE% (1))
%2I → Stored in VALUE% (1) and VALUE% (2).
%3I → Stored in VALUE% (3), VALUE% (4), and VALUE% (5).
Therefore, five VALUE% ( ) array elements must be provided.

The number of words actually read will be the number of words specified by NUM. Be sure to set the total number of words designated by “n” of FMT so that is the same as the NUM value.

If the total number of words designated by “n” of FMT is greater than the reception buffer, operation will not be guaranteed. If it is smaller than the reception buffer, data cannot be read into the remaining portion of the reception buffer.

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
4	Ended by Esc Key.
5	There is a parameter error.
7	Ended abnormally.
8	There is an error in the routing table.

Related Functions

PCWRITE

Program Examples

In this example, data is read from words 0 to 3 in the DM area of the Programmable Controller at network address 1 and node address 2. The data is displayed as characters and numerals.

```
'*******************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM              *
'*                  QUICK BASIC                        *
'*******************************************************
'
        DECLARE SUB pcopen CDECL ALIAS ”_b_pcopen”
        DECLARE SUB pcread CDECL ALIAS ”_b_pcread”
        DECLARE SUB pcclose CDECL ALIAS ”_b_pcclose”

        DIM value%(100), value$(100)

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT ”pcopen error : rtn = ”; rtn%
               GOTO finish
        END IF
RETRY0:
        ne% = 1                     'Network address
        no% = 2                     'Node address
        FOR i% = 1 TO 4
               value$(i%) = STRING$(4, ” ”)     'Read
buffer
        NEXT
   CALLS pcread(ne%,no%,”@D,0,4,$2I,$2H, value$(1),rtn%)
        IF rtn%=3 GOTO RETRY0
        IF rtn%<>0 THEN
               PRINT ”pcread error : rtn = ”; rtn%
        ELSE
```

```
               PRINT ”Word 0 data ($2I):”;value$(1)
               PRINT ”Word 1 data ($2I):”;value$(2)
               PRINT ”Word 2 data ($2H):”;value$(3)
               PRINT ”Word 3 data ($2H):”;value$(4)
        END IF
RETRY1:
   CALLS pcread(ne%,no%,”@D,0,4,%2I,%2H, value%(1),rtn%)
        IF rtn%=3 GOTO RETRY1
        IF rtn%<>0 THEN
               PRINT ”pcread error : rtn = ”; rtn%
        ELSE
               PRINT ”Word 0 data (%2I):”;value%(1)
               PRINT ”Word 1 data (%2I):”;value%(2)
               PRINT ”Word 2 data (%2H):”;value%(3)
               PRINT ”Word 3 data (%2H):”;value%(4)
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT ”Exit sample program.”
        END
```

PCWRITE WRITE AREA

Purpose

This function writes data to a data area of a Programmable Controller.

Specifiers

Operation number:	04H (4)
Library function name:	_b_pcwrite

Format

```
PCWRITE (NE%,NO%,SUBFMT$,VALUE$ (1),RTN%)
PCWRITE (NE%,NO%,SUBFMT$,VALUE% (1),RTN%)
```

Parameters

NE%	:	Destination network address (input/output)
NO%	:	Destination node address (input/output)
SUBFMT$	:	Subformat (input)
VALUE$ (1)	:	Reception buffer (string) (input)
VALUE% (1)	:	Reception buffer (numeral) (input)
RTN%	:	Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
SUBFMT$	String	Refer to the comments below.
VALUE$ (1)	String	Array variable: 1 must be entered in the parentheses.
VALUE% (1)	Integer	Array variable: 1 must be entered in the parentheses.
RTN%	Integer	Status entered as integer after execution of function.

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

If the destination node is in the same network, then use 0 for NE%.

If the destination node is the local Programmable Controller, then use 0 for NE% and NO%.

Use array variables VALUE $ (1) or VALUE % (1) specified with SUBFMT$.

For the variable SUBFMT$, assign strings (subformats) that specify the write area, data conversion method, and so on.

Subformats can be in the following form:
" [SUB, START, NUM,] FMT [,FMT....]"
The contents inside of the brackets can be omitted. FMT can specify multiple items. Character types ($) and integer types (%), however, cannot be present. In addition, if the contents inside of the brackets are omitted, only one integer type (%) can be specified.

SUB : Subcommand
START : Beginning word for writing
NUM : Number of words to be written
FMT : Storage format

Note Characters for SUBFMT $ must all be in upper case.

For SUB (subcommand), set the Programmable Controller data area.

Subcommands

Subcommand	Specified area	Word/bit addresses	Data unit
@R	CIO Area	0 to 2,555	Word
@A	Auxiliary Area (A) (A256 to A511 are read only.)	0 to 511 0 to 1,023	Word
@TN	Transition Flags (read only)	0 to 1,023	Bit
@ST	Step Flags (read only)	0 to 1,023	Bit
@TF	Timer Flags (read only)	0 to 1,023	Bit
@T	Timer present values	0 to 1,023	Word
@CF	Counter Flags (read only)	0 to 1,023	Bit
@C	Counter present values	0 to 24,575	Word
@D	DM	0 to 32,765	Word
@E0	Expansion DM (EM) Bank 0	0 to 32,765	Word
@E1	Expansion DM (EM) Bank 1	0 to 32,765	Word
@E2	Expansion DM (EM) Bank 2	0 to 32,765	Word
@E3	Expansion DM (EM) Bank 3	0 to 32,765	Word
@E4	Expansion DM (EM) Bank 4	0 to 32,765	Word
@E5	Expansion DM (EM) Bank 5	0 to 32,765	Word
@E6	Expansion DM (EM) Bank 6	0 to 32,765	Word
@E7	Expansion DM (EM) Bank 7	0 to 32,765	Word
@SG	CPU Bus Link Area (G)	0 to 255	Word

The ranges shown for word and bit numbers are for the CV1000 Programmable Controller. Refer to the *CV-series PC Operation Manual: Ladder Diagrams (W202)* for address ranges for other PCs.

If SUB, START, and NUM are designated, then data will be written from the area specified by subcommand. The designated number of words will be read, starting with the beginning word for writing.

SUB, START, and NUM must all be designated or all omitted. The user can't designate just one or two of these values.

If SUB, START, and NUM are omitted, the PCWRITE function won't write data to the Programmable Controller's data area; operation will be controlled by the Programmable Controller's RECV(193) command.

When RECV(193) is executed, the Programmable controller will be able to receive the data and the data will be written to the data area. The Personal Computer Unit will standby until the RECV(193) command is executed. (The standby state can be cancelled by pressing the Esc Key.) One integer type (%) can be specified for the storage format. The network and node addresses of the Programmable Controller that executed RECV(193) will be entered in NE% and NO%.

The range for NUM (number of words to be written) is 1 to 256.

FMT (storage format) is a character string that specifies the conversion method when data stored in the transmission buffer is written to a Programmable Controller area in the designated data format.

Data conversion by FMT will be as follows:

a) Data stored in array variables serving as a transmission buffer will be converted to the designated data format.

b) The data will be written to the Programmable Controller area.

The FMT form will be:
{% or $} n {I, H, O, or A}

In n, the number of words to be written with this storage format is specified.

Storage Format (FMT) Chart

FMT	Data stored in transmission buffer	Format of data written to Programmable Controller area
%nI	Integer	Decimal
%nH	Integer	Hexadecimal
%nO	Integer	Octal
$nI	String expressing decimals	Decimal
$nH	String expressing hexadecimals	Hexadecimal
$nO	String expressing octals	Octal
$nA	2n characters	2-character ASCII code

The following are examples of the respective storage formats.

1, 2, 3...

1. **I-type (Decimal) Formats** (%nI, $nI)
   Character data: VALUE$ (1) = ”1234”
   PCWRITE (....”...$1I,” VALUE$ (1))
   or
   Numeral data: VALUE%(1) = 1234
   PCWRITE (....”...%1I,” VALUE% (1))

   Results: Programmable Controller word data: 1234
2. **H-type Formats** (%nH, $nH)
   Character data: VALUE$ (1) = ”89AB”
   PCWRITE (....”...$1H,” VALUE$ (1))
   or
   Numeral data:VALUE%(1) = &H89AB = –30293
   PCWRITE (....”...%1H,” VALUE% (1))

   Results: Programmable Controller word data: 89AB
3. **O-type Formats** (%nO, $nO)
   Character data: VALUE$ (1) = ”1234”
   PCWRITE (....”...$1O,” VALUE$ (1))
   or
   Numeral data: VALUE%(1) = $O1234 = 668
   PCWRITE (....”...%1O,” VALUE% (1))

   Results: Programmable Controller word data: 1234
4. **A-type Format** ($nA)
   Character data: VALUE$ (1) = ”QR”
   PCWRITE (....”...$1A,” VALUE$ (1))
   The ASCII codes for Q and R are &H51 and &H52 respectively.

   7Results: Programmable Controller word data: 5152

When character data is specified for FMT, it is necessary to make an array declaration for the number of FMT characters for the character array variable VALUE$ ( ) as a transmission buffer. In addition, it is necessary to assign data in advance for FMT contents. The number of bytes for the assigned transmission data will be as follows, with “n” being the number of FMT words written.

I, H, and O types: 4 bytes
A type: 2 x n bytes

Example: PCWRITE (NE%, NO%, ”@R, 100, 5, $2I, $3I,” VALUE$ (1))
$2I → Transmission data necessary for VALUE$ (1) and VALUE$ (2).
$3I → Transmission data necessary for VALUE$ (3), VALUE$ (4), and VALUE$ (5).

Example: PCWRITE (NE%, NO%, ”@D, 100, 2, $2A,” VALUE$ (1))
$2A → 2 x 2 byte region is necessary for VALUE$ (1).

When numeral data is specified for FMT, it is necessary to make an array declaration of the integer array variable VALUE% ( ), just the same as NUM, and to assign transmission data.

Example: PCWRITE (NE%, NO%, ”@R, 100, 5, %2I, %3I,” VALUE% (1))
%2I → Transmission data necessary for VALUE% (1) and VALUE% (2).
%3I → Transmission data necessary for VALUE% (3), VALUE% (4), and VALUE% (5).
Therefore, five VALUE% ( ) array elements must be provided.

The number of words actually written will be the number of words specified by NUM. Be sure to set the total number of words designated by “n” of FMT so that is the same as the NUM value.

If the total number of words designated by “n” of FMT is greater than the transmission buffer, operation cannot be ensured. If it is smaller than the transmission buffer, data cannot be read into the remaining portion of the transmission buffer.

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
4	Ended by Esc Key.
5	There is a parameter error.
7	Ended abnormally.
8	There is an error in the routing table.

Related Functions *PCREAD*

Program Examples

In this example, data is written to words 0 to 3 in the DM area of the Programmable Controller at network address 1 and node address 2.

```
'********************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM               *
'*                  QUICK BASIC                         *
'********************************************************
'
        DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"

        DECLARE SUB pcwrite CDECL ALIAS "_b_pcwrite"
        DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

        DIM value%(100), value$(100)

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT "pcopen error : rtn = "; rtn%
               GOTO finish
        END IF
RETRY0:
        ne% = 1                     'Network address
        no% = 2                     'Node address
        value$(1) = "1001"          'Write data
        value$(2) = "2002"          'Write data
        value$(3) = "3003"          'Write data
        value$(4) = "4004"          'Write data
  CALLS pcwrite(ne%,no%,"@D,0,4,$2I,$2H, value$(1),rtn%)
        IF rtn%=3 GOTO RETRY0
        IF rtn%<>0 THEN
               PRINT "pcwrite error : rtn = "; rtn%
        END IF
RETRY1:
        value%(1) = 500             'Write data
        value%(2) = 600             'Write data
        value%(3) = 700             'Write data
        value%(4) = 800             'Write data
  CALLS pcwrite(ne%,no%,"@D,0,4,%2I,%2H, value%(1),rtn%)
        IF rtn%=3 GOTO RETRY1
        IF rtn%<>0 THEN
               PRINT "pcwrite error : rtn = "; rtn%
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT "Exit sample program."
        END
```

PCSTAT PROGRAMMABLE CONTROLLER STATUS

Purpose

This function accesses and controls Programmable Controller status.

Specifiers

Operation number: 05H (5)
Library function name: _b_pcstat

Format

```
PCSTAT (NE%,NO%,MAIN%,REC%,VALUE$,RTN%)
PCSTAT (NE%,NO%,MAIN%,REC%,VALUE%,RTN%)
```

Parameters

NE%	:	Network address (input)
NO%	:	Node address (input)
MAIN%	:	Command (input)
REC%	:	Number of records read (input/output)
VALUE$	:	Storage buffer (string) (input/output)
VALUE%	:	Storage buffer (numeral) (input/output)
RTN%	:	Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
MAIN%	Integer	0 to 3 (Refer to the comments below.)
REC%	Integer	0 to 20
VALUE$	String	Refer to the comments below.
VALUE%	Integer	Refer to the comments below.
RTN%	Integer	Status entered as integer after execution of function.

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

If the node is in the same network, then use 0 for NE%.

If the local Programmable Controller is being controlled, then use 0 for NE% and NO%.

For the variable MAIN%, assign an integer from 0 to 3 as shown in the following table.

MAIN%	Operation
0	Read Controller Status
1	Clear Error
2	Read Error Log
3	Clear Error Log

For the variable REC%, assign an integer from 0 to 20. This parameter will only be valid, however, when “2” is assigned for MAIN%. If “0” is assigned for REC%, and the present total number of history records is from 1 to 20, then the number of records actually read will be stored in REC%.

The value assigned to MAIN% will determine whether the storage buffer will be the character variable VALUE$ or the numeral variable VALUE%.

When MAIN% is 0 or 2

The character variable VALUE$ will be taken as the storage buffer. Provide the memory area by assigning in advance characters (e.g., spaces) for the necessary number of bytes. If the memory area is not provided, an error will be generated. If “0” is assigned for MAIN%, 26 bytes will be provided. If “2” is assigned, 10 bytes will be provided for the error log 1 record. Refer to page 66 for the format and meaning of the data stored.

When MAIN% is 1

For the integer variable VALUE%, assign the FAL number for clearing errors.

When MAIN% is 3

The storage buffer will not be used, so no matter what value is assigned it will be ignored. The variable description, however, cannot be omitted.

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	There is a parameter error.
7	Ended abnormally.

Program Examples

In this example, the status is read for the Programmable Controller at network address 1 and node address 2, and then errors are cleared.

```
'*******************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM              *
'*                  QUICK BASIC                        *
'*******************************************************
'
        DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"
        DECLARE SUB pcstat CDECL ALIAS "_b_pcstat"
        DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT "pcopen error : rtn = "; rtn%
               GOTO finish
        END IF
RETRY0:
        ne% = 1                             'Network address
        no% = 2                             'Node address
        main% = 0                           'Command (read status)
        rec% = 0
        value$ = STRING$(30 ," ")   'Read buffer
        CALLS pcstat(ne%, no%, main%, rec%, value$,rtn%)
        IF rtn%=3 GOTO RETRY0
        IF rtn%<>0 THEN
               PRINT "pcstat error : rtn = "; rtn%
        ELSE
         PRINT "Read status..0..1..2..3..4..5..6..7..8";
          PRINT "..9..a..b..c..d..e..f"
          PRINT "                  .. ";
                    FOR I = 1 TO 26
                            IF (I MOD 16) = 1 THEN
                                    PRINT""
                         PRINT "                  .. ";
                    END IF
                    B$ = MID$(value$,I,1)
                    PRINT"";RIGHT$("0"+HEX$(ASC(B$)),2);
               NEXT
               PRINT""
        END IF
RETRY1:
        main% = 1                           'Command (clear error)
        value% = 2                          'Reset FAL no.
        PRINT "Reset FAL no.:";value%
        CALLS pcstat(ne%, no%, main%, rec%, value%,rtn%)
        IF rtn%=3 GOTO RETRY1
        IF rtn%<>0 THEN
               PRINT "pcstat error : rtn = "; rtn%
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT "Exit sample program."
        END
```

PCMODE PROGRAMMABLE CONTROLLER MODE

Purpose

This function changes the Programmable Controller's mode (PROGRAM, DEBUG, MONITOR, and RUN).

Specifiers

Operation number:	06H (6)
Library function name:	_b_pcmode

Format

```
PCMODE (NE%,NO%,MODE%,RTN%)
```

Parameters

NE%	: Network address (input)
NO%	: Node address (input)
MODE%	: Programmable Controller mode (input)
RTN%	: Returned value (output)

Parameters	Format	Contents
NE%	Integer	0 to 127
NO%	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
MODE%	Integer	0 to 3 (Refer to the comments below.)
RTN%	Integer	0 to 3 (Refer to the comments below.)

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

If the node is in the same network, then use 0 for NE%.

If the local Programmable Controller is being controlled, then assign 0 for NE% and NO%.

For the variable MODE%, assign an integer from 0 to 3.

MODE%	Operation
0	Puts Programmable Controller in PROGRAM Mode (operation stopped).
1	Puts Programmable Controller in DEBUG Mode.
2	Puts Programmable Controller in MONITOR Mode.
3	Puts Programmable Controller in RUN Mode.

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	There is a parameter error.
7	Ended abnormally.

Program Examples

In this example, the Programmable Controller at network address 1, node address 2, is placed in DEBUG Mode.

```
'*******************************************************
'*           BASIC LIBRARY SAMPLE PROGRAM              *
'*                  QUICK BASIC                        *
'*******************************************************
'
        DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"
        DECLARE SUB pcmode CDECL ALIAS "_b_pcmode"
        DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

        CALLS pcopen(rtn%)
        IF rtn%<>0 THEN
               PRINT "pcopen error : rtn = "; rtn%
               GOTO finish
        END IF
RETRY0:
        ne% = 1                             'Network address
        no% = 2                             'Node address
        mode% = 1                           'Mode
           PRINT"Mode value: ";mode%
        CALLS pcmode(ne%, no%, mode%, rtn%)
        IF rtn%=3 GOTO RETRY0
        IF rtn%<>0 THEN
               PRINT "pcmode error : rtn = "; rtn%
        END IF

        CALLS pcclose(rtn%)
finish:
        PRINT "Exit sample program."
        END
```

PCCLOSE CLOSE CPU BUS

Purpose

This function terminates usage of the CPU bus.

Specifiers

Operation number: 07H (7)
Library function name: _b_pcclose

Format

`PCCLOSE (RTN%)`

Parameters

RTN% : Returned value (output)

Parameters	Format	Contents
RTN%	Integer	Status entered as integer after execution of function.

Comments

This function must be executed at the end to terminate use of the CPU bus.

Be sure to designate RTN%. If the argument is omitted, an error will be generated.

Returned Values

RTN%	Meaning
0	Ended normally.
1	Bus is not open.

Related Functions

PCOPEN

Program Examples

For a program example, refer to page 26, *PCOPEN*.

3-4 Before Using C

This section will explain how to use the CPU Bus Library for C.

3-4-1 Functions

The following table lists the available functions.

Name	Operation	Reference
pcopen( )	Opens the CPU bus.	p. 46
pcmsrd( )	Receives a message from another Unit.	p. 46
pcmswr( )	Sends a message to another Unit.	p. 49
pcread( )	Reads data from a Programmable Controller data area.	p. 51
pcwrite( )	Writes data into a Programmable Controller data area.	p. 58
pcstat( )	Accesses and controls Programmable Controller status.	p. 65
pcmode( )	Changes Programmable Controller operating modes.	p. 73
pcclose( )	Closes the CPU bus.	p. 75

3-4-2 Using Library Functions

1, 2, 3...

1. Declare each function when creating the source file for the application software.
2. Create an execution program by linking with the library (CLIBx.LIB) when compiling.

Note The functions used depend on the program model that is created. There are four sects of functions available, depending on the model size.

Example: Creating Execution Program SAMPLE.EXE (Small Model) from Source File SAMPLE.C

```
CL /c SAMPLE.C . . . . . . . . . . . . . . . . . . . . . . . .   Creates object.
LINK SAMPLE.OBJ,SAMPLE.EXE,NUL,CLIBS   Links library.
```

3-5 C Functions

This section will explain the functions available in the C library.

Headings

Each function will be covered in terms of the following headings.

Purpose An outline of the function’s operation will be provided.

Specifiers When functions are defined with Quick BASIC, the specifiers are the names, in the library, of the functions that are to be used.

Format This is the format for functions used in a program.

Parameters The meanings of the parameters used in the function’s format are given. A chart will show each parameter’s format, range, and type.

Comments Parameter contents and setting precautions will be explained.

Returned Values The meanings of the values returned by the functions will be given. Based on these values you can determine whether functions have executed correctly or whether errors have been generated.

Related Functions This is the names of other functions related to the function being described. You can refer to the other functions in order to gain a better understanding of the one in question.

Program Examples One or more examples will be given of programs in Quick BASIC. Program files are kept in directory \CVLIB\C\SAMPLE in the program disk.

pcopen( ) OPEN CPU BUS

Purpose This function opens the CPU bus so that it can be used.

Format

```
unsigned int pcopen( )
```

Parameters None

Comments The CPU bus is declared to be open. When you want to use other functions of the CPU Bus Library, you must execute *pcopen* ( ) first.

Returned Values

Value	Meaning
0	Ended normally.
1	CPU Bus Driver is not installed.
2	Bus is already open.

Related Functions *pcclose*

Program Examples In this example, the CPU bus is opened and then closed.

```
/*******************************************/
/*         Open processing                */
/*******************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );

void main(void)
{
        printf(”Opening bus \n”);
       switch( pcopen( ) ) {        /*Open*/
        case 0:
               printf(”Opened normally. \n”);
               break;
        case 1:
               printf(”Driver not installed \n”);
               break;
        case 2:
               printf(”Already open \n”);
               break;
        }
pcclose( );                          /*Close*/
}
```

pcmsrd( ) RECEIVE MESSAGE

Purpose With this function, messages can be received from other Units.

Format

```
unsigned int pcmsrd (ne, no, un, bytes, val, dsz, timer)
unsinged char far *ne;
unsigned char far *no;
unsigned char far *un;
unsigned int far *bytes;
unsigned char far *val;
unsigned int dsz;
unsigned int timer;
```

Parameters

ne : “far” pointer for storage area of source network address
no : “far” pointer for storage area of source node address
un : “far” pointer for storage area of source node address
bytes : “far” pointer for storage area of actual number of reception bytes

val : “far” pointer for reception buffer
dsz : Number of bytes requested for reception
timer : Timer value

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
un	Integer	Absolute address
bytes	Integer	Length of message actually received
val	String	Beginning address for reception buffer
dsz	Integer	Maximum length for message received (1 to 538)
timer	Integer	0 to 32767 (Unit: x110 ms)

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

The source unit address is specified by absolute address. The absolute address is the unit number + $10. For example, if a CPU Bus Unit’s unit number is set to 4, its absolute address would be $14.

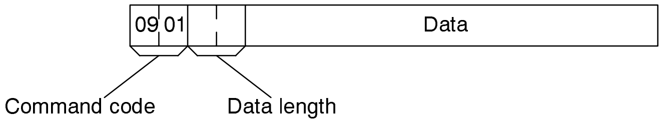
When *pcmsrd* ( ) is executed, the Personal Computer Unit will receive a message from any Unit. When a message is received, the network address, node address, and unit address of the receiving Unit will be stored respectively in the areas indicated by *ne*, *no*, and *un.* The length of the message will be stored in the area indicated by “bytes.”

Assign a value for the reception waiting time to the variable *timer*. The reception waiting time can be set within a range of 0 to 65535, in units of 110 ms. If the message cannot be received within the waiting time, a timeout (returned value: 6) will result. If “0” is set, arriving messages will be received and there will be no waiting time.

This function will be ended once a single message of any length has been received, even if no message for the number of bytes requested for reception (dsz%) is received. In addition, if a message of a size exceeding the *dsz*% is received, the *dsz*% portion of the message will be returned to the user and the remainder will be discarded.

This function uses command 0901 of the FINS commands. When this function is executed, the Unit will standby for a message that has command code 0901.

The following diagram shows the format for FINS command 0901. Use this format when transmitting from outside the Personal Computer Unit.



The following example shows the command format needed to send “ABCDEF.”

09	01	00	06	A	B	C	D	E	F

Return the following response to the message source if the Personal Computer Unit received the message normally.

09	01	00	00

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
3	The Programmable Controller is busy. Please retry.
5	The argument is not correct.
6	No message was received from the specified Unit. A timeout occurred.
8	There is an error in the routing tables.

Related Functions

pcmswr

Program Examples

In this example, a message will be received from another Unit.

```
/********************************************/
/*    Message receive processing           */
/********************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcmsrd( );

unsigned char buf[3000];

void main(void)
{
        int ret;
        unsigned char ne;
        unsigned char no;
        unsigned char un;
        int timer=50;
        int bytes;
        int dsz;
        int i;

        printf(”Message reception \n”);
        printf(”ret = %d\n”,ret = pcopen( ));
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }
        dsz=100;
          printf(”Request to receive message of %d bytes
\n”,dsz);
        printf(”Reception waiting time is %d in units of
110 ms \n”,timer);

        switch(pcmsrd((unsigned char far *)&ne,
        (unsigned char far *)& no, (unsigned char far *)
        &un, (unsigned char far *)&bytes, (unsigned char
        far *)buf, dsz, timer)) {
        case 0:
               printf(”PCMSRD normal \n”);
               printf(”Network 0x%x\n”, ne);
               printf(”Node 0x%x\n”, no);
               printf(”Unit 0x%x\n”, un);
                      printf(”Number of reception bytes
%d\n”,bytes)
               for (i = 0; i < bytes ; i++)
                      printf(”0x%02x ”,buf[i]);
               printf(”\n”);
               break;
        case 1:
               printf(”PCOPEN not executed \n”);
               break;
        case 3:
                    printf(”Programmable Controller busy
\n”);
               break;
        case 5:
               printf(”Improper argument passed \n”);
               break;
        case 6:
               printf(”Message not received \n”);
               break;
        case 8:
               printf(”Routing table error \n”);
               break;
        }

        pcclose( );
}
```

pcmswr( ) SEND MESSAGE

Purpose With this function, messages can be transmitted to other Units.

Format

```
unsigned int pcmswr (ne, no, un, bytes, &val)
unsigned char ne;
unsigned char no;
unsigned char un;
unsigned int byte;
unsigned char far *val;
```

Parameters

ne	:	Destination network address
no	:	Destination node address
un	:	Destination unit address
bytes	:	Number of bytes requested for transmission
val	:	Reception buffer

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
un	Integer	Specifies absolute address.
bytes	Integer	1 to 538
val	String	Transmission buffer beginning address

Note The maximum length varies according to the type of network through which the message is transmitted.

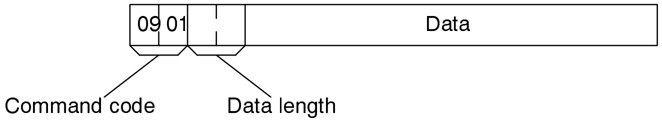
Comments The unit address is specified by the unit number + $10 (the absolute address). For example, if a CPU Bus Unit’s unit number is set to 5, its absolute address will be $15.

When *pcmswr* ( ) is executed, the Personal Computer Unit will transmit a message only to the designated Unit.

This function will not be ended until the message has been completely received at the destination (i.e., until a response is returned to the transmission source).

This function uses command 0901 of the FINS commands, so messages can be transmitted to BASIC Units.

This function uses command 0901 of the FINS commands. When this function is executed, a message with command code 0901 is sent. The format of FINS command 0901 is shown in the following diagram.



The following example shows the command format needed to send “ABCDEF.”

09	01	00	06	A	B	C	D	E	F

When the destination Unit isn’t a Personal Computer Unit and the message is received, return the following response addressed to the Personal Computer Unit.

09	01	00	00

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	The argument is not correct.
7	Ended abnormally.

Related Functions *pcmsrd*

Program Examples In this example, the 10-bit message “0123456789” is transmitted to another Unit.

```
/*******************************************/
/*    Message transmission processing      */
/*******************************************/
unsigned char  buf[] = ”0123456789”;
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcmswr( );

void main(void)
{
        int   ret;
        unsigned char ne=0x01;
        unsigned char no=0x01;
        unsigned char un=0x15
        int bytes=10;
        int i;

        printf(”ret = %d\n”,ret = pcopen( ));
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }
        printf(”Message transmission \n”);

        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
        printf(”Unit 0x%x\n”, un);
           printf(”Will transmit message of %d bytes to
\n\n”,bytes);

        printf(”Transmission data is as follows: \n\t”);
        for (i = 0; i < bytes; i++)
               printf(”0x%02x ”,buf[i]);
        printf(”\n\n”);

        switch (pcmswr(ne, no, un, bytes, (unsigned char
        far *)buf)) {
        case 0:
               printf(”PCMSWR normal \n”);
               break;
        case 1:
               printf(”PCOPEN not executed \n”);
               break;
        case 2:
               printf(”Network address error \n”);
               break;
        case 3:
              printf(”Programmable Controller busy \n”);
               break;
        case 5:
               printf(”Improper argument passed \n”);
               break;
        case 7:
               printf(”Ended abnormally \n”);
               break;
        }
        pcclose( );
}
```

pcread( ) READ AREA

Purpose

This function reads data from a data area of a Programmable Controller.

Format

```
unsigned int pcread (ne, no, sub_format, val[,val...])
unsigned char ne;
unsigned char no;
unsigned char far *sub_format;
unsigned int fat *val; ....... (1)
or
unsigned char far *val; ...... (2)
```

Parameters

ne : Source network address
no : Source node address
sub_format : Subformat
val : Reception buffer

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
sub_format	String	Refer to comments.
val (1)	Integer	Reception buffer beginning address
val (2)	String	---

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

For *val*, specify the beginning address for the area that is to store the data that is read. Depending on the number of words read, and the storage format, several *val* may be specified. In addition, *val* may be either integer type or character type.

For the variable *sub_format*, assign character strings for the various specifications for the data to be read. The character strings that can be assigned for *sub_format* are as follows:

Sub_format Forms
[sub, start, num,] format [, format...]
Each string is separated by commas. The contents inside of the brackets may be omitted. Several formats may be specified.

sub: Subformat
start: Beginning word for reading
num: Number of words to be read
format: Storage format

Note Characters for sub_format must all be in upper case.

For *sub*, set the Programmable Controller data area type.

Subcommands

Subcommand	Specified area	Word/bit addresses	Data unit
@R	CIO Area	0 to 2,555	Word
@A	Auxiliary Area (A) (A256 to A511 are read only.)	0 to 511	Word
@TN	Transition Flags (read only)	0 to 1,023	Bit
@SN	Step Flags (read only)	0 to 1,023	Bit
@TF	Timer Flags (read only)	0 to 1,023	Bit
@T	Timer present values	0 to 1,023	Word
@CF	Counter Flags (read only)	0 to 1,023	Bit
@C	Counter present values	0 to 1,023	Word
@D	DM	0 to 24,575	Word
@E0	Expansion DM (EM) Bank 0	0 to 32,765	Word
@E1	Expansion DM (EM) Bank 1	0 to 32,765	Word
@E2	Expansion DM (EM) Bank 2	0 to 32,765	Word
@E3	Expansion DM (EM) Bank 3	0 to 32,765	Word
@E4	Expansion DM (EM) Bank 4	0 to 32,765	Word
@E5	Expansion DM (EM) Bank 5	0 to 32,765	Word
@E6	Expansion DM (EM) Bank 6	0 to 32,765	Word
@E7	Expansion DM (EM) Bank 7	0 to 32,765	Word
@SG	CPU Bus Link Area (G)	0 to 255	Word

Note
1. Commands must be specified in upper case.
2. The ranges shown for word and bit numbers are for CV1000 Programmable Controllers. Refer to the *CV-series PC Operation Manual: Ladder Diagrams (W202)* for address ranges for other PCs.

Start specifies the beginning address, in the area designated by the subcommand, from which data is to be read. If the data is to be read from the top of the designated area, then set this to “0.”

Num specifies how many words are to be read, beginning with the address designated by *start.* This can be set from 1 to 256.

Sub, *start*, and *num* data takes on meaning only when it is all complete. No part can be omitted. It is possible, however, for the data to be omitted completely. In that case, the *pcread* ( ) function will not read Programmable Controller areas. Instead, data can be read from the Programmable Controller by means of SEND(192). That is, when SEND(192) is executed at the Programmable Controller, the data that is transmitted from the Programmable Controller is read. Until SEND(192) is executed at the Programmable Controller, execution will wait. (Wait status can be ended by pressing the Esc Key.) One integer type (%) can be specified for the storage format. The external variables *cv_netadr* and *cv_nodadr* respectively are assigned the network address and the node address of the Programmable Controller executing SEND(192). When executing via SEND(192), add the following two lines at the top of the program.

```
extern unsigned char cv_netadr;
extern unsigned char cv_nodadr;
```

Format is a character string that interprets one word of data that is read and specifies the conversion method for storing the data in memory.

Data conversion by *format* will be as follows:

a) The contents of the Programmable Controller data will be interpreted according to specifications. (Data that cannot be interpreted according to specifications will be regarded as 0.)

b) The data will be converted to numerals or character strings.

c) It will be stored in the reception buffer.

The *format* form will be:
{% or $} n {I, H, O, or A}

In n, the number of words to be read with this storage format is specified. If n is not specified, it will be regarded as 1.

Note The number of words actually read will be the number of words specified by*num.* Be sure to set the total number of words designated by *n* of *format* such that is the same as the *num* value.

Storage Formats

Format	Operation and storage method
%nI	Interprets read data as decimal, and stores it as numerals. Data that cannot be interpreted as decimal will be regarded as “0.” A single word of data will be stored in a single *val* variable. If a numeral of 2 or greater is specified for *n*, then *n* number of *val* variables will be required. The variable *val* type will be numeral for *unsigned int far* *.
%nH	Interprets read data as hexadecimal, and stores it as numerals. Data that cannot be interpreted as hexadecimal will be regarded as “0.” Specifications regarding *val* are the same as for %*nI*.
%nO	Interprets read data as octal, and stores it as numerals. Data that cannot be interpreted as octal will be regarded as “0.” Specifications regarding *val* are the same as for %*nI*.
%SnI	Interprets read data as decimal, and stores it as numerals. Data that cannot be interpreted as decimal will be regarded as “0.” Read data will be stored in array variable *val* [ ]. If two or more numerals are specified for n, then n number of arrays will be required for the variable *val* [ ]. The variable *val* [ ] type will be numeral for *unsigned int far* *.
%SnH	Interprets read data as hexadecimal, and stores it as numerals. Data that cannot be interpreted as hexadecimal will be regarded as “0.” Specifications regarding *val* are the same as for %*SnI*.
%SnO	Interprets read data as octal, and stores it as numerals. Data that cannot be interpreted as octal will be regarded as “0.” Specifications regarding *val* are the same as for %*nI*.
$nI	Interprets read data as decimal, and converts it to character strings for storage. Data that cannot be interpreted as decimal will be regarded as “0.” A single word of data will be stored in a single *val* variable. If a numeral of 2 or greater is specified for n, then n number of *val* variables will be required. A single word (two bytes) of data will be converted to a 4-byte character string expressing a 4-digit numeral. Thus four bytes of data area will be required for a single variable *val*. The variable *val* type will be character for *unsigned char far* *.
$nH	Interprets read data as hexadecimal, and stores it as numerals. Data that cannot be interpreted as hexadecimal will be regarded as “0.” Specifications regarding *val* are the same as for *$nI*.
$nO	Interprets read data as octal, and stores it as numerals. Data that cannot be interpreted as octal will be regarded as “0.” Specifications regarding *val* are the same as for *$nI*.

Format	Operation and storage method
$nA	Interprets read data as ASCII, and converts it to character strings for storage. Data read into array variable *val* [ ] will be stored. A single word (two bytes) of data will be converted to a 2-byte character string. Thus a data area of n x 2 bytes will be required for an array variable *val* [ ]. The variable *val* type will be character, for *unsigned char far **.
$SnI	Interprets read data as decimal, and converts it to character strings for storage. Data that cannot be interpreted as decimal will be regarded as “0.” Read data will be stored in array variable *val* [ ]. If two or more numerals are specified for n, then n number of arrays will be required for the variable *val* [ ]. A single word (two bytes) of data will be converted to a 4-byte character string expressing a 4-digit numeral. Thus a data area of n x 4 bytes will be required for an array variable *val* [ ]. The variable *val* type will be character for *unsigned char far **.
$SnH	Interprets read data as hexadecimal, and converts it to character strings for storage. Data that cannot be interpreted as hexadecimal will be regarded as “0.” Specifications regarding *val* are the same as for *$SnI*.
$SnO	Interprets read data as octal, and converts it to character strings for storage. Data that cannot be interpreted as octal will be regarded as “0.” Specifications regarding *val* are the same as for *$SnI*.
$SnA	Interprets read data as ASCII, and converts it to character strings for storage. Data read into array variable *val* [ ] will be stored. A single word (two bytes) of data will be converted to a 2-byte character string. Thus a data area of n x 2 bytes will be required for an array variable *val* [ ]. The variable *val* type will be character, for *unsigned char far **.

Conversion Examples

Examples are given below, according to the various storage formats, of converting data that has been read.

1, 2, 3... 1. **I-type (Decimal) Format**

Read data 12345678

Numerals

```
unsigned int val1, val2;
unsigned int far *p1, *p2;
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcread(...”...,%2I”,p1,p2);
```

Results

```
val1 = 1234 = 0x04d2
val2 = 5678 = 0x162e
```

Numeral Array

```
signed int val[2]
unsigned int far *p;
p = (unsigned int far *)val;
pcread(...,”...,%S2I”,p);
```

Results

```
val[0] = 1234 = 0x04d2
val[1] = 5678 = 0x162e
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;
p1 = (unsigned char far *)val1;
p2 = (unsigned char far *)val2;
pcread(...,”...,$2I”,p1,p2);
```

Results

```
val1[0] = ’1’ = 0x31; val1[1] = ’2’ = 0x32;
val1[2] = ’3’ = 0x33; val1[3] = ’4’ = 0x34;
val2[0] = ’5’ = 0x35; val2[1] = ’6’ = 0x36;
val2[2] = ’7’ = 0x37; val2[3] = ’8’ = 0x38;
```

Character Array

```
unsigned char val[8];
unsigned char far *p;
```

```
p = (unsigned char far *)val;
pcread(...,”...,$S2I”,p);
```

Results

```
val[0] = ’1’ = 0x31; val[1] = ’2’ = 0x32;
val[2] = ’3’ = 0x33; val[3] = ’4’ = 0x34;
val[4] = ’5’ = 0x35; val[5] = ’6’ = 0x36;
val[6] = ’7’ = 0x37; val[7] = ’8’ = 0x38;
```

2. **H-type (Hexadecimal) Format**

Read data 789ABCDE

Numerals

```
unsigned int val1, val2;
unsigned int far *p1, *p2;
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcread(...”...,%2H”,p1,p2);
```

Results

```
val1 = 0x789a
val2 = 0xbcde
```

Numeral Array

```
unsigned int val[2];
unsigned int far *p;
p = (unsigned int far *)val;
pcread(...,”...,%S2H”,p);
```

Results

```
val[0] = 0x789a
val[1] = 0xbcde
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;
p1 = (unsigned char far *)val1;
p2 = (unsigned char far *)val2;
pcread(...,”...,$2H”,p1,p2);
```

Results

```
val1[0] = ’7’ = 0x37; val1[1] = ’8’ = 0x38;
val1[2] = ’9’ = 0x39; val1[3] = ’A’ = 0x41;
val2[0] = ’B’ = 0x42; val2[1] = ’C’ = 0x43;
val2[2] = ’D’ = 0x44; val2[3] = ’E’ = 0x45;
```

Character Array

```
unsigned char val[8];
unsigned char far *p;
p = (unsigned char far *)val;
pcread(...,”...,$S2H”,p);
```

Results

```
val[0] = ’7’ = 0x37; val[1] = ’8’ = 0x38;
val[2] = ’9’ = 0x39; val[3] = ’A’ = 0x41;
val[4] = ’B’ = 0x42; val[5] = ’C’ = 0x43;
val[6] = ’D’ = 0x44; val[7] = ’E’ = 0x45;
```

3. **O-type (Octal) Format**

Read data 12345670

Numerals

```
unsigned int val1, val2;
unsigned int far *pl, *p2;
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcread(...”...,%20”,p1,p2);
```

Results

```
val1 = 01234 = 0x029c
val2 = 05670 = 0x0bb8
```

Numeral Array

```
unsigned int val[2];
unsigned int far *p;
p = (unsigned int far *)val;
pcread(...,”...,%S20”,p);
```

Results

```
val[0] = 01234 = 0x029c
val[1] = 05670 = 0x0bb8
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;
p1 = (unsigned char far *)val1;
p2 = (unsigned char far *)val2;
pcread(...,”...,$20”,p1,p2);
```

Results

```
val1[0] = ’1’ = 0x31; val1[1] = ’2’ = 0x32;
val1[2] = ’3’ = 0x33; val1[3] = ’4’ = 0x34;
val2[0] = ’5’ = 0x35; val2[1] = ’6’ = 0x36;
val2[2] = ’7’ = 0x37; val2[3] = ’0’ = 0x30;
```

Character Array

```
unsigned char val1[8];
unsigned char far *p;
p = (unsigned char far *)val;
pcread(...,”...,$S20”,p);
```

Results

```
val[0] = ’1’ = 0x31; val[1] = ’2’ = 0x32;
val[2] = ’3’ = 0x33; val[3] = ’4’ = 0x34;
val[4] = ’5’ = 0x35; val[5] = ’6’ = 0x36;
val[6] = ’7’ = 0x37; val[7] = ’0’ = 0x30;
```

4. **A-type (ASCII Code) Format**

Read data 51525354

Characters

```
unsigned char val1[4];
unsigned char far *p1;
p1 = (unsigned char far *)val1;
pcread(...,”...,$2A”,p1);
```

Results

```
val1[0] = ’Q’ = 0x51; val1[1] = ’R’ = 0x52;
val1[2] = ’S’ = 0x53; val1[3] = ’T’ = 0x54;
```

Character Array

```
unsigned char val[4];
unsigned char far *p;
p = (unsigned char far *)val;
pcread(...,”...,$S2A”,p);
```

Results

```
val[0] = ’Q’ = 0x51; val[1] = ’R’ = 0x52;
val[2] = ’S’ = 0x53; val[3] = ’T’ = 0x54;
```

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
4	Ended by Esc Key.
5	The argument is not correct.
7	Ended abnormally.
8	There is an error in the routing tables.

Related Functions

pcwrite

Program Examples

In this example, data is read from DM words 0 to 2 of a Programmable Controller at the same node.

```
/*******************************************/
/*    Data area read processing            */
/*******************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcread( );

#define INT_SIZE 2              /* % */
#define WORD    3               /* WORD */

void main(void)
{
        int   ret;
        int i,j;
        unsigned char sub[80];
        unsigned char far *subp;
        unsigned int  buf[WORD][INT_SIZE]; /* 3 words */
        unsigned int  far *bufp;
        unsigned char ne=0;
        unsigned char no=0;

        bufp = (unsigned int far *)&buf[0][0];

        printf(”ret = %d\n”,ret = pcopen( ));
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }
        subp = (unsigned char far *)&sub[0];
        strcpy(sub, ”@D,0,3,%S3H”);

        printf(”Data area read processing \n”);

        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
           printf(”Programmable Controller area will be
read\n\n”);
          printf(”Specified command for reading is\”%s\”
\n”, sub);

        switch (pcread(ne, no, subp, bufp)) {
```

```
        case 0:
               printf(”PCREAD normal \n”);
                     printf(”Data from words read is as
follows: \n\t”);
               for ( i = 0; i < 3; i++)
                       printf(”0x%04x ”,*bufp++);
               break;
        case 1:
               printf(”PCOPEN not executed \n”);
               break;
        case 2:
               printf(”Network address error \n”);
               break;
        case 3:
                    printf(”Programmable Controller busy
\n”);
               break;
        case 4:
               printf(”Ended by Esc Key \n”);
               break;
        case 5:
               printf(”Improper argument passed \n”);
               break;
        case 7:
               printf(”Ended abnormally \n”);
               break;
        case 8:
               printf(”Routing table error \n”);
               break;
        }
        pcclose( );
}
```

pcwrite( ) WRITE AREA

Purpose This function writes data to the data area of a Programmable Controller.

Format

```
unsigned int pcwrite (ne, no, sub_format, val[,val...])

unsigned char ne;
unsigned char no;
unsigned char far *sub_format;

unsigned int far *val; ....... (1)
or
unsigned char far *val; ...... (2)
```

Parameters

ne	: Destination network address
no	: Destination node address
sub_format	: Subformat
val	: Transmission buffer

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
sub_format	String	Refer to comments.
val (1)	Integer	---
val (2)	String	Transmission buffer beginning address

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

For *val*, specify the beginning address for the area that is to store the data that is written. Depending on the number of words written and the storage format, several *val* may be specified. In addition, *val* may be either integer type or character type.

For the variable *sub_format*, assign character strings for the various specifications for the data to be written. The character strings that can be assigned for *sub_format* are as follows:

Sub_format Forms

[sub, start, num,] format [, format...]

Each string is separated by commas. The contents inside of the brackets may be omitted. Several formats may be specified.

sub:	Subformat
start:	Beginning word for writing
num:	Number of words to be written
format:	Storage format

Note Characters for *sub_format* must all be in upper case.

Subcommands

Subcommand	Specified area	Word/bit addresses	Data unit
@R	CIO Area	0 to 2,555	Word
@A	Auxiliary Area (A) (A256 to A511 are read only.)	0 to 511	Word
@TN	Transition Flags (read only)	0 to 1,023	Bit
@ST	Step Flags (read only)	0 to 1,023	Bit
@TF	Timer Flags (read only)	0 to 1,023	Bit
@T	Timer present values	0 to 1,023	Word
@CF	Counter Flags (read only)	0 to 1,023	Bit
@C	Counter present values	0 to 1,023	Word
@D	DM	0 to 24,575	Word
@E0	Expansion DM (EM) Bank 0	0 to 32,765	Word
@E1	Expansion DM (EM) Bank 1	0 to 32,765	Word
@E2	Expansion DM (EM) Bank 2	0 to 32,765	Word
@E3	Expansion DM (EM) Bank 3	0 to 32,765	Word
@E4	Expansion DM (EM) Bank 4	0 to 32,765	Word
@E5	Expansion DM (EM) Bank 5	0 to 32,765	Word
@E6	Expansion DM (EM) Bank 6	0 to 32,765	Word
@E7	Expansion DM (EM) Bank 7	0 to 32,765	Word
@SG	CPU Bus Link Area (G)	0 to 255	Word

Note

1. Commands must be in upper case.
2. The ranges shown for word and bit numbers are for CV1000 Programmable Controllers. Refer to the *CV-series PC Operation Manual: Ladder Diagrams W202)* for address ranges for other PCs.

3. The CPU Bus Link Area cannot be designated for the local Programmable Controller. Specify ne =0, no = 0, and word = 128+unit number× 8 ±8 words.

Start specifies the beginning address, in the area designated by the subcommand, from which data is to be written. If the data is to be written from the top of the designated area, then set this to "0."

Num specifies how many words are to be written, beginning with the address designated by *start.* This can be set from 1 to 256.

Sub, *start*, and *num* data takes on meaning only when it is all complete. No part can be omitted. It is possible, however, for the data to be omitted completely. In that case, the *pcwrite* ( ) function will not write to Programmable Controller areas. Instead, data can be written to the Programmable Controller by means of RECV(193). That is, when RECV(193) is executed at the Programmable Controller, the Programmable Controller will prepare to receive and the data is then written. Until RECV(193) is executed at the Programmable Controller, execution will wait. (Wait status can be ended by pressing the Esc Key.) One integer type (%) can be specified for the storage format. The external variables *cv_netadr* and *cv_nodadr* respectively will be assigned the network address and the node address of the Programmable Controller executing RECV(193). When executing via RECV(193),add the following two lines at the top of the program.

```
extern unsigned char cv_netadr;
extern unsigned char cv_nodadr;
```

Format is a character string that specifies the method of conversion in order to write into a Programmable Controller area, in the designated data type, the data stored in the transmission buffer.

Data conversion by *format* will be as follows:

a) The data stored in the transmission buffer will be converted to the designated data type. (When converting to numerals, overflow portions will be ignored.)

b) The data will be written to the Programmable Controller area.

The *format* form will be:
{% or $} n {I, H, O, or A}
In n, the number of words to be read with this storage format is specified.

Note The number of words actually written will be the number of words specified by *num.* Be sure to set the total number of words designated by *n* of *format* such that is the same as the *num* value.

Storage Formats

Format	Operation and storage method
%nI	Regards write data as numerals and expands it to decimal (BCD conversion) for writing. At the time of expansion to decimal, overflow portions will be ignored. A single word of data will be stored in a single variable *val*. Thus *n* number of variable *val* will be required. The variable *val* type will be numeral for *unsigned int far *.*
%nH	Regards write data as numerals and expands it to hexadecimal for writing. At the time of expansion to decimal, overflow portions will be ignored. Specifications regarding *val* are the same as for %*nI*.
%nO	Regards write data as numerals and expands it to octal for writing. At the time of expansion to octal, overflow portions will be ignored. Specifications regarding *val* are the same as for %*nI*.
%SnI	Regards write data as numerals and expands it to decimal (BCD conversion) for writing. At the time of expansion to decimal, overflow portions will be ignored. Write data will be regarded as being stored in array variable *val* [ ]. For array variable *val* [ ] with n number of arrays, it is necessary to specify one array. The variable *val* [ ] type will be numeral for *unsigned int far *.*
%SnH	Regards write data as numerals and expands it to hexadecimal for writing. At the time of expansion to decimal, overflow portions will be ignored. Specifications regarding *val* are the same as for %*SnI*.
%SnO	Regards write data as numerals and expands it to octal for writing. At the time of expansion to octal, overflow portions will be ignored. Specifications regarding *val* are the same as for %*SnI*.

Format	Operation and storage method
$nI	Regards write data as characters and expands it to decimal (BCD conversion) for writing. At the time of expansion to decimal, overflow portions will be ignored. A single word of data will be stored in a single variable *val*. Thus n number of variable *val* will be required. Four bytes of data will become a single word (two bytes) of data. Thus four bytes of data area will be required for a single variable *val*. The variable *val* type will be character for *unsigned char far **.
$nH	Regards write data as characters and expands it to hexadecimal for writing. At the time of expansion to decimal, overflow portions will be ignored. Specifications regarding *val* are the same as for *$nI*.
$nO	Regards write data as characters and expands it to octal for writing. At the time of expansion to octal, overflow portions will be ignored. Specifications regarding *val* are the same as for *$nI*.
$nA	Regards write data as ASCII code and expands it for writing. Data from n words is stored in a single variable *val*. Two bytes of data will become a single word (two bytes) of data. Thus variable *val* will require a data area of n x 2 bytes. The variable *val* type will be character for *unsigned char far **.
$SnI	Regards write data as characters and expands it to decimal (BCD conversion) for writing. At the time of expansion to decimal, overflow portions will be ignored. Write data will be regarded as being stored in array variable *val* [ ]. Four bytes of data will become a single word (two bytes) of data. Thus it will be necessary to specify a character array variable *val* [ ] which has n x 4 arrays. The variable *val* type will be character for *unsigned char far **.
$SnH	Regards write data as characters and expands it to hexadecimal for writing. At the time of expansion to decimal, overflow portions will be ignored. Specifications regarding [ ] are the same as for *$SnI*.
$SnO	Regards write data as characters and expands it to octal for writing. At the time of expansion to octal, overflow portions will be ignored. Specifications regarding [ ] are the same as for *$SnI*.
$SnA	Regards write data as ASCII code and expands it for writing. Write data will be regarded as being stored in array variable *val* [ ]. Two bytes of data will become a single word (two bytes) of data. Thus it will be necessary to specify a character array variable *val* [ ] which has a data area of n x 2 bytes. The variable *val* type will be character for *unsigned char far *.*

Conversion Examples

Examples are given below, according to the various storage formats, of converting data that has been read.

1, 2, 3...

1. **I-type (Decimal) Format**

Data written to Programmable Controller 12345678

Numerals

```
unsigned int val1, val2;
unsigned int far *p1, *p2;

val1 = 1234;
val2 = 5678;
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcwrite(...”...,%2I”,p1,p2);
```

Numeral Array

```
unsigned int val[2];
unsigned int far *p;

val[0] = 1234;
val[1] = 5678;
p = (unsigned int far *)val;
pcwrite(...,”...,%S2I”,p);
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;

val1[0] = ’1’ = 0x31; val1[1] = ’2’ = 0x32;
val1[2] = ’3’ = 0x33; val1[3] = ’4’ = 0x34;
val2[0] = ’5’ = 0x35; val2[1] = ’6’ = 0x36;
val2[2] = ’7’ = 0x37; val2[3] = ’8’ = 0x38;
p1 = (unsigned char far *)val1;
```

```
p2 = (unsigned char far *)val2;
pcwrite(...,”...,$2I”,p1,p2);
```

Character Array

```
unsigned char val[8];
unsigned char far *p;

val[0] = ’1’ = 0x31; val[1] = ’2’ = 0x32;
val[2] = ’3’ = 0x33; val[3] = ’4’ = 0x34;
val[4] = ’5’ = 0x35; val[5] = ’6’ = 0x36;
val[6] = ’7’ = 0x37; val[7] = ’8’ = 0x38;
p = (unsigned char far *)val;
pcwrite(...,”...,$S2I”,p);
```

2. **H-type (Hexadecimal) Format**

   Data written to Programmable Controller 789ABCDE

   Numerals

```
unsigned int val1, val2;
unsigned int far *pl, *p2;

val1 = 0x789a
val2 = 0xbcde
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcwrite(...”...,%2H”,p1,p2);
```

Numeral Array

```
unsigned int val[2];
unsigned int far *p;

val[0] = 0x789a
val[1] = 0xbcde
p = (unsigned int far *)val;
pcwrite(...,”...,%S2H”,p);
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;

val1[0] = ’7’ = 0x37; val1[1] = ’8’ = 0x38;
val1[2] = ’9’ = 0x39; val1[3] = ’A’ = 0x41;
val2[0] = ’B’ = 0x42; val2[1] = ’C’ = 0x43;
val2[2] = ’D’ = 0x44; val2[3] = ’E’ = 0x45;
p1 = (unsigned char far *)val1;
p2 = (unsigned char far *)val2;
pcwrite(...,”...,$2H”,p1,p2);
```

Character Array

```
unsigned char val[8];
unsigned char far *p;

val[0] = ’7’ = 0x37; val[1] = ’8’ = 0x38;
val[2] = ’9’ = 0x39; val[3] = ’A’ = 0x41;
val[4] = ’B’ = 0x42; val[5] = ’C’ = 0x43;
val[6] = ’D’ = 0x44; val[7] = ’E’ = 0x45;
p = (unsigned char far *)val;
pcwrite(...,”...,$S2H”,p);
```

3. **O-type (Octal) Format**

   Data written to Programmable Controller 12345670

Numerals

```
unsigned int val1, val2;
unsigned int far *pl, *p2;

val1 = 01234;
val2 = 05670;
p1 = (unsigned int far *)&val1;
p2 = (unsigned int far *)&val2;
pcwrite(...”...,%20”,p1,p2);
```

Numeral Array

```
unsigned int val[2];
unsigned int far *p;

val[0] = 01234;
val[1] = 05670;
p = (unsigned int far *)val;
pcwrite(...,”...,%S20”,p);
```

Characters

```
unsigned char val1[4],val2[4];
unsigned char far *p1,*p2;

val1[0] = ’1’ = 0x31; val1[1] = ’2’ = 0x32;
val1[2] = ’3’ = 0x33; val1[3] = ’4’ = 0x34;
val2[0] = ’5’ = 0x35; val2[1] = ’6’ = 0x36;
val2[2] = ’7’ = 0x37; val2[3] = ’0’ = 0x30;
p1 = (unsigned char far *)val1;
p2 = (unsigned char far *)val2;
pcwrite(...,”...,$20”,p1,p2);
```

Character Array

```
unsigned char val[8];
unsigned char far *p;

val[0] = ’1’ = 0x31; val[1] = ’2’ = 0x32;
val[2] = ’3’ = 0x33; val[3] = ’4’ = 0x34;
val[4] = ’5’ = 0x35; val[5] = ’6’ = 0x36;
val[6] = ’7’ = 0x37; val[7] = ’0’ = 0x30;
p = (unsigned char far *)val;
pcwrite(...,”...,$S20”,p);
```

4. **A-type (ASCII Code) Format**

   Data written to Programmable Controller 51525354

Characters

```
unsigned char val[4];
unsigned char far *p;

val[0] = ’Q’ = 0x51; val[1] = ’R’ = 0x52;
val[2] = ’S’ = 0x53; val[3] = ’T’ = 0x54;
p = (unsigned char far *)val;
pcwrite(...,”...,$2A”,p);
```

Character Array

```
unsigned char val[4];
unsigned char far *p;

val[0] = ’Q’ = 0x51; val[1] = ’R’ = 0x52;
val[2] = ’S’ = 0x53; val[3] = ’T’ = 0x54;
```

```
         p = (unsigned char far *)val;
         pcwrite(...,”...,$S2A”,p);
```

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
4	Ended by Esc Key.
5	The argument is not correct.
7	Ended abnormally.
8	There is an error in the routing table.

Related Functions

pcread

Program Examples

In this example, data is read from DM words D00000 to D00002 of a Programmable Controller at the same node.

```
/********************************************/
/*    Data area write processing           */
/********************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcwrite( );

void main(void)
{
        int   ret, i
        unsigned char sub[20];
        static unsigned char buf[8] =
              {0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38};
        unsigned char  far *subp;
        unsigned char  far *bufp;
        unsigned char ne=0;
        unsigned char no=0;

        printf(”ret = %d\n”,ret = pcopen( ));
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }
        subp = (unsigned char far *)sub;
        bufp = (unsigned char far *)buf;
        strcpy(sub, ”@D,0,3,%S3H”);

        printf(”Data area write processing \n”);

        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
          printf(” Programmable Controller area will be
written to \n\n”);
           printf(”Specified command for writing is \”%s\
\n”, sub);
        printf(”Write data:”);
        for ( i = 0; i < 8; i++)
               printf(”%02x ”,buf[i]);
        printf(”\n\n”);
        switch (pcwrite(ne, no, subp, bufp)) {
        case 0:
               printf("PCWRITE normal \n");
               break;
        case 1:
               printf("PCOPEN not executed \n");
               break;
        case 2:
               printf("Network address error \n");
               break;
        case 3:
                    printf("Programmable  Controller  busy
\n");
               break;
        case 4:
               printf("Ended by Esc Key \n");
               break;
        case 5:
               printf("Improper argument passed \n");
               break;
        case 7:
               printf("Ended abnormally \n");
               break;
        case 8:
               printf("Routing table error \n");
               break;
        }
        pcclose( );
}
```

pcstat( ) PROGRAMMABLE CONTROLLER STATUS

Purpose This function accesses and controls Programmable Controller status.

Format

```
unsigned int pcstat (ne, no, mcmd, ch, val)
unsigned char ne;
unsigned char no;
unsigned char mcmd;
unsigned char far *ch;
unsigned int far *val; ....... (1)
or
unsigned char far *val; ...... (2)
```

Parameters

ne	:	Network address
no	:	Node address
mcmd	:	Command
ch	:	Number of records read
val	:	Storage buffer

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
mcmd	Integer	0 to 3 (Refer to comments.)
ch	Integer	0 to 20
val (1)	Integer	Refer to comments.
val (2)	String	Refer to comments.

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

For *mcmd*, assign the data to be controlled with respect to the Programmable Controller. The control items are shown below, according to the value for *mcmd*.

mcmd	Control item
0	Controller status readout
1	Error clearing
2	Error Log readout
3	Error Log clearing

The values for *ch* and *val* will change according to the value for mcmd. The assigned values for mcmd and the set values for *ch* and *val* are explained next.

1, 2, 3...

1. **Read Controller Status (When *mcmd* = 0)**
   This lets the user know the present status of the Programmable Controller (current errors, for example). Controller status data is configured in 26 bytes, as shown in the chart below. Set for *val* the beginning address of the 26-byte data reception area. In this case, *ch* will not be used, so set dummy data.

1 byte	1 byte	2 bytes	2 bytes	2 bytes	2 bytes	16 bytes
Operating status	Mode	Fatal error information	Non-fatal error information	Message notice	Present FAL No.	Error message
0						25

The content of each data section is as follows:

- **Operating Status**
  $ 00: Stopped (User program not being executed)
  $ 01: Running (User program being executed)
  $ 80: CPU waiting
- **Mode**
  $ 00: PROGRAM Mode
  $ 01: DEBUG Mode
  $ 02: MONITOR Mode
  $ 04: RUN Mode
- **Fatal Error Information**
  The meanings the bits in the 2-byte area are as follows:

Bit	Meaning when bit is ON
00	Watchdog timer error
01 to 05	Not used
06	FALS error
07	SFC fatal error
08	Cycle time too long
09	Program error
10	I/O setting error
11	I/O capacity exceeded
12	CPU bus error
13	Duplicate number error
14	I/O bus error
15	Memory error

- **Non-fatal Error Information**
  The meanings the bits in the 2-byte area are as follows:

Bit	Meaning when bit is ON
00, 01	Not used
02	Momentary power interruption
03	CPU Bus setting error
04	Battery error
05	SYSMAC BUS error
06	SYSMAC BUS/2 error
07	CPU Bus Unit error
08	Not used
09	I/O verification error
10	Not used
11	SFC non-fatal error
12	Indirect DM BCD error
13	JMP error
14	Not used
15	FAL error

- **Message Notice**
  When a particular bit turns ON, it indicates that there is a message.

Bit	Meaning when bit is ON
00	Message No. 0
01	Message No. 1
02	Message No. 2
03	Message No. 3
04	Message No. 4
05	Message No. 5
06	Message No. 6
07	Message No. 7

- **Present FAL No.**
  The error with the highest current priority is indicated. If there is no FAL or FALS error, the value will be $ 00.

- **Error Message**
  The error message for the present FAL No. is indicated. If there is no FAL or FALS error, or if there is no message for the present FAL number, then spaces will be entered. The value is $ 4100 greater than the FAL or FALS number.

2. **Error Clearing (When mcmd = 1)**
   This clears current FAL or FALS errors. For *val*, specify the beginning address of the area in which the FAL number to be cleared is stored. In this case, *ch* will not be used, so set dummy data. This operation will be ended normally even if there is no error for the specified FAL number.

3. **Error Log Readout (When mcmd = 2)**
   This lets the user know the history of errors that have occurred at the Pro-

grammable Controller. A single error log record consists of 10 bytes, as shown in the following table.

No. of bytes	Item
2	Error code
2	Content
2	Min/sec
2	Day/time
2	Year/month

Up to 20 error records can be read at one time. For *ch*, specify the beginning address of the area for which the number of records are to be read. For *val*, set the beginning address of the reception area (the number of error records to be read x 10 bytes). When returning from this function, the number of records actually read will be stored in the area set by *ch*. If “0” is specified for the number of records read (*ch*), then all current records will be stored in the area set by *ch*. (Nothing will be stored in *val*.)

4. **Error Log Clearing (When *mcmd* = 3)**
   This clears the error log stored in the Programmable Controller. *ch* and *val* are not used, so assign dummy data.

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	The argument is not correct.
7	Ended abnormally.

Program Examples

In this example, the status of the Programmable Controller is read.

```
/********************************************/
/* Status: Controller information readout  */
/********************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcstat( );

void main(void)
{
        int ret,i,j;
        unsigned char val[26];      /*   Status   storage
area */
        unsigned char  far *valp;
        unsigned char  dmy;         /* Dummy */
        unsigned char  far *dmyp;
        unsigned char ne=0;         /*  Network   address
*/
        unsigned char no=0;         /* Node address */
        unsigned int mcmd = 0;      /* Main command */

        valp = (unsigned char far *)&val[0];
        dmyp = (unsigned char far *)&dmy;

        ret = pcopen( );
        printf(”pcopen( ) ret = %d\n”,ret);
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
  }
  printf(”Controller status readout \n\n”);
  printf(”Network 0x%x\n”, ne);
  printf(”Node 0x%x\n”, no);
  printf(” Programmable Controller will be read \n\n”);

  switch (pcstat(ne, no, mcmd, dmyp, valp)) {
   case 0:
    printf(”PCSTAT normal \n”);
    printf(”Controller information is as follows: \n”);
        printf(”Operating status (1byte) : 0x%02x\n”,
val[0]);
    printf(”Mode (1byte) : 0x%02x\n”, val[1]);
     printf(”Fatal error information (2 bytes) : 0x%02x
0x%02x\n”, val[2], val[3]);
       printf(”Non–fatal error information (2 bytes) :
0x%02x 0x%02x\n”, val[4], val[5]);
      printf(”Message Y/N (2 bytes) : 0x%02x 0x%02x\n”,
val[6], val[7]);
        printf(”Present FAL No. (2 bytes) : 0x%02x
0x%02x\n”, val[8], val[9]);
    printf(”Error message (16 bytes) : \n\t”);
    for ( i = 10; i < 26; i++)
      printf(”%02x ”, val[i]);
      printf(”\n\n”);
      break;
   case 1:
    printf(”PCOPEN not executed \n”);
    break;
   case 2:
    printf(”Network address is invalid \n”);
    break;
   case 3:
    printf(”Programmable Controller busy \n”);
    break;
   case 5:
    printf(”Improper argument passed \n”);
    break;
   case 7:
    printf(”Ended abnormally \n”);
    break; ”
  }
  pcclose( );
}
```

In this example, a Programmable Controller error is cleared.

```
/********************************************/
/*       Status: Error clearing             */
/********************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcstat( );

void main(void)
{
        int ret;
        unsigned char val[2];        /*  FAL  No.  setting
area */
        unsigned char  far *valp;
        unsigned char  dmy;          /* Dummy */
        unsigned char  far *dmyp;
        unsigned char ne=0;          /*  Network  address
*/
        unsigned char no=0;          /* Node address */
        unsigned int mcmd = 1;       /* Main command */

        printf(”Error clearing \n\n”);

        if (pcopen( ) == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }

        val[0] = 0xff;
        val[1] = 0xfe;
            /* Present error clearing */
        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
         printf(”    Programmable Controller errors will
be cleared \n\n”);
          printf(”Specified FAL No. is 0x%02x 0x%02x\n”,
val[0], val[1]);

        valp = (unsigned char far *)&val[0];
        dmyp = (unsigned char far *)&dmy;

        switch (pcstat(ne, no, mcmd, dmyp, valp)) {
        case 0:
         printf(”PCSTAT normal \n”);
         break;
        case 1:
         printf(”PCOPEN not executed \n”);
         break;
        case 2:
         printf(”Network address is invalid \n”);
         break;
        case 3:
         printf(”Programmable Controller busy \n”);
         break;
        case 5:
         printf(”Improper argument passed \n”);
         break;
        case 7:
         printf(”Ended abnormally \n”);
         break;
        }
        pcclose( );
}
```

In this example, the Programmable Controller’s error log is read.

```
/******************************************/
/*    Status: Error Log readout           */
```

```
/*******************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcstat( );

void main(void)
{
        int ret,i,j;
        unsigned char val[20][10];  /* Error Log storage
history */
        unsigned char  far *valp;
        unsigned char  ch=0;        /* Controller status
readout */
        unsigned char  far *chp;
        unsigned char ne=0;         /* Network number */
        unsigned char no=0;         /* Node number */
        unsigned int mcmd = 2;      /* Main command */

        printf(”Error Log readout \n”);

        if (pcopen( ) == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }

        ch = 20;
        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
             printf(”Will read first error log for ___
Programmable Controller \n”);
         printf(”Number of history readouts requested is
%d \n”, ch);

        valp = (unsigned char far *)&val[0][0];
        chp = (unsigned char far *)&ch;

        switch (pcstat(ne, no, mcmd, chp, valp)) {
        case 0:
         printf(”PCSTAT normal \n\n”);
           printf(”Actual number of histories read is %d
\n\n”, ch);
         if ( ch > 0) {
                      printf(”Error Log data read is as
follows: \n”);
               for ( i = 0; i < ch; i++) {
                   printf(”No. %d error log: ”,i+1);
                   for ( j = 0; j < 10; j++)
                       printf(”0x%02x ”,val[i][j]);
                   printf(”\n”);
               }
         } else
               printf(”Error Log is not stored \n”);
         printf(”\n”);
         break;
        case 1:
         printf(”PCOPEN not executed \n”);
         break;
        case 2:
         printf(”Network address is invalid \n”);
         break;
        case 3:
         printf(”Programmable Controller busy \n”);
         break;
        case 5:
         printf(”Improper argument passed \n”);
         break;
        case 7:
         printf(”Ended abnormally \n”);
         break;
        }
        pcclose( );
}
```

In this example, the error log is cleared.

```
/*******************************************/
/*    Status: Error Log clearing           */
/*******************************************/
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcstat( );

void main(void)
{
        int ret,i,j;
        unsigned char dmy;          /* Dummy */
        unsigned char  far *dmyp;
        unsigned char ne=0;         /* Network number */
        unsigned char no=0;         /* Node number */
        unsigned int mcmd = 3;      /* Main command */

        if (pcopen( ) == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }

        printf(”Error Log clearing \n\n”);
        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
        printf(”  Programmable Controller error log will
be cleared \n\n”);

        dmyp = (unsigned char far *)&dmy;

        switch (pcstat(ne, no, mcmd, dmyp, dmyp)) {
        case 0:
         printf(”PCSTAT normal \n”);
         break;
        case 1:
         printf(”PCOPEN not executed \n”);
         break;
        case 2:
         printf(”Network address is invalid \n”);
         break;
        case 3:
         printf(”Programmable Controller busy \n”);
         break;
        case 5:
         printf(”Improper argument passed \n”);
         break;
        case 7:
         printf(”Ended abnormally \n”);
         break;
        }
        pcclose( );
}
```

pcmode( ) PROGRAMMABLE CONTROLLER MODE

Purpose

This function changes the Programmable Controller’s mode (PROGRAM, DEBUG, MONITOR, and RUN).

Format

```
unsigned int pcmode (ne, no, mode)
unsigned char ne;
unsigned char no;
unsigned char mode;
```

Parameters

ne : Network address
no : Node address
mode : Programmable Controller mode

Parameters	Format	Contents
ne	Integer	0 to 127
no	Integer	SYSMAC NET, Ethernet: 0 to 127 SYSMAC LINK: 0 to 62 (see note)
mode	Integer	Refer to comments.

Note The maximum length varies according to the type of network through which the message is transmitted.

Comments

For *mode*, the following four operating modes can be set.

mode	Operating mode
0	PROGRAM Mode (operation stopped)
1	DEBUG Mode
2	MONITOR Mode
3	RUN Mode

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.
2	Network address is invalid.
3	Programmable Controller or Repeater is busy. Try again.
5	The argument is not correct.
7	Ended abnormally.

Program Examples

In this example, the Programmable Controller’s mode will be changed.

```
/********************************************/
/*       Mode change processing            */
/********************************************/
```

```
extern unsigned int pcopen( );
extern unsigned int pcclose( );
extern unsigned int pcstat( );

void main(void)
{
        int   ret;
        unsigned char ne=0;         /*   Network   address
*/
        unsigned char no=0;         /* Node address */
        unsigned int mode;          /* Mode */

        ret = pcopen( );
        printf(”pcopen( ) ret = %d\n”,ret);
        if (ret == 1) {
               printf(”Driver not installed \n”);
               exit( );
        }

        printf(”Mode change processing \n”);
        printf(”Network 0x%x\n”, ne);
        printf(”Node 0x%x\n”, no);
         printf(”    Programmable Controller’s mode will
be changed \n\n”);
        printf(”0: PROGRAM Mode (trace) \n”);
        printf(”1: DEBUG Mode \n”);
        printf(”2: MONITOR Mode \n”);
        printf(”3: RUN Mode \n”);
        printf(”Change to which mode?”);
        scanf(”%d”,&mode);

        switch (pcmode(ne, no, mode)) {
               case 0:
                       printf(”PCMODE normal \n”);
                       break;
               case 1:
                       printf(”PCOPEN not executed \n”);
                       break;
               case 2:
                              printf(”Network address is
invalid \n”);
                       break;
               case 3:
                     printf(”Programmable Controller busy
\n”);
                       break;
               case 5:
                         printf(”Improper argument passed
\n”);
                       break;
               case 7:
                       printf(”Ended abnormally \n”);
                       break;
        }
        pcclose( );
}
```

pcclose( ) CLOSE CPU BUS

Purpose

This function terminates usage of the CPU bus.

Format

```
unsigned int pcclose( )
```

Parameters

None

Comments

Use this function to close the CPU bus. When quitting from within the program, this command must be executed.

Returned Values

Value	Meaning
0	Ended normally.
1	Bus is not open.

Related Functions

pcopen

Program Examples

For a program example, refer to page 46, *pcopen*.

SECTION 4
CPU Bus Driver

The CPU Bus Driver provides efficient communications through the CPU bus. This section introduces the CPU Bus Driver and describes the FINS commands used in the CPU Bus Driver.

4-1 Introduction 78
4-2 The FINS Format 78
4-2-1 FINS Commands 79
4-2-2 FINS Response 80
4-2-3 FINS Command/Response Example 81
4-3 Using the CPU Bus Driver 82
4-3-1 Access by Opening a Driver 83
4-3-2 Direct Request 84
4-3-3 Limitations 84
4-4 CPU Bus Driver Operations List 85
4-5 CPU Bus Driver Operations 85
4-5-1 Cyclic Service Transmission Address, Length, and Direction 85
4-5-2 Reading the Cyclic Area 86
4-5-3 Writing to the Cyclic Area 87
4-5-4 Transmitting FINS Commands 88
4-5-5 Transmitting FINS Responses 89
4-5-6 Receiving FINS Commands 90
4-5-7 Receiving FINS Responses 91
4-5-8 Branching Upon Completion of FINS Command Transmission 92
4-5-9 Branching Upon Completion of FINS Response Transmission 92
4-5-10 Branching Upon Completion of FINS Command Reception 93
4-5-11 Branching Upon Completion of FINS Response Reception 93
4-5-12 Setting Timeout Values 94
4-5-13 Flushing Reception Buffers 94
4-5-14 Reading Information Reserved for System 94
4-5-15 Reading the Link Area 95
4-5-16 Writing to the Link Area 96
4-5-17 User Timer Service Processing 97
4-5-18 Resetting the Personal Computer Unit 97
4-5-19 Unit Address Inquiry 97
4-5-20 Reception Status Inquiry 98
4-6 FINS Commands Serviced by Drivers 98
4-6-1 Read Controller Information (0501) 99
4-6-2 Read Time Information (0701) 100
4-6-3 Write Time Information (0702) 100
4-6-4 Loopback Test (0801) 101
4-6-5 Read Error Log (2102) 102
4-6-6 Clear Error Log (2103) 103
4-7 Sample Programs 104
4-8 Measuring CPU Bus Access Performance 117
4-8-1 Cyclic Service 118
4-8-2 Event Service 118
4-8-3 CPU Bus Link Service 119

4-1 Introduction

In order to conduct efficient communications using services via the CPU bus interface, the user can employ the CPU Bus Driver.

When this driver is installed, the three types of service listed below can be received. Thus the user can conduct communications without being aware of the CPU bus interface.

Name	Explanation
Cyclic service	Communication possible with Programmable Controller in the same node.
CPU bus link service	Communication possible with Programmable Controller and other devices in the same node.
Event service	Communication possible with Programmable Controller and other devices in the same node, as well as with other Programmable Controllers and devices throughout the same network and other networks.

Refer to *1-4 Communications/Control Service Details* for more details on these services.

Installing the CPU Bus Driver

The CPU Bus Driver (SBUS.SYS) is recorded in the CONFIG.SYS file in drive F in advance. Be sure that SBUS.SYS is recorded in the active CONFIG.SYS file when a system disk is being modified or a new CONFIG.SYS file is being created.

Add the following line to the CONFIG.SYS file to record the CPU Bus Driver. Refer to *2-6 Installing Device Drivers* for details on other SBUS.SYS parameters. (The F:\CONFIG.SYS file is used when the Unit is started from Built-in ROM.)

```
DEVICE=E:\SBUS.SYS /V65
```

Drive name

The CPU Bus Driver is recorded in the Personal Computer Unit's Built-in ROM (drive E) in advance.

4-2 The FINS Format

This section explains the FINS command/response format used for event servicing when the CPU Bus Driver is used. Event service can communicate with other devices on a network. To communicate with a particular device, it is necessary to specify that device's network address, node address, and Unit number address.

In addition, when receiving a command from another device, it will not be possible to return a response unless it is known where the command was transmitted from. A FINS (Factory Interface Network System) format is thus used with this driver to specify the transmission source and destination. Refer to the*FINS Command Reference Manual (W227)* for more details.

This format incorporates the information essential for communications in front of the command or response data that is to be actually transmitted. Based on this information, the transmission destination can be determined and the transmission source can be confirmed.

Note When using event service with this driver, edit the data in FINS format and leave the processing to the driver. The maximum length of data that can be processed by event service is the total length of the FINS format.

From this point on, the explanation of the FINS format, and particularly the designation of the transmission destination, will be divided into command format (FINS command) and response format (FINS response).

4-2-1 FINS Commands

The FINS commands have the data format shown below, and the transmission destination and data to be transmitted are specified according to this format. Command transmission and reception are always executed in this format.

1B	1B	1B	1B	1B	1B	
ICF	RSV	GCNT	DNA	DA1	DA2	

	1B	1B	1B	1B	1B	1B	
	SNA	SA1	SA2	SID	MRC	SRC	DATA

ICF

Information Control Field:

Specifies contents for controlling FINS command. This field indicates whether a response should be sent from the destination device. A "0" requests a response; a "1" indicates a response is not required.

RSV

Reserve:

Reserved area. As a rule, the contents will be $00.
This area is sometimes used in communications with SYSMAC BUS/2 and BASIC Units.

GCNT

Number of Times Through Gateway (Bridge):

The gateway cannot be passed through more than this number of times.

Note When a command is transmitted from the Personal Computer Unit, this field is set by the driver and cannot be set by the user.

DNA

Destination Network Address:

This specifies the network address of the transmission destination. This address is the final target location address. (A setting of $00 indicates that the destination network is the same as the source network.)

$00: Same network address

DA1

Destination Node Address:

This specifies the node address of the transmission destination. This address is the final target location address. The following codes have special meanings.

$00: Same node address
$FF: Broadcast to all nodes on specified network

DA2

Destination Unit Address:

The unit address for the transmitted data is specified by the absolute address. (Add $10 to the unit number to calculate the absolute address.)

Example: Special I/O Unit #0 will have an address of $10.

This address is the final target location address. The following codes have special meanings.

`$00`	Unit address of Programmable Controller
`$10 to $2F`	CPU Bus Units
`$FD`	Peripheral Tools (e.g., FIT)
`$FE`	Communications Units (e.g., SYSMAC NET, SYSMAC LINK)

SNA

Transmission Source Network Address:

This specifies the network address of the source of the transmitted data.

Note When a command is transmitted from the Personal Computer Unit, this field is set by the driver and cannot be set by the user.

SA1

Transmission Source Node Address:

This specifies the node address of the source of the transmitted data.

Note When a command is transmitted from the Personal Computer Unit, this field is set by the driver and cannot be set by the user.

SA2

Transmission Source Unit Address:

This specifies the unit address of the source of the transmitted data.

Note When a command is transmitted from the Personal Computer Unit, this field is set by the driver and cannot be set by the user.

SID

Service ID:

The same service ID number is used in a command and the response to that command. Use the service ID to distinguish between several commands and responses.

MRC

Main Request Class:

This classifies the service. Refer to the *FINS Command Reference Manual (W227)* for details.

SRC

Sub-request Class:

This specifies the service details. Refer to the *FINS Command Reference Manual (W227)* for details.

DATA

Data:

This is the data determined by MRC and SRC. Refer to the*FINS Command Reference Manual (W227)* for details.

4-2-2 FINS Response

When a FINS command is issued to another device, a corresponding response can be returned from that device. In addition, when a command requiring a response is transmitted to the Personal Computer Unit from another device, a response must be returned to that device from the Personal Computer Unit. The data format for the response is as shown below.

1B	1B	1B	1B	1B	1B	1B	1B	
ICF	RSV	GCNT	DNA	DA1	DA2	SNA	SA1	

	1B	1B	1B	1B	1B	1B	
	SA2	SID	MRC	SRC	MRES	SRES	DATA

ICF

Information Control Field:

Specifies contents for controlling FINS command.

When a command is transmitted from the Personal Computer Unit, this field is set by the driver and cannot be set by the user.

RSV

Reserve:

Reserved area

Note When a response is transmitted from the Personal Computer Unit, specify the same contents as for the RSV of the FINS command that was received.

GCNT

Number of Times Through Gateway (Bridge):

The gateway cannot be passed through more than this number of times.

Note When a command is transmitted from the Personal Computer Unit, this field is set to 2 by the driver and cannot be set by the user.

DNA to SA2

Just as for the FINS command described above, this specifies the transmission destination and source addresses.

Note When a response is transmitted from the Personal Computer Unit, replace the following items of the FINS command that was received and specify the response data.

DNA ↔ SNA:	Network address
DA1 ↔ SA1:	Node address
DA2 ↔ SA2:	Unit address

SID

Service ID:

This specifies an identifier for recognizing request source processes.

Note When a response is transmitted from the Personal Computer Unit, specify the same contents as for the SID of the FINS command that was received.

MRC

Main Request Class:

This specifies the service classification.

Note When a response is transmitted from the Personal Computer Unit, specify the same contents as for the MRC of the FINS command that was received.

SRC

Sub-request Class:

This specifies the service details.

Note When a response is transmitted from the Personal Computer Unit, specify the same contents as for the SRC of the FINS command that was received.

MRES

Main Response Code:

This code indicates the response (i.e., normal, error, error details) to a request. Refer to the *FINS Command Reference Manual (W227)* for details. When returning a response from the Personal Computer Unit, enter the result of the command execution.

SRES

Sub-response Code:

This code indicates detailed information that cannot be adequately expressed by the main response code. Refer to the *FINS Command Reference Manual (W227)* for details. When returning a response from the Personal Computer Unit, enter the result of the command execution.

DATA

Data:

This is the part of the service indicating the results. Refer to the*FINS Command Reference Manual (W227)* for details.

4-2-3 FINS Command/Response Example

Example: Reading PC Time Information in the Same Node

The following FINS command is transmitted to a Programmable Controller address within the same node.

ICF	RSV	GCNT	DNA	DA1	DA2
$00	$00	$00	$00	$00	$00

SNA	SA1	SA2	SID	MRC	SRC
$00	$00	$00	$01	$07	$01

The following response is returned to the Personal Computer Unit from the Programmable Controller in the same node. The Programmable Controller time information can be recognized by analysis of these codes. In this case, the Personal Computer Unit's unit number is 0.

ICF	RSV	GCNT	DNA	DA1	DA2	SNA	SA1
$41	$00	$02	$00	$00	$10	$00	$00

SA2	SID	MRC	SRC	MRES	SRES
$00	$01	$07	$01	$00	$00

Year	Month	Date	Hour	Minute	Second	Day
$94	$10	$01	$12	$34	$00	$03

Programmable Controller time information:
1994, October 1, Sunday, 12:34:00

The DA2 value of $10 indicates that the response is addressed to the Personal Computer Unit, and the SID value of $01 indicates that the response corresponds to the command shown above. The MRES and SRES values of $00 are the normal response codes. Refer to the *FINS Command Reference Manual (W227)* for more details.

4-3 Using the CPU Bus Driver

There are two methods, described below, for using the CPU Bus Driver.

Opening and Accessing the Driver Through MS-DOS

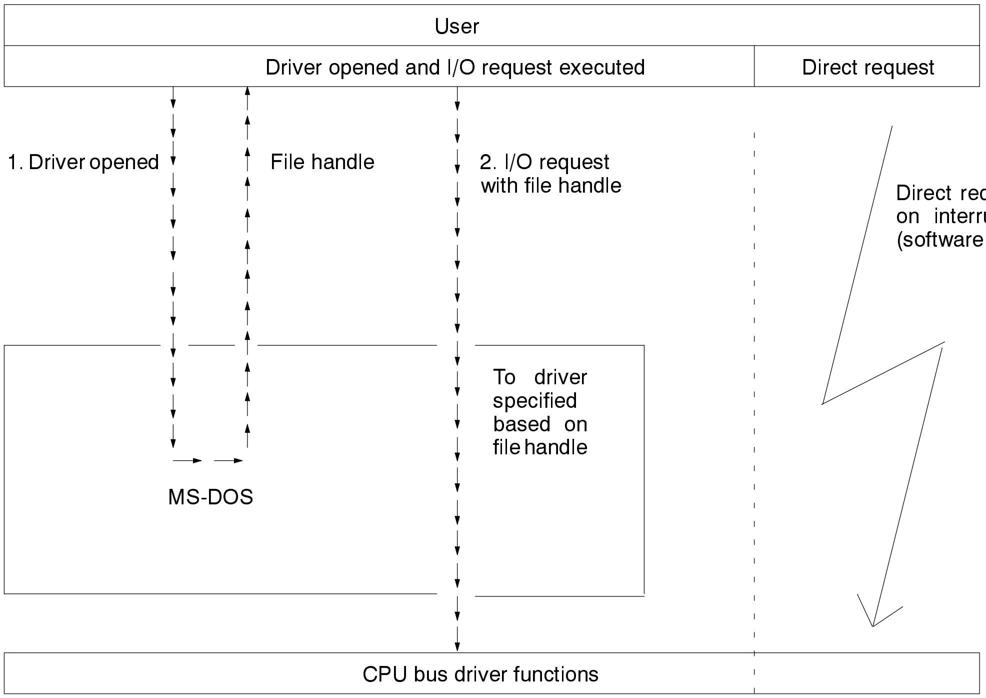
First of all, the driver is opened through MS-DOS, and the “file handle” file identifier is obtained.

When the driver services are employed, I/O requests are executed based on this file handle.

Direct Request Without Going Through MS-DOS

With this method, driver services are used without going through MS-DOS, but rather by executing system interrupts directly.

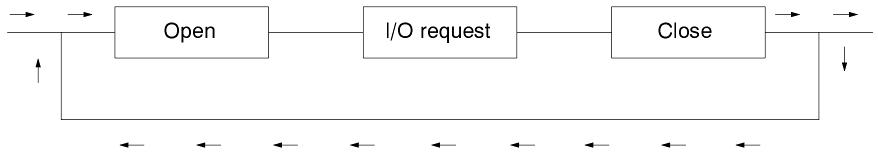
Differences Between I/O Requests by Opening Driver and Direct Requests


User
Driver opened and I/O request executed
Direct request
1. Driver opened
File handle
2. I/O request with file handle
Direct request based on interrupt number (software interrupt)
To driver specified based on file handle
MS-DOS
CPU bus driver functions

Additional Information **Opening and Closing**

In our own lives, let us consider how we utilize a book. First of all, from among a large number of books, we must get the book that contains the material that we want to utilize. Once we have the book, we look it over and extract that portion of the contents that we need. If the material is not needed, we return the book to its original spot and the contents of the book then cannot be utilized.

The situation is much the same with a driver. If we want to utilize a driver through MS-DOS, first we must get (or, in other words, open) the driver. Then, by means of a file identifier that has been obtained, the driver is specified and utilized. When the operation is finished, the driver is closed. Once the driver has been closed, it can no longer be used and must be reopened in order to be used again.



These respective methods are described below.

4-3-1 Access by Opening a Driver

Opening the Driver

Finds the file handle.

Call Procedure:

AH = 3DH
AL = 02H (File access mode)
DS:DX = Leading address of path name (“CVIF”)
INT 21H

Return:

When Carry is Set (Error)

AX = 02H	File does not exist.
AX = 03H	Path name is invalid.
AX = 04H	There are too many files open.
AX = 05H	Access was denied.
AX = 0CH	Access code is invalid.

When Carry is Not Set
AX = File handle (normal termination)

Closing the Driver

Closes the driver.

Call Procedure:

AH = 3EH
BX = File handle
INT 21H

Return:

When Carry is Set (Error)

AX = 06H	File handle is invalid.

When Carry is Not Set

AX = 00H	Normal termination

Requesting I/O

Prepares the prescribed I/O parameters, and makes the request using the file handle.

Call Procedure:

AH = 44H
AL = 03H

BX = File handle
CX = Number of bytes in data buffer
DS:DX = Data buffer segment: offset
(Refer to *4-5 CPU Bus Driver Operations* for details on the data buffer.)
INT 21H

Return:

When Carry is Set (Error)
AX = 01H Function is invalid.
AX = 05H Access was denied.
AX = 06H File handle is invalid.
AX = 0DH Data is invalid (data buffer error).

When Carry is Not Set
AX = 00H Normal termination
AX = FFH Improper command
AX = Other Error status (Refer to *4-5 CPU Bus Driver Operations* for details.)

4-3-2 Direct Request

Prepares the prescribed I/O parameters, and makes the request using the interrupt table.

Call Procedure:

AH = xxH Operation number (Refer to *4-4 CPU Bus Driver Operations List* for details.)
CX = Number of bytes in data buffer
DS:DX = Data buffer segment: offset
(Refer to *4-5 CPU Bus Driver Operations* for details on the data buffer.)
INT xxH Empty interrupt table number (60H to 65H)
(The table to be used is specified by CONFIG.SYS.)

Return:

When Carry is Set (Error)
AX = 01H Function is invalid.
AX = 05H Access was denied.
AX = 0DH Data is invalid (data buffer error).

When Carry is Not Set
AX = 00H Normal termination
AX = Other Error status (Refer to *4-5 CPU Bus Driver Operations* for details.)

4-3-3 Limitations

Each service has the following timing standards. Refer to *4-8 Data Transmission Timing* for details.

Service	Time required
Cyclic service	7 ms (when 256 words are read or written)
Event service	20 ms from issuance of FINS command to receipt of the response (when 997 DM words are read)
CPU Bus Link service	10 ms (when there are no other CPU Bus Link Units)

Do not mask the interrupts (IRQ10 and IRQ11) when the event service's functions are used.

4-4 CPU Bus Driver Operations List

Number	Service	Summary
01H	Cyclic service	Specifies the area to be read or written.
02H		Reads specified area.
03H		Writes to specified area.
04H	Event service	Transmits FINS command.
05H		Transmits FINS response.
06H		Transmits request to receive FINS command.
07H		Transmits request to receive FINS response.
08H		Saves branch entry address when transmission of FINS command is complete.
09H		Saves branch entry address when transmission of FINS response is complete.
0AH		Saves branch entry address when reception of FINS command is complete.
0BH		Saves branch entry address when reception of FINS response is complete.
0CH		Sets reception timeout value.
0DH		Flushes reception buffer.
0EH	CPU Bus Link service	Reads information reserved for system.
0FH		Reads link data.
10H		Writes link data.
11H	Other services	Branches to specified entry address after time specified by user has elapsed.
12H		Resets Personal Computer Unit.
13H		Inquires regarding address of Unit itself.
14H		Inquires regarding reception status of FINS commands and responses.

4-5 CPU Bus Driver Operations

This section will explain CPU Bus Driver operations, including contents of data buffers when these operations are used, as well as the return values, for each of the operations. The command numbers given in this section will match the operation numbers given in *4-4 CPU Bus Driver Operations List*.

4-5-1 Cyclic Service Transmission Address, Length, and Direction

Parameter Format

0	(Command) 01
1	Reserve (00)
2	Transmission direction
3	Cyclic area set number (1 to 6)
4	PC's real space address (least-significant byte)
5	
6	
7	(most-significant byte)
8	Transmission length
9	

Transmission Direction:

This sets the direction in which data will be transmitted.

0: Programmable Controller → Personal Computer Unit
Other than 0: Personal Computer Unit → Programmable Controller

Cyclic Area Set Number:

This is the number for specifying the cyclic area that is to set the transmission status. It can be set within a range of 1 to 6.

PC's Real Space Address:

This is the real space address of the Programmable Controller that is to execute cyclic service. It can be set within a range of $400000 to $4FFFFF. For the area, IOM, DM, or EM can be specified, but UM cannot.

Transmission Length:

This is the cyclic area transmission length. It is specified in word units. An error will be generated if the sum of all the transmission lengths allocated to the set numbers exceeds 3,987 words. The default settings will be as follows:

Bit:	Number 1
PC → Personal Computer Unit:	15 words ($400BB8)
Personal Computer Unit → PC:	10 words ($400BD6)

If "0" is set, the setting for the specified cyclic area will be cleared.

Return Status (AX)

00H:	Normal termination
01H:	Incorrect set number
02H:	Incorrect real space address
03H:	Total transmission length overflow (3,987 words max.)
0AH:	Memory access error

Operation

This operation sets the address and transmission length for the Programmable Controller executing cyclic service. A total of 12 places can be set simultaneously, including six for transmission from the Programmable Controller to the Personal Computer Unit and six for transmission from the Personal Computer Unit to the Programmable Controller. When the setting is made, it will overwrite the previous status of the relevant specified bit number and transmission length.

This operation only sets the status of the cyclic service, and does not read or write data. Once these settings are made, they are valid until the Personal Computer Unit is reset; they aren't cleared by the closing processes.

4-5-2 Reading the Cyclic Area

Parameter Format

0	(Command) 02
1	Reserve (00)
2	Cyclic area set number (1 to 6)
3	Reserve (00)
4	Cyclic data reception buffer address
5	Offset
6	Cyclic data reception buffer address
7	Segment
8	Number of words requested to be read
9	
10	Number of words actually read
11	

Cyclic Area Set Number:

This is the number for specifying the cyclic area that is to read data. It can be set within a range of 1 to 6.

Cyclic Data Reception Buffer Address:

This is the leading address of the buffer for storing cyclic data that is read.

Number of Words Requested to be Read:

This is the number of words of cyclic data requested to be read.

Number of Words Actually Read:

This is the area for storing the number of words that are actually read.

Return Status (AX)

00H:	Normal termination
01H:	Set number incorrect
02H:	Reception buffer address incorrect
03H:	Number of words requested incorrect
0AH:	Memory access error
0BH:	Programmable Controller busy
0CH:	Parity error

Operation

From the real space address of the Programmable Controller set by the specified cyclic area set number, the number of words requested will be read and then stored in the specified reception buffer. At that time, the actual number of words read will be stored in the area specified for that purpose.

If the specified number of words to be read should exceed the number of words set for the cyclic area, then code 03H (number of words requested incorrect) will be generated. The only valid areas are ones that have had the transmission direction set to Programmable Controller → Personal Computer Unit (0) with operation number 01.

4-5-3 Writing to the Cyclic Area

Parameter Format

0	(Command) 03
1	Reserve (00)
2	Cyclic area set number (1 to 6)
3	Reserve (00)
4	Cyclic data storage buffer address
5	Offset
6	Cyclic data storage buffer address
7	Segment
8	Number of words requested to be written
9	
10	Number of words actually written
11	

Cyclic Area Set Number:

This is the number for specifying the cyclic area for writing data. It can be set within a range of 1 to 6.

Cyclic Data Reception Buffer Address:

This is the leading address of the buffer for storing cyclic data that is written.

Number of Words Requested to be Written:

This is the number of words of cyclic data requested to be written.

Number of Words Actually Written:

This is the area for storing the number of words that are actually written.

Return Status (AX)

00H:	Normal termination
01H:	Set number incorrect
02H:	Data storage buffer address incorrect
03H:	Number of words requested incorrect
0AH:	Memory access error
0BH:	Programmable Controller busy

Operation

The data indicated by the data storage buffer address will be written to the real space address of the Programmable Controller set by the specified cyclic area set number, for the number of words requested. At that time, the actual number of words written will be stored in the area specified for that purpose.

If the specified number of words to be written should exceed the number of words set for the cyclic area, then code 03H (number of words requested incorrect) will be generated. The only valid areas are ones that have had the transmission direction set to Personal Computer Unit → Programmable Controller (nonzero) with operation number 01.

4-5-4 Transmitting FINS Commands

Parameter Format

0	(Command) 04
1	Reserve (00)
2	FINS command storage buffer address Offset
3	
4	FINS command storage buffer address Segment
5	
6	Number of bytes requested for transmission
7	
8	Number of bytes actually transmitted
9	

FINS Command Storage Buffer Address:

This is the leading address of the buffer for storing transmitted FINS commands.

Number of Bytes Requested for Transmission:

This is the number of bytes requested for the FINS command. Specifies the total number of bytes from the ICF.

Number of Bytes Actually Transmitted:

This is the area for storing the number of bytes actually transmitted.

Return Status (AX)

00H:	Normal termination
01H:	Command storage buffer address incorrect
02H:	Number of words requested incorrect (i.e., fewer than 14 or more than 2012 bytes)
03H:	Transmission destination network address incorrect
0AH:	Memory access error
0BH:	Programmable Controller busy error Retry transmission or increase the number of retries. (Refer to *2-6 Installing Device Drivers* for details on the number of retries.)

Operation

The data stored in the FINS command storage buffer address (in conformity with Programmable Controller FINS commands) is analyzed, the corresponding ICF, GCNT, SNA, SA1, and SA2 are set, and the number of bytes requested is transmitted to the specified address. At that time, the actual number of words written will be stored in the area specified for that purpose. A response to the transmitted FINS command can be received by means of a “receive FINS response” request.

If “branch entry address upon completion of command transmission” is set, then control will be transferred to the specified address after the transmission is complete.

4-5-5 Transmitting FINS Responses

Parameter Format

0	(Command) 05
1	Reserve (00)
2	FINS response storage buffer address
3	Offset
4	FINS response storage buffer address
5	Segment
6	Number of bytes requested for transmission
7	
8	Number of bytes actually transmitted
9	

FINS Command Storage Buffer Address:

This is the leading address of the buffer for storing transmitted FINS responses.

Number of Bytes Requested for Transmission:

This is the number of bytes requested for the FINS response. Specifies the total number of bytes from the ICF.

Number of Bytes Actually Transmitted:

This is the area for storing the number of bytes actually transmitted.

Return Status (AX)

00H: Normal termination
01H: Command response storage buffer address incorrect
02H: Number of words requested incorrect
(i.e., fewer than 14 or more than 2012 bytes)
03H: Transmission destination network address incorrect
0AH: Memory access error
0BH: Programmable Controller busy error
Retry transmission or increase the number of retries. (Refer to *2-6 Installing Device Drivers* for details on the number of retries.)

Operation

The data stored in the FINS command storage buffer address is analyzed, the corresponding ICF and GCNT are set, and the number of bytes requested is transmitted to the specified address. At that time, the actual number of words written will be stored in the area specified for that purpose.

This transmission request is used when a FINS command transmitted from another device is received and a response is transmitted to the source of the transmitted command. To receive the FINS command from the other device, use a “receive FINS command” request.

If “branch entry address upon completion of command transmission” is set, then control will be transferred to the specified address after the transmission is complete.

4-5-6 Receiving FINS Commands

Parameter Format

0	(Command) 06
1	Timeout processing
2	FINS command storage buffer address
3	Offset
4	FINS command storage buffer address
5	Segment
6	Number of bytes requested for transmission
7	
8	Number of bytes actually transmitted
9	

Timeout Processing:

0: Waits to receive until the timeout value has elapsed.
(The timeout value is set with command 0C.)

Other than 0: Waits to receive until Esc Key is pressed.

FINS Command Reception Buffer Address:

This is the leading address of the buffer for storing FINS commands that are received.

Number of Bytes Requested for Reception:

This is the number of bytes requested for receiving a FINS command.

Number of Bytes Actually Received:

This is the area for storing the number of actual bytes stored in the reception buffer.

Return Status (AX)

00H:	Normal termination
01H:	Command reception buffer address incorrect
02H:	Number of bytes requested incorrect (i.e., fewer than 14 or more than 2012 bytes)
0AH:	Memory access error
0CH:	Parity error
0EH:	Forcibly ended by Escape Key
0FH:	Timeout error

Operation

A FINS command transmitted from another device is received, and the number of bytes requested is stored in the specified FINS command reception buffer. At that time, the actual number of bytes stored in the buffer will be stored in the area specified for that purpose.

This reception request is used when a FINS command transmitted from another device is received. To transmit a response to the data received, use a "transmit FINS response" request.

If no command is received from any device when this command reception request is made, the Unit will wait to receive a command until the timeout value set with command 0C has elapsed (if timeout processing has been specified) or until the Esc Key is pressed (if timeout processing hasn't been specified).

Timeout Processing Specified

Waiting-to-receive status will remain in effect until the time period registered with command 0C has elapsed. If a command is received during that period, the data that is received will be stored in the specified buffer. If no command is received, a timeout error will occur.

Timeout Processing Not Specified

Waiting-to-receive status will remain in effect regardless of the time period registered with command 0C. If a command is received, the data that is received will be stored in the specified buffer. An exit can be forced during that period by pressing the Escape Key.

If even a portion of the command data that has been received is read, the command data will be cleared.

4-5-7 Receiving FINS Responses

Parameter Format

0	(Command) 07
1	Timeout processing
2	FINS response reception buffer address Offset
3	
4	FINS response reception buffer address Segment
5	
6	Number of bytes requested for transmission
7	
8	Number of bytes actually transmitted
9	

Timeout Processing:

0: Waits to receive until timeout value has elapsed.
(The timeout value is set with command 0C.)

Other than 0: Waits to receive until Esc Key is pressed.

FINS Response Reception Buffer Address:

This is the leading address of the buffer for storing FINS responses that are received.

Number of Bytes Requested for Reception:

This is the number of bytes requested for receiving a FINS response.

Number of Bytes Actually Received:

This is the area for storing the number of actual bytes stored in the reception buffer.

Return Status (AX)

00H:	Normal termination
01H:	Command response reception buffer address incorrect
02H:	Number of bytes requested incorrect (i.e., fewer than 14 or more than 2012 bytes)
0AH:	Memory access error
0BH:	Parity error
0EH:	Forcibly ended by Escape Key
0FH:	Timeout error

Operation

A FINS response to a service requested of another device is received, and the number of bytes requested for receiving the response is stored in the specified FINS response reception buffer. At that time, the actual number of bytes stored in the buffer will be stored in the area specified for that purpose.

This reception request is used when a FINS response to a service requested of another device is received. To request a service of another device, use a “transmit FINS command” request.

If no command is received from any device when this command reception request is made, the Unit will wait to receive a command until the timeout value set with command 0C has elapsed (if timeout processing has been specified) or until the Esc Key is pressed (if timeout processing hasn’t been specified).

Timeout Processing Specified

Waiting-to-receive status will remain in effect until the time period registered with command 0C has elapsed. If a response is received during that period, the data that is received will be stored in the specified buffer. If no command is received, a timeout error will occur.

Timeout Processing Not Specified

Waiting-to-receive status will remain in effect regardless of the time period registered with command 0C. If a response is received, the data that is received will be stored in the specified buffer. An exit can be forced during that period by pressing the Escape Key.

If even a portion of the command data that has been received is read, the command data will be cleared.

4-5-8 Branching Upon Completion of FINS Command Transmission

Parameter Format

0	(Command) 08
1	Reserve (00)
2	Entry address upon completion of command transmission
3	Offset
4	Entry address upon completion of command transmission
5	Segment

Entry Address Upon Completion of Command Transmission:

This is the leading address of the process that will be executed when the command transmission has been completed.

Return Status (AX) 00H: Normal termination

Operation This operation registers the user entry address for branching when the FINS command transmission has been completed. If this address is set to "0," the registered user entry address will be cleared.

Note

1. For user routines, use the same stack area.
2. When user routine processing has been completed, return control with FAR RET.
3. In the process that is executed, the ds register points out the data segment of the driver. The register will be returned to its original status after the process is completed.

4-5-9 Branching Upon Completion of FINS Response Transmission

Parameter Format

0	(Command) 09
1	Reserve (00)
2	Entry address upon completion of response transmission
3	Offset
4	Entry address upon completion of response transmission
5	Segment

Entry Address Upon Completion of Response Transmission:

This is the leading address of the process that will be executed when the response transmission has been completed.

Return Status (AX) 00H: Normal termination

Operation

This operation registers the user entry address for branching when the FINS response transmission has been completed. If this address is set to “0,” the registered user entry address will be cleared.

Note
1. For user routines, use the same stack area.
2. When user routine processing has been completed, return control with FAR RET.
3. In the process that is executed, the ds register points out the data segment of the driver. The register will be returned to its original status after the process is completed.

4-5-10 Branching Upon Completion of FINS Command Reception

Parameter Format

0	(Command) 0A(H)
1	Reserve (00)
2	Entry address upon completion of command transmission
3	Offset
4	Entry address upon completion of command transmission
5	Segment

Entry Address Upon Completion of Command Reception:

This is the leading address of the process that is executed when the command reception has been completed.

Return Status (AX) 00H: Normal termination

Operation

This operation registers the user entry address for branching when the FINS command reception has been completed. If this address is set to “0,” the registered user entry address will be cleared.

Note
1. For user routines, use the same stack area.
2. When user routine processing has been completed, return control with FAR RET.
3. In the process that is executed, the ds register points out the data segment of the driver. The register will be returned to its original status after the process is completed.

4-5-11 Branching Upon Completion of FINS Response Reception

Parameter Format

0	(Command) 0B(H)
1	Reserve (00)
2	Entry address upon completion of response transmission
3	Offset
4	Entry address upon completion of response transmission
5	Segment

Entry Address Upon Completion of Response Reception:

This is the leading address of the process that is executed when the response reception has been completed.

Return Status (AX) 00H: Normal termination

Operation

This operation registers the user entry address for branching when the FINS response reception has been completed. If this address is set to “0,” the registered user entry address will be cleared.

Note

1. For user routines, use the same stack area.
2. When user routine processing has been completed, return control with FAR RET.
3. In the process that is executed, the ds register points out the data segment of the driver. The register will be returned to its original status after the process is completed.

4-5-12 Setting Timeout Values

Parameter Format

0	(Command) 0C(H)
1	Reserve (00)
2	Timeout value
3	

Timeout Value:

This is the timeout value used when data reception is requested. The setting can be made within a range of 0 to 65,535 (in units of 110 ms).

Return Status (AX) 00H Normal termination

Operation This operation registers the time period for waiting-to-receive when a FINS command or a FINS response reception is requested. The timeout value is set in units of 110-ms. If “0” is registered, the Unit will not wait to receive the transmission. The default for the timeout value is 0 ms.

4-5-13 Flushing Reception Buffers

Parameter Format

0	(Command) 0D(H)
1	Reserve (00)
2	Parameters
3	

Parameters:

Specifies which reception buffers are to be flushed. (The reception buffers are allocated by the driver.)

0: Reception buffer for command data
1: Reception buffer for response data
2: Reception buffer for both command and response data

Return Status (AX):

00H: Normal termination
01H: Parameter values incorrect

Operation This operation flushes (clears) the specified reception buffers, according to the specified parameters.

4-5-14 Reading Information Reserved for System

Parameter Format

0	(Command) 0E(H)
1	Reserve (00)
2	System-reserved information reception buffer address
3	Offset
4	System-reserved information reception buffer address
5	Segment

System-reserved Information Reception Buffer Address:

This is the leading address of the buffer for storing system-reserved information that is read.

Return Status (AX)

00H: Normal termination
01H: Reception buffer address incorrect
0AH: Memory access error
0CH: Parity error

Operation

This operation reads information reserved for the system and stores it in the specified reception buffer. The system-reserved information is configured in eight words, so fixed 8-word blocks are read. This operation can be performed even when the CPU bus link service is stopped.

Refer to *1-4-3 CPU Bus Link Service* for details on the system-reserved information.

4-5-15 Reading the Link Area

Parameter Format

0	(Command) 0F(H)
1	Reserve (00)
2	Beginning word for reading CPU bus link area
3	
4	Read data reception buffer address Offset
5	
6	Read data reception buffer address Segment
7	
8	Number of words requested to be read
9	
10	Number of words actually read
11	

Beginning Word for Reading CPU Bus Link Area:

This is the beginning word of the CPU bus link area for reading data. (It can be set within a range of word 0 to word 255.)

Read Data Reception Buffer Address:

This is the leading address of the buffer for storing the link data that is read.

Number of Words Requested to be Read:

This is the number of words of link data that is requested to be read.

Number of Words Actually Read:

This is the area for storing the number of words that is actually read.

Return Status (AX)

00H: Normal termination
01H: Beginning word incorrect
02H: Reception buffer address incorrect
03H: Number of words requested incorrect
0AH: Memory access error
0CH: Parity error

Operation

From the CPU bus link area address corresponding to the specified beginning word, the number of words requested will be read and then stored in the specified reception buffer. At that time, the actual number of words read will be stored in the area specified for that purpose. If the specified number of words to be read should exceed the CPU bus link area, then code 03H (number of words requested incorrect) will be generated.

4-5-16 Writing to the Link Area

Parameter Format

0	(Command) 10(H)
1	Reserve (00)
2	Beginning word for writing to CPU bus link area
3	
4	CPU bus link data storage buffer address Offset
5	
6	CPU bus link data storage buffer address Segment
7	
8	Number of words requested to be written
9	
10	Number of words actually written
11	

Beginning Word for Writing to CPU Bus Link Area:

This is the beginning word of the CPU bus link area for writing data. (It can be set within a range of word 0 to word 7.)

CPU Bus Link Data Storage Buffer Address:

This is the leading address of the buffer where the CPU bus link data that is to be written is stored.

Number of Words Requested to be Written:

This is the number of words of link data that is requested to be written.

Number of Words Actually Written:

This is the area for storing the number of words that is actually written.

Return Status (AX)

00H:	Normal termination
01H:	Beginning word incorrect
02H:	Data storage buffer address incorrect
03H:	Number of words requested incorrect
0AH:	Memory access error

Operation

The data indicated by the data storage buffer address will be written to the CPU bus link area address corresponding to the specified beginning word, for the number of words requested. At that time, the actual number of words written will be stored in the area specified for that purpose. If the specified number of words to be written should exceed the CPU bus link area, then code 03H (number of words requested incorrect) will be generated.

4-5-17 User Timer Service Processing

Parameter Format

0	(Command) 11(H)	
1	Reserve (00)	
2	Timer interrupt service Entry address	Offset
3		
4		Segment
5		
6	Timer/counter (110 ms)	
7		

Timer Interrupt Service Entry Address:

This is the leading address of the process that is started when the time period specified by the timer/counter has elapsed.

Timer/Counter:

This is the time period until the timer is started. It can be set within a range of 0 to 65,535, in 110-ms units.

Return Status (AX) 00H: Normal termination

Operation This operation registers the user entry address for starting after the time period specified by the timer/counter has elapsed. If this address is set to “0,” the registered user entry address will be cleared. If the user entry address is registered, timer monitoring will begin immediately.

Note

1. For user routines, use the same stack area.
2. When user routine processing has been completed, return control with FAR RET.
3. In the process that is executed, the ds register points out the data segment of the driver. The register will be returned to its original status after the process is completed.

4-5-18 Resetting the Personal Computer Unit

Parameter Format

0	(Command) 12(H)
1	Reserve (00)

Return Status (AX) 0AH: Memory access error

Operation This operation resets the Personal Computer Unit and is just like performing an AR reset from the PC. Cyclic service settings and CPU bus link service contents are initialized, and the routing table is read again.

4-5-19 Unit Address Inquiry

Parameter Format

0	(Command) 13(H)
1	Reserve (00)
2	Local unit address
3	Reserve (00)

Local Unit Address:

The unit address of the Personal Computer Unit is stored in this area.

Return Status (AX) 00H: Normal termination

Operation This operation returns the Personal Computer Unit’s (local unit’s) unit address.

4-5-20 Reception Status Inquiry

Parameter Format

0	(Command) 14(H)
1	Timeout processing

Timeout Processing:

0: Waits to receive until timeout value has elapsed.
(The timeout value is set with command 0C.)
Other than 0: Waits to receive until Esc Key is pressed.

Return Status (AX)

04H:	Reception command data present
05H:	Reception response data present
0EH:	Forcibly ended by Escape Key
0FH:	Timeout error

Operation

This operation inquires regarding the present status of event data reception. When both command data and event data have been received, the response data reception status is given priority in being returned.

If neither command data nor response data have been received, the Unit will wait to receive a command until the timeout value set with command 0C has elapsed (if timeout processing has been specified) or until the Esc Key is pressed (if timeout processing hasn't been specified).

Timeout Processing Specified
Waiting-to-receive status will remain in effect until the time period registered with command 0C has elapsed. If a command or response is received during that period, then that will be reported. If no command is received, a timeout error will occur.

Timeout Processing Not Specified
Waiting-to-receive status will remain in effect regardless of the time period registered with command 0C. If a command or response is received, then that will be reported. An exit can be forced during that period by pressing the Escape Key.

This operation only inquires regarding the reception status, and does not actually read the data. Therefore reception requests for commands or responses can be executed based on the returned values of this operation.

4-6 FINS Commands Serviced by Drivers

With event service, when a service is requested of the Personal Computer Unit by another device (i.e., when the Personal Computer Unit receives the command), the response processing will be executed in the CPU Bus Driver with respect to several commands, and a response will be returned to the source of the transmission.

Therefore, because the commands that the user can receive by means of a "command reception request" may be commands other than those processed in the CPU Bus Driver, it may be necessary for the user to execute the response processing and return the response to the source of the transmission.

The commands for which response processing is executed in the CPU Bus Driver are explained below, along with the response contents.

Note

1. The service request PDU is the format, from MRC onwards, of the FINS command created at the time of requesting a service.
2. The service response PDU is the format, from MRC onwards, of the FINS response when a response is transmitted or received with respect to a service request.

Commands for which response processing is executed by driver are as follows.

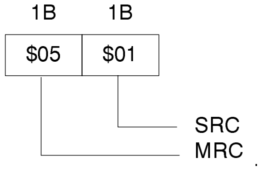
No.	MRC	SRC	Command contents
1	$05	$01	Read Controller Information
2	$07	$01	Read Time Information
3	$07	$02	Write Time Information
4	$08	$01	Loopback Test
5	$21	$02	Read Error Log
6	$21	$03	Clear Error Log

4-6-1 Read Controller Information (0501)

Operation

This operation inquires regarding the format and version of the Personal Computer Unit and the version of the driver.

Service Request PDU



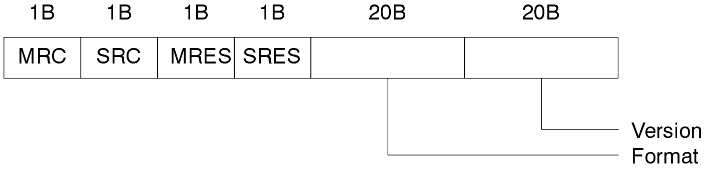
MRC (1 Byte):

Main Request Classification. Read Device Information is indicated by $05.

SRC (1 Byte):

Sub-request Classification. Read Controller Information is indicated by $01.

Service Response PDU



MRC (1 Byte):

Main Request Classification. The response to Read Device Information is indicated by $05.

SRC (1 Byte):

Sub-request Classification. The response to Read Controller Information is indicated by $01.

MRES (1 Byte):

Main Response Code. $00 is always returned.

SRES (1 Byte):

Sub-response Code. $00 is always returned.

Model (20 Bytes):

The Personal Computer Unit model is returned in ASCII code, with a maximum of 20 bytes. If the full 20 bytes is not needed, the remaining portion will be filled with spaces.

Version (20 Bytes):

The Personal Computer Unit OS and the CPU Bus Driver version are returned in ASCII code, with a maximum of 20 bytes. If the full 20 bytes is not needed, the remaining portion will be filled with spaces.

4-6-2 Read Time Information (0701)

Operation

This operation inquires regarding Personal Computer Unit's time information.

Service Request PDU

1B	1B
$07	$01

SRC
MRC

MRC (1 Byte):

Main Request Classification. Access Time Information is indicated by $07.

SRC (1 Byte):

Sub-request Classification. Read Time Information is indicated by $01.

Service Response PDU

1B	1B	1B	1B	1B	1B	1B	1B	1B	1B	1B
MRC	SRC	MRES	SRES	Year	Month	Date	Hour	Minute	Second	Day

MRC (1 Byte):

Main Request Classification. The response to Access Time Information is indicated by $05.

SRC (1 Byte):

Sub-request Classification. The response to Read Time Information is indicated by $01.

MRES (1 Byte):

Main Response Code. $00 is always returned.

SRES (1 Byte):

Sub-response Code. $00 is always returned.

Year (Two Rightmost Digits) to Second:

The time information that is read is returned as 1-byte pieces of BCD data.

Day:

Information on the day of the week is coded as follows:
$00=Sunday, $01=Monday, $02=Tuesday, $03=Wednesday,
$04=Thursday, $05=Friday, $06=Saturday

4-6-3 Write Time Information (0702)

Operation

This operation writes time information to the Personal Computer Unit.

Service Request PDU

1B	1B	1B	1B	1B	1B	1B	1B	1B
$07	$02	Year	Month	Date	Date	Hour	Minute	Second

SRC
MRC

MRC (1 Byte):

Main Request Classification. Access Time Information is indicated by $07.

SRC (1 Byte):

Sub-request Classification. Write Time Information is indicated by $02.

Year (Two Rightmost Digits) to Minute:

The time information to be written is indicated by 1-byte pieces of BCD data.

Second:

The second information to be written is indicated by 1-byte pieces of BCD data. This parameter is optional and can be omitted.

Day:

Information on the day of the week is coded as follows:

$00=Sunday, $01=Monday, $02=Tuesday, $03=Wednesday,
$04=Thursday, $05=Friday, $06=Saturday

This parameter is optional and can be omitted.

Service Response PDU

1B	1B	1B	1B
MRC	SRC	MRES	SRES

MRC (1 Byte):

Main Request Classification. The response to Access Time Information is indicated by $07.

SRC (1 Byte):

Sub-request Classification. The response to Write Time Information is indicated by $02.

MRES (1 Byte):

Main Response Code. $00 is returned when correct, and $11 when there is an error.

SRES (1 Byte):

Sub-response Code. The cause of the error is reported.

$00:	Normal termination
$01:	Command format error
$02:	Parameter error

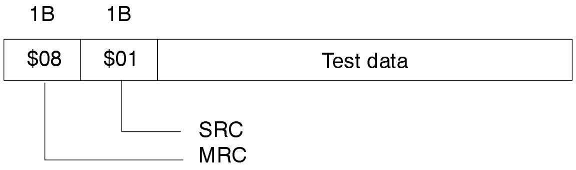
Note The Personal Computer Unit reads the the PC's clock and sets the time automatically when the CPU driver is installed.

4-6-4 Loopback Test (0801)

Operation

This operation indicates the connected devices, and conducts a loopback test with each of the devices.

Service Request PDU



1B	1B	
$08	$01	Test data

MRC (1 Byte):

Main Request Classification. Communications Tests are indicated by $08.

SRC (1 Byte):

Sub-request Classification. The Loopback Test is indicated by $01.

Test Data:

This is the arbitrary test code.

Service Response PDU

1B	1B	1B	1B	
MRC	SRC	MRES	SRES	Test data

MRC (1 Byte):

Main Request Classification. The response to Communications Tests is indicated by $08.

SRC (1 Byte):

Sub-request Classification. The response to Loopback Test is indicated by $01.

MRES (1 Byte):

Main Response Code. $00 is returned when correct, and $11 when there is an error.

SRES (1 Byte):

Sub-response Code. The cause of the error is reported.

$00:	Normal termination
$01:	Command format error

Test Data:

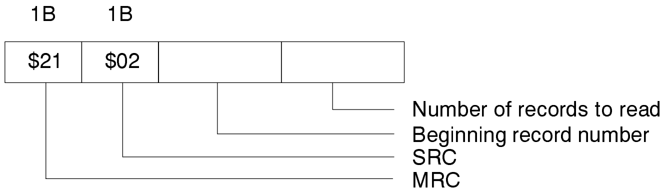
Data the same as that transmitted by service request is returned. The length of the test data is the same as the service request PDU.

4-6-5 Read Error Log (2102)

Operation

This operation inquires regarding error information generated at the Personal Computer Unit. The error information is erased from the system after it is read.

Service Request PDU



MRC (1 Byte):

Main Request Classification. Error Logging Operations are indicated by $21.

SRC (1 Byte):

Sub-request Classification. Read Error Log is indicated by $02.

Beginning Record Number (2 Bytes):

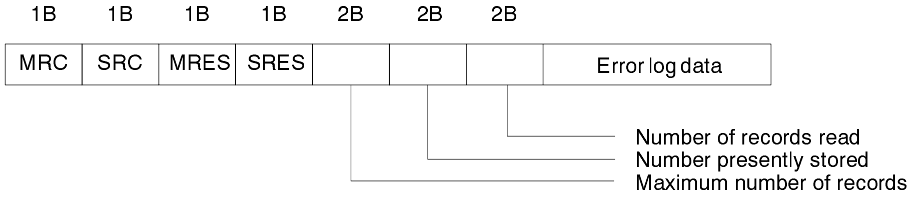
This indicates the first record number to be read. Each record consists of 10 bytes of data, and stores information on a single error.

The first record number is $0000. When the error history is read from a later record (greater than $0000), all earlier records will be erased.

Number of Records to Read (2 Bytes):

This indicates the number of records to be read.

Service Response PDU



MRC (1 Byte):

Main Request Classification. The response to Error Logging Operations is indicated by $21.

SRC (1 Byte):

Sub-request Classification. The response to Read Error Log is indicated by $02.

MRES (1 Byte):

Main Response Code. $00 is returned when correct, and $11 when there is an error.

SRES (1 Byte):

Sub-response Code. The cause of the error is reported.

$00: Normal termination
$01: Command format error
$02: Parameter error

Maximum Number of Records (2 Bytes):

This indicates the maximum number of error records that can be stored.

Number Presently Stored (2 Bytes):

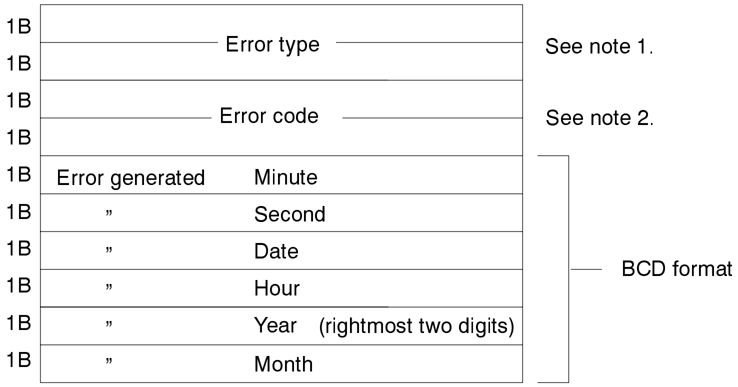
This indicates the number of items remaining after reading the error log.

Number of Records Read (2 Bytes):

The number of records actually read is returned.

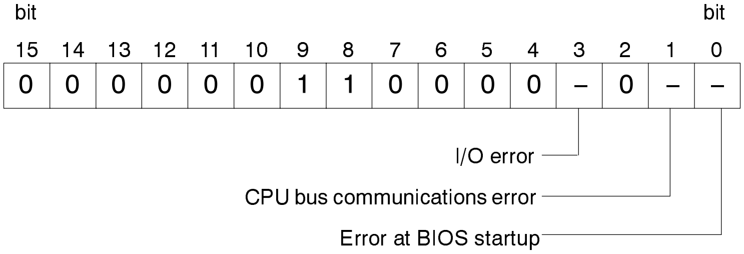
Error Log Data:

Error records are stored in the format shown below.



1B	Error type		See note 1.
1B			
1B	Error code		See note 2.
1B			
1B	Error generated	Minute	BCD format
1B	"	Second	
1B	"	Date	
1B	"	Hour	
1B	"	Year (rightmost two digits)	
1B	"	Month	

Note 1. Content and Meaning of Error Types



bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	bit 0
0	0	0	0	0	0	1	1	0	0	0	0	–	0	–	–

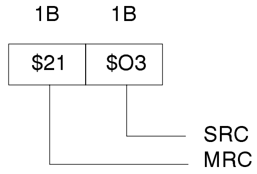
When the error log is read with the Personal Computer Unit's ERRLOG.EXE program, bits 8 and 9 will be 0 (OFF).

2. For a list of error codes, refer to the appendices.

4-6-6 Clear Error Log (2103)

Operation This operation resets the error log generated at the Personal Computer Unit.

Service Request PDU



MRC (1 Byte):

Main Request Classification. Error Logging Operations are indicated by $21.

SRC (1 Byte):

Sub-request Classification. Clear Error Log is indicated by $03.

Service Response PDU

1B	1B	1B	1B
MRC	SRC	MRES	SRES

MRC (1 Byte):

Main Request Classification. The response to Error Logging Operations is indicated by $21.

SRC (1 Byte):

Sub-request Classification. The response to Clear Error Log is indicated by $03.

MRES (1 Byte):

Main Response Code. $00 is always returned.

SRES (1 Byte):

Sub-response Code. $00 is always returned.

4-7 Sample Programs

Sample programs using the CPU Bus Driver are provided below.

```
/************************************************/
/* CPU Bus Driver                               */
/*                                              */
/* Cyclic Service Sample Program                */
/************************************************/
#include        <dos.h>
#include        <stdio.h>

typedef void far        *farptr;

union    REGS   inregs,outregs;         /*I/O register structure*/
int      fd;                            /*File handle*/
char     length;                        /*Number of bytes for IOCTL transmission*/
static   char   buf[16];                /*Data buffer for IOCTL*/

static   char   driver_id[] = ”CVIF”;            /*Driver name*/
static   char   d_addrset[] = { 10, 0x01, 0x00 }:   /*Sets transmission status*/
static   char   d_datread[] = { 12, 0x02, 0x00 }:   /*Data reading*/
static   char   d_datwrit[] = { 12, 0x03, 0x00 }:   /*Data writing*/

void     errclose();

void
```

```
main()
{
  int   i;
  register char *bufp;
  register short *sp;

  inregs.h.ah = 0x3d;         /*Opens driver.*/
  inregs.h.al = 0x02;         /*Read/write mode*/
  inregs.x.dx = (short)driver_id;
  intdos(&inregs, &outregs); /*int 21h*/
  if (outregs.x.cflag !=0){
         printf("Driver not loaded./n");
         printf("/tError code = 0x%x/n",outregs.x.ax);
         exit(1);
  }
  fd = outregs.x.ax;          /*Acquires file handle.*/

  strncpy (buf,&d_addrset[1],2);           /*Sets cyclic service status*/
  length = d_addrset[0];
  bufp = &buf[2];
  *bufp++ = 0;          /*Transmission direction, bus driver to Unit*/
  *bufp++ = 1;          /*Set number*/
  *(long*)bufp = (long)0x00404000;           /*Sets real address of bus driver*/
  bufp += 4;
  *(short*)bufp = 10;          /*Sets transmission length*/
  if (ioctl())
         errclose();

  strncpy (buf,&d_addrset[1],2);           /*Sets cyclic service status*/
  length = d_addrset[0];
  bufp = &buf[2];
  *bufp++ = 1;          /*Transmission direction, Unit to bus driver*/
  *bufp++ = 1;          /*Set number*/
  *(long*)bufp = (long)0x00404000;           /*Sets real address of bus driver*/
  bufp += 4;
  *(short*)bufp = 10;          /*Sets transmission length*/
  if(ioctl())
  errclose();

  strcpy(buf,&d_datwrit[1],[2]);           /*Writing cyclic area*/
  length = d_datwrit[0];
  bufp = &buf[2];
  *bufp++ = 1;          /*Set number*/
  *bufp++ = 0;          /*Reserve*/
  strcpy (databuf,"1234567890abcdefghij");
  *(long*)bufp = (long)(farptr)databuf;           /*Sets writing data buffer*/
  bufp += 4;
  *(short*)bufp = 10;             /*Sets number of words requested for writing*/
  if (ioctl())
         errclose();
  printf ("The number of words actually written is %d.
/n",*(short*)&buf[10]);

  strncpy(buf,&d_datread[1],[2]);           /*Reading cyclic area*/
  length = d_datread[0];
  bufp = &buf[2];
  *bufp++ = 1;          /*Set number*/
  *bufp++ = 0;          /*Reserve*/
  *(long*)bufp = (long)(farptr)databuf;         /*Sets receiving buffer address*/
  bufp += 4;
  *(short*)bufp = 10;             /*Sets number of words requested for reading*/
  if (ioctl())
         errclose();

  printf ("Data received in the DM area /n");
  sp = (short*)databuf;  /*Displays received data*/
  for (i = *(short*)&buf[10];--i> = 0;)
         printf("0x%x",*sp++);
  printf("/n");

  inregs.h.ah = 0x3e;        /*Closes driver.*/
  inregs.x.bx = fd;
  intdos(&inregs, &outregs); /*int 21h*/
}

int
ioct()
{
  inregs.x.ax = 0x4403;        /*I/O request*/
  inregs.x.bx = fd;
  inregs.x.cx = length;        /*Sets number of bytes for transmission.*/
  inregs.x.dx = (short)buf;    /*Sets parameter buffer address.*/
  intdos(&inregs,&outregs);
  if (outregs.x.cflag||outregs.x.ax)
         return(1);
  return(0);
}

void
errclose()
{
  printf("IOCTL error/n");
  printf("/tcmd=0x%x carry=0x%x AX=0x%x/n",
                     buf[0],outregs.x.cflag,outregs.x.ax);
  inregs.h.ah = 0x3e;          /*Closes driver.*/
  inregs.x.bx = fd;
  intdos(&inregs, &outregs);   /*int 21h*/
  exit(2);
}
```

```
/**************************************************/
/* CPU Bus Driver                                 */
/*                                                 */
/* Event Transmission Sample Program              */
/**************************************************/
#include        <dos.h>
#include        <stdio.h>
#include        <time.h>

typedef unsigned char uchar;
typedef unsigned short ushort;
typedef void far *farptr;

```

```
union    REGS   inregs,outregs;        /*I/O register structure*/
union    REGS   inregs2,outregs2;      /*I/O register structure*/
int      fd;                           /*File handle*/
int      rcnt;                         /*Transmission counter*/
char     length;                       /*Number of ioctl transmission digits*/
uchar    mbuf[15],*buf;                /*ioctl buffer*/
uchar    recvbuf[2048];                /*Reception buffer*/
char     cmd04hdr[] = {0xa,0x04,0x00}; /*Transmits event/FINS command.*/
char     cmd07hdr[] = {0xa,0x07,0x00}; /*Receives event/FINS response.*/
char     cmd0bhdr[] = {0x6,0x0b,0x00}; /*Branches at completion of FINS response
reception.*/
char     cmd11hdr[] = {0x8,0x11,0x00}; /*Registers time–up branch entry address.*/

/*Assembler function declarations*/
extern   int    f–sense(),c–sense();
extern   void   f–set(),f–cls();
extern   void   c–inc(),c–dec(),c–cls()
void cmd04(),cmd07(),cmd0b(),cmd11(),
void far jmp0ba(),far jmp11a();
void disp_recv(),ioctl();

void far *jmp0badr[] = {jmp0ba};      /*Response reception–complete entry address*/
void far *jmp11adr[] = {jmp11a};      /*Time–up branch entry address*/
void     timetime(int);               /*WAIT function (54.9 ms)*/
void     moritime(double*);

void     main()
{

inregs.h.ah = 0x3d;        /*Opens file.*/
inregs.h.al = 0x02;        /*read/write*/
inregs.x.dx = (short)”CVIF”;
intdos(&inregs, &outregs); /*int 21h*/
if(outregs.x.cflag !=0)
   printf(”Driver not loaded.errcode = %x/n”outregs.x.ax);
else{
   fd = outregs.x.ax;         /*Saves file handle.*/
   rcnt = 0;                  /*SID counter*/
   f_cls();                   /*Clears flag.*/
   c_cls();                   /*Clears counter.*/
   cmd0b();              /*Branches at completion of FINS response reception.*/
   cmd11();                   /*Registers time–up branch entry address.*/
   do{
         if(!c_sense()){
                timetime(2);        /*110ms*/

                cmd04();            /*Transmits event/FINS command.*/
                }else{
                      cmd07();      /*Receives event/FINS response.*/
                      c_dec();
                }
         }
         while(!f_sense()||c_sense());

         inregs.h.ah = 0x3e;        /*Closes file.*/
         inregs.x.bx = fd;
         intdos(&inregs, &outregs); /*int 21h*/
   }
}

/*FINS command transmission processing*/
void     cmd04()
{
   char *adr;
   static char  finsdata[] = {
/*       ICF,RSV,GCNT,DNA,DA1,DA2,SNA,SA1,SA2,SID*/
         0,  0,  0,   0,  0,  0x14,0, 0,  0,  0,
/*       MRC,SRC,DATA*/
         0x11,0x12,1,2};

   buf = mbuf;
   length = *cmd04hdr;                    /*Number of ioctl transmission digits*/
   adr = cmd04hdr;
   strncpy(buf,++adr,2);                  /*Command*/
   buf+=2;
   (uchar)famsdata[9] = (uchar)rcnt++;    /*SID*/
   *(long*)buf = (long)(farptr)finsdata;  /*Command buffer*/
   buf += 4;
  *((ushort*)buf) = 14;               /*Number of bytes requested for transmission*/
   ioctl();
   if(outregs.x.ax)
         printf(”Command transmission  Code error: %d/n”,outregs.x.ax);
}

/*FINS response processing*/
void     cmd07()
{
   char   *adr;

   buf = mbuf;
   length = *cmd07hdr;                      /*Number of ioctl transmission digits*/
   adr = cmd07hdr;
   strncpy(buf,++adr,2);                    /*Command*/
   buf++;
   *((uchar*)buf)++ = 1;                 /*Timer monitoring: 0: Yes; Other: No*/
   *((long*)buf)++ = (long)(farptr)recvbuf; /*Command buffer*/
   *((ushort*)buf) = 1000;                /*Number of bytes requested for reception*/
   ioctl();
   if(!outregs.x.ax)
         disp_recv(*(short*)(mbuf+8));/*Displays data received.*/
   else
         printf(”Response reception  Code error: %d/n”,outregs.x.ax);
}

/*Branch processing at completion of FINS response reception*/
void     cmd0b()
{
   char   *adr;

   buf = mbuf;
   length = *cmd0bhdr;                       /*Number of ioctl transmission digits*/
   adr = cmd0bhdr;
   strncpy(buf,++adr,2);                     /*Command*/
   buf+=2;
   *(long*)buf = (long)(farptr)imp0badr[0];  /*Reception–complete entry address*/
   ioctl();
   if(outregs.x.ax)
         printf(*Branch at completion of response reception  Code
error:*:%d/n”,outregs.x.ax);
}
#pragma check_stack(off)
void     far jmp0ba()
{
   c_inc();
}
#pragma check_stack(on)

/*User timer service setting processing*/
void     cmd11()
{
   char   *adr;

   buf = mbuf;
   length = *cmd11hdr;                       /*Number of ioctl transmission digits*/
   adr = cmd11hdr;
   strncpy(buf,++adr,2);                     /*Command*/
   buf+=2;
   *(long*)buf = (long)(farptr)imp11adr[0];  /*Timer interrupt entry address*/
   buf += 4;
   *((ushort*)buf) = 200;                    /*Timer value*/
   ioctl();
   if(outregs.x.ax)
         printf(User timer set Code error::%d/n”,outregs.x.ax);
}
#pragma check_stack(off)
void     far jmp11a()
{
   f_set();
}
#pragma check_stack(on)

void     ioctl()
{
   inregs.x.ax = 0x4403;/*SEND*/
   inregs.x.bx = fd;
   inregs.x.cx = length;
   inregs.x.dx = (short)mbuf;
   intdos(&inregs, &outregs);
   if(outregs.x.cflag!=0){
         printf(”MS–DOS system error*/n”);
         inregs.h.ah = 0x3e;   /*Closes file.*/
         inregs.x.bx = fd;
         intdos(&inregs, &outregs); /*int 21h*/
         exit(2);
  }
}

void     disp_recv(j)
```

```
intj;
{
   register int  i,k;

   printf(”Number of bytes of real data received = %d/n/n”,j);
   printf(”/t/t0 1 2 3 4 5 6 7 8 9 a b c d e f /n”);
   printf(”Reception data = /t”);
   for(i=0,k=0;i<j;i++){
         printf(”%02x”,recvbuf[i]);
         k++;
         if(k==16){
                 printf(”/n/t/t”);
                 k = 0;
         }
   }
   printf(”/n”);
}
void timetime(c)
{
   double nowtime;
   double aftertime;
   moritime(&nowtime);
   moritime(&aftertime);
   for(;(aftertime–nowtime)<=c;)
         moritime(&aftertime);
}
void moritime(t)
double   *t;
{
   inregs2.h.ah=0;
   int86(0x1a,&inregs2,&outregs2);
   *t=(double)outregs2.x.dx;
}
```

```
/*************************************************/
/* CPU Bus Driver                                */
/*                                               */
/* Event Reception Sample Program                */
/*************************************************/
#include        <dos.h>
#include        <stdio.h>

typedef unsigned char uchar;
typedef unsigned short ushort;
typedef void far *farptr;

union    REGS   inregs,outregs;         /*I/O register structure*/
int      fd;                            /*File handle*/
char     length;                        /*Number of ioctl transmission digits*/
uchar    mbuf[15],*buf;                 /*ioctl buffer*/
uchar    recvbuf[2048];                 /*Reception buffer*/
char     cmd05hdr[] = {0xa,0x05,0x00};  /*Transmits event/FINS response.*/
char     cmd06hdr[] = {0xa,0x06,0x00};  /*Receives event/FINS command.*/
char     cmd0ahdr[] = {0x6,0x0a,0x00};  /*Branches at completion of event/FINS reception.*/
char     cmd0chdr[] = {0x4,0x0c,0x00};  /*Sets event timeout value.*/
char     cmd11hdr[] = {0x8,0x11,0x00};  /*Registers time–up branch entry address.*/
```

```

/*Assembler function declarations*/
extern   int    f-sense(),c-sense();
extern   void   f-set(),f-cls();
extern   void   c-inc(),c-dec(),c-cls()
void cmd05(),cmd06(),cmd0c(),cmd11(),
void far jmp0aa(),far jmp11a();
void disp_recv(),ioctl();

void far *jmp0aadr[] = {jmp0aa};      /*Entry address at completion of reception*/
void far *jmp11adr[] = {jmp11a};      /*Time-up branch entry address*/

void     main()
{
   inregs.h.ah = 0x3d;        /*Opens file.*/
   inregs.h.al = 0x02;        /*read/write*/
   inregs.x.dx = (short)”CVIF”;
   intdos(&inregs, &outregs); /*int 21h*/
   if(outregs.x.cflag !=0){
         printf(”Driver not loaded.errcode = %x/n”outregs.x.ax);
         exit(1);
   }
   else{
         fd = outregs.x.ax;     /*Saves file handle.*/
         c_cls();
         f_cls();
         cmd0a();               /*Branches at completion of event/FINS reception.*/
   cmd0c();                     /*Sets event timeout value.*/
         cmd11();               /*Registers time-up branch entry address.*/
         do{
                if(c_sense()){
                       c_dec();
                       cmd06();         /*Receives event/FINS command.*/
                       cmd05();         /*Transmits event/FINS response.*/
                }
         }
         while(!f_sense()||c_sense());

         inregs.h.ah = 0x3e;        /*Closes file.*/
         inregs.x.bx = fd;
         intdos(&inregs, &outregs); /*int 21h*/
   }
}

/*Event service and FINS response transmission processing*/
void     cmd05()
{
   char   *adr;
   static char  finsdata[] = {
/*       ICF,RSV,GCNT,DNA,DA1,DA2,SNA,SA1,SA2,SID*/
         0,  0,  0,   0,  0,  0,  0,  0,  0,  0,
/*       MRC,SRC,MRES,SRES,RES-DATA*/
         0x11,0x12,0,0,3,4};

   buf = mbuf;
   length = *cmd05hdr;            /*Number of ioctl transmission digits*/
   adr = cmd05hdr;
   strncpy(buf,++adr,2);          /*Command*/
   buf+=2;
   finsdata[0] = recvbuf[0]       /*ICF*/
   finsdata[1] = recvbuf[6]       /*RSV*/
   finsdata[3] = recvbuf[6]       /*DNA*/
   finsdata[4] = recvbuf[7]       /*DA1*/
   finsdata[5] = recvbuf[8]       /*DA2*/
   finsdata[9] = recvbuf[9]       /*SID*/
   finsdata[10] = recvbuf[10]     /*MRC*/
   finsdata[11] = recvbuf[11]     /*SRC*/
   *(long*)buf = (long)(farptr)finsdata;  /*Buffer address*/
   buf += 4;
  *((ushort*)buf) = 16;              /*Number of bytes requested for transmission*/
   ioctl();
   if(outregs.x.ax)
         printf("Response transmission Code error:%d/n”,outregs.x.ax);
}

/*FINS command reception request*/
void     cmd07()
{
   char   *adr;

   buf = mbuf;
   length = *cmd06hdr;                   /*Number of ioctl transmission digits*/
   adr = cmd06hdr;
   strncpy(buf,++adr,2);                 /*Command*/
   buf+=2;
   *(long*)buf = (long)(farptr)recvbuf;  /*Reception buffer address*/
   buf += 4;
  *((ushort*)buf) = 1000;                /*Number of bytes requested for reception*/
   ioctl();
   if(!outregs.x.ax)
         disp_recv(*(short*)(mbuf+8));   /*Displays data received.*/
   else
         printf(Command reception  Code error:%d/n”,outregs.x.ax);
}

/*Branch processing at completion of FINS command reception*/
void     cmd0a()
{
   char    *adr;

   buf = mbuf;
   length = *cmd0ahdr;                       /*Number of ioctl transmission digits*/
   adr = cmd0ahdr;
   strncpy(buf,++adr,2);                     /*Command*/
   buf+=2;
   *(long*)buf = (long)(farptr)imp0aadr[0];  /*Reception–complete entry address*/
   ioctl();
   if(outregs.x.ax)
         printf(*Branch at completion of command reception  Code
error:%d/n”,outregs.x.ax);
}
#pragma check_stack(off)
```

```
void far jmp0aa()
{
c_inc();
}
#pragma check_stack(on)

/*Event timer setting processing*/
void     cmd0c()
{
   char   *adr;

   buf = mbuf;
   length = *cmd0chdr;                 /*ioctlNumber of ioctl transmission
digits*/
   adr = cmd0chdr;
   strncpy(buf,++adr,2);               /*Command*/
   buf+=2;
   *((ushort*)buf) = 100;             /*Timeout value*/
   ioctl();
   if(outregs.x.ax)
         printf(Timer value setting Code error:%d/n”,outregs.x.ax);
}

/*User timer service setting processing*/
void     cmd11()
{
   char   *adr;

   buf = mbuf;
   length = *cmd11hdr;                 /*Number of ioctl transmission digits*/
   adr = cmd11hdr;
   strncpy(buf,++adr,2);               /*Command*/
   buf+=2;
   *(long*)buf = (long)(farptr)imp11adr[0];  /*Timer interrupt entry address*/
   buf += 4;
   *((ushort*)buf) = 100;             /*Timer value*/
   ioctl();
   if(outregs.x.ax)
         printf(User timer setting Code error:%d/n”,outregs.x.ax);
}
#pragma check_stack(off)
void     far jmp11a()
{
   f_set();
}
#pragma check_stack(on)

void     ioctl()
{
   register int  i;

   inregs.x.ax = 0x4403;            /*SEND*/
   inregs.x.bx = fd;
   inregs.x.cx = length;
   inregs.x.dx = (short)mbuf;
   intdos(&inregs, &outregs);
   if(outregs.x.cflag!=0){
         printf(”MS–DOS system error*/n”);
         inregs.h.ah = 0x3e;        /*Closes file*/
         inregs.x.bx = fd;
         intdos(&inregs, &outregs); /*int 21h*/
         exit(2);
  }
}

void     disp_recv(j)
intj;
{
   register int  i,k;

   printf(”Number of bytes of real data received = %d/n/n”,j);
   printf(”/t/t0 1 2 3 4 5 6 7 8 9 a b c d e f/n”);
   printf(”Reception data = /t”);
   for(i=0,k=0;i<j;i++){
         printf(”%02x”,recvbuf[i]);
         k++;
         if(k=16){
                 printf(”/n/t/t”);
                 k = 0;
         }
  }
  printf(”/n”);
}
```

```
/*************************************************/
/* CPU Bus Driver                                */
/*                                               */
/* CPU Bus Link Service Sample Program           */
/*************************************************/
#include        <dos.h>
#include        <stdio.h>

typedef void far *farptr;

union    REGS   inregs,outregs;        /*I/O register structure*/
int      fd;                           /*File handle*/
char     length;                       /*Number of IOCTL transmission digits*/
static   char   buf[16],databuf[512];  /*Data buffer for IOCTL*/

static   char   driver_id[] = ”CVIF”;           /*Driver name*/
static   char   d_rsvread[] = {6,0x0e,0x00};    /*Reads reserved information.*/
static   char   d_lnkread[] = {12,0x0f,0x00};   /*Reads link area.*/
static   char   d_lnkwrit[] = {12,0x10,0x00};   /*Writes link area.*/
static   char   d_gokiadr[] = {4,0x13,0x00};    /*Inquires regarding unit address.*/

void     errclose();

void
main()
{
   int    i;
   char   goki;
   register short *bufp;
```

```

        inregs.h.ah = 0x3d;        /*Opens driver.*/
        inregs.h.al = 0x02;        /*Read/write mode*/
        inregs.x.dx = (short)driver_id;
        intdos(&inregs, &outregs); /*INT 21H*/
        if(outregs.x.cflag){
              printf(”Driver not loaded./n”);
              printf(”/tError code = 0x%x/n”outregs.x.ax);
              exit(1);
        }
        fd = outregs.x.ax;         /*Acquires file handle.*/

        strncpy(buf,&d_rsvread[1],2);           /*Reads information reserved for system.*/
        length = d_rsvread[0];
        bufp = (short*)&buf[2];
        *(long*)bufp = (long)(farptr)databuf;   /*Sets reception buffer address.*/
        if(ioct())
              errclose();

        printf(System-reserved information data/n”);
        bufp = (short*)databuf;                 /*Displays data received.*/
        for(i=8;-i>=0;)
              printf(”0x%x”,*bufp++);
        printf(”/n”);

        strncpy(buf,&d_lnkwrit[1],2);           /*Writes CPU bus link area.*/
        length = d_lnkwrit[0];
        bufp = (short*)&buf[2];
        *bufp++=0;                              /*Sets beginning word for writing.*/
        strcpy(databuf,”0123456789abcdef”);
        *(long*)bufp = (long)(farptr)databuf;   /*Sets buffer for written data.*/
        bufp+=2;
        *bufp=8;                             /*Sets number of words requested for writing.*/
        if(ioct())
              errclose();
        printf(The number of words actually written is %d./n”,
*(short*)&buf[10]);

        strncpy(buf,&d_gokiadr[1],2);           /*Inquires regarding unit address.*/
        length = d_gokiadr[0];
        if(ioct())
              errclose();
        goki=buf[2];                            /*for Special I/O Unit*/
        printf(The unit address is %d./n”,goki);

        strncpy(buf,&d_lnkread[1],2);           /*Reads CPU bus link area.*/
        length = d_lnkread[0];
        bufp = (short*)&buf[2]
        *bufp++=goki*8+128;                     /*Sets beginning word for reading.*/
        *(long*)bufp = (long)(farptr)databuf;  /*Sets reception buffer address.*/
        bufp+=2;
        *bufp=8;                            /*Sets number of words requested for reading.*/
        if(ioct())
              errclose();

        printf(Unit number data read/n”);
```

```
   bufp = (short*)databuf;                   /*Displays data received.*/
   for(i=*(short*)&buf[10];-i>=0;)
         printf("0x%x",*bufp++);
   printf("/n");

   inregs.h.ah = 0x3e;          /*Closes driver.*/
   inregs.x.bx = fd;
   intdos(&inregs, &outregs); /*INT 21H*/
}

int
ioctl()
{
   inregs.x.ax = 0x4403;           /*I/O request*/
   inregs.x.bx = fd;
   inregs.x.cx = length;           /*Sets number of bytes to receive.*/
   inregs.x.dx = (short)buf;       /*Sets parameter buffer address.*/
   intdos(&inregs, &outregs);
   if(outregs.x.cflag||outregs.x.ax)
         return(1);
   return(0);
}

void
errclose()
{
   printf("IOCTL error*/n");
   printf("/tcmd=0x%x carry=0x%x AX=0x%x/n");
                     buf[0],outregs.x.cflag,outregs.x.ax);
   inregs.h.ah = 0x3e;          /*Closes driver.*/
   inregs.x.bx = fd;
   intdos(&inregs, &outregs); /*INT 21H*/
   exit(2);
}
```

```
;************************************************************
;* S Bus Interface Communications Driver                    *
;*                                                          *
;* Sample program for branch at completion and user timer   *
;************************************************************
_TEXT segment byte public 'code'
      assume cs:_TEXT
      assume ds:_TEXT
      public _f_set,_f_sense,_f_cls,_endflg
      public _c_inc,_c_dec,_c_sense,_c_cls,_cnt
;************************************
;*     Flag set processing          *
;************************************
_f_set proc near
      mov          cs:_endflg,1
      ret
_f_set endp
;************************************
;*     Flag sense processing        *
;************************************
_f_sense proc near
      mov          ax,cs:_endflg
      ret
_f_sense endp
;*************************************
;*     Flag clear processing         *
;*************************************
_f_cls proc near
      xor          ax,ax
      mov          cs:_endflg, ax
      ret
_f_cls endp
;*************************************
;*     Counter increment processing  *
;*************************************
_c_inc proc near
      inc          cs:_cnt
      ret
_c_inc endp
;*************************************
;*     Counter decrement processing  *
;*************************************
_c_dec proc near
      dec          cs:_cnt
      ret
_c_dec endp

;*************************************
;*     Counter sense processing      *
;*************************************
_c_sense proc near
      mov          ax,cs:_cnt
      ret
_c_sense endp
;*************************************
;*     Counter clear processing      *
;*************************************
_c_cls proc near
      xor          ax,ax
      mov          cs:_cnt, ax
      ret
_c_cls endp
;*************************************
;*     Flag and counter              *
;*************************************
_endflg dw 0
_cnt dw 0

_TEXT ends
      end
```

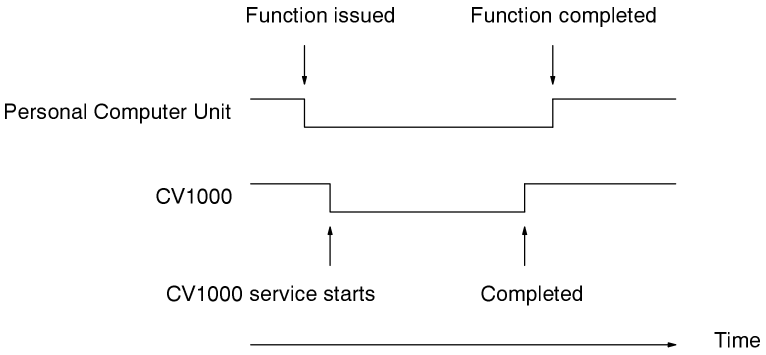
4-8 Measuring CPU Bus Access Performance

This section explains how to measure the performance of the CPU bus. The performance will vary depending on the program and the configuration of Units, such as I/O Units and CPU Bus Units. The processing time of a CV1000 is used in these examples.

4-8-1 Cyclic Service

This example shows how to measure the time required to read or write data in the PC's DM Area, from the issuance of the command (02 or 03) to the completion of the operation. Refer to *4-5-2 Reading the Cyclic Area* for details on command 02 or *4-5-3 Writing to the Cyclic Area* for details on command 03.

The time required to read/write the data is:
Number of words × 2 µs + 420 µs + The CV1000's processing time



Refer to *Section 6* of the *CV-series PC Operation Manual: Ladder Diagrams* for details on the CV1000's processing time. In these examples, the Personal Computer Unit is the only CPU Bus Unit connected to the CV1000.

The minimum processing time for the Personal Computer Unit is 1.8 ms, and occurs when the request for peripheral processing is received immediately.

The maximum processing time for the Personal Computer Unit is 7.0 ms, and occurs when the request for peripheral processing encounters the maximum delay. There will be an additional delay for program execution when the CV1000 is set for synchronous operation.

Example

Use the following equation to calculate the time required to read 1000 words of DM data when the CV1000 is set for synchronous operation and the Personal Computer Unit is the only CPU Bus Unit connected.

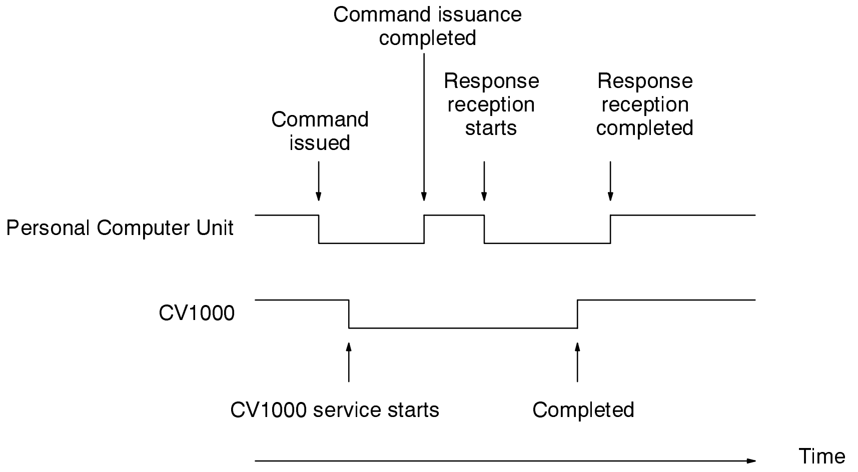
1000 × 2 µs + 420 µs + (1.8 to 7.0 ms) = 4.22 to 9.42 ms

4-8-2 Event Service

This example shows how to measure the time required to read or write data in the PC's DM Area, from the issuance of command 04 to the reception of the response with command 07. Refer to *4-5-4 Transmitting FINS Commands* for details on command 04 or *4-5-7 Receiving FINS Responses* for details on command 07.

The time required to read/write the data is:
Number of words × 2 × 1.6 µs + 1.5 ms + The CV1000's processing time

In this case, the response reception was executed in the Personal Computer Unit before the CV1000's processing was completed.


Command issuance completed
Response reception starts
Response reception completed
Command issued
Personal Computer Unit
CV1000
CV1000 service starts
Completed
Time

Refer to *Section 6* of the *CV-series PC Operation Manual: Ladder Diagrams* for details on the CV1000's processing time. In these examples, the Personal Computer Unit is the only CPU Bus Unit connected to the CV1000.

The minimum processing time for the Personal Computer Unit is 1.8 ms, which occurs when the request for peripheral processing is received immediately and the processing is completed within one event service.

The maximum processing time for the Personal Computer Unit is 15.8 ms (1.8+5.2+1.8+5.2+1.8 ms), which occurs when the request for peripheral processing encounters the maximum delay and processing isn't completed in two event services. Depending on the command, there will be an additional delay of the cycle time required for program execution.

Example

Use the following equation to calculate the time required to read 500 words of DM data when the CV1000 is set for synchronous operation and the Personal Computer Unit is the only CPU Bus Unit connected.

500 × 2 × 1.6 μs + 1.5 ms + (1.8 to 15.8 ms) = 4.9 to 18.9 ms

4-8-3 CPU Bus Link Service

The CPU Bus Link service must be enabled in the CV1000's PC Setup (letter I). The service is always executed once every 10 ms.

SECTION 5
FINS Library

This section describes the FINS Library.

5-1 Introduction 122
5-2 BASIC Program and FINS Library Structure 122
5-3 Processing Flow 123
5-3-1 Processing Procedure 1 123
5-3-2 Processing Procedure 2 124
5-4 Using the FINS Library 124
5-4-1 Designing with Quick BASIC 124
5-4-2 Designing with BASIC Other than Quick BASIC 125
5-5 FINS Library Operations 126
5-5-1 SOPEN: Initialize 126
5-5-2 SCLOSE: Close 126
5-5-3 SSEND: Transmit 126
5-5-4 SRECV: Receive (Without Timer Monitoring) 128
5-5-5 SRCVT: Receive (With Timer Monitoring) 130
5-6 Sample Programs 132

5-1 Introduction

When BASIC is used for programming, drivers such as the CPU Bus Driver cannot be used because settings to the register are required. The FINS Library allows the BASIC user, as well, to utilize event service through the CPU bus interface.

Note

1. Event service is the only service that can be utilized when the FINS Library is used. For details regarding event service, refer to *1-4-2 Event Service*.
2. The FINS Library operates using the CPU Bus Driver. Therefore the CPU Bus Driver must be installed in order to use the library.
3. The FINS Library doesn't have to be installed. This library is compatible with Microsoft's Quick BASIC 4.5. Refer to *2-4 Changing the Quick Library Version* when using Quick BASIC 4.2.

Installing the CPU Bus Driver

The CPU Bus Driver (SBUS.SYS) is recorded in the CONFIG.SYS file in drive F in advance. Be sure that SBUS.SYS is recorded in the active CONFIG.SYS file when a system disk is being modified or a new CONFIG.SYS file is being created.

Add the following line to the CONFIG.SYS file to record the CPU Bus Driver. Refer to *2-6 Installing Device Drivers* for details on other SBUS.SYS parameters. (The F:\CONFIG .SYS file is used when the Unit is started from Built-in ROM.)

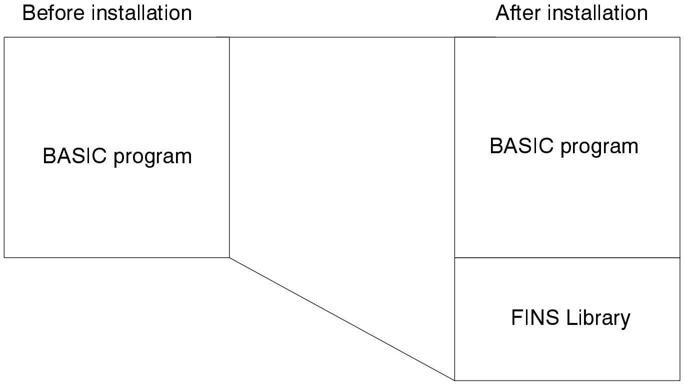
```
DEVICE=E:\SBUS.SYS /V65
```

Drive name

The CPU Bus Driver is recorded in the Personal Computer Unit's Built-in ROM (drive E) in advance.

5-2 BASIC Program and FINS Library Structure

When the FINS Library is installed in a BASIC program, the structure will be as shown below.



As is illustrated above, when the FINS Library is installed the programs registered in the library can be linked with the user's BASIC program to enlarge the load module. In addition, when a program registered in the library is called by the BASIC program, that program will run and services will be available to the BASIC program through the CPU bus interface.

Note For instructions on using the FINS Library, refer to *5-4 Using the FINS Library*.

5-3 Processing Flow

This section will describe the communications procedure (flow) when communications are executed using the FINS Library.

When the FINS Library is used, the only service that can be employed is event service.

The communications procedure when event service is used is as follows:

1, 2, 3...

1. Service request command is transmitted to a device offering a service.
2. Response to the service requested is received from the device offering the service.

Since this is the communications procedure for event service, the procedure will be the same when the FINS Library is used. In the case of the library, however, the service is processed using the CPU Bus Driver. Thus the communications data will be transmitted and received via the CPU Bus Driver.

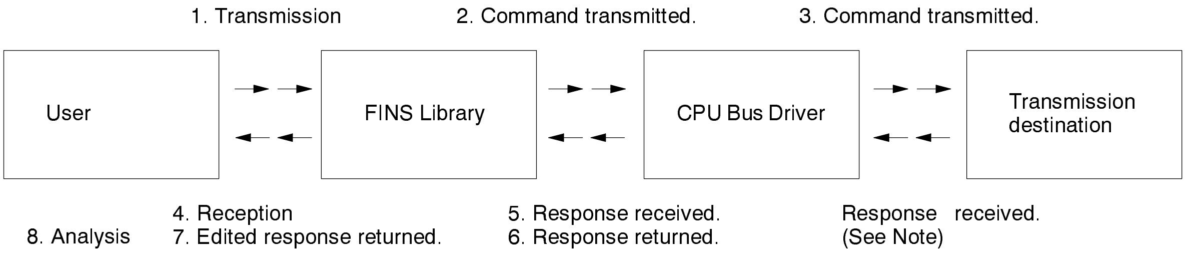
The communications data handled by the FINS Library conforms to the portion of the Programmable Controller mailbox command from MRC onwards. A maximum of 2,002 bytes of data can be transmitted.

The contents of FINS commands will vary depending on the contents of the service requested. In addition, the contents of the response will vary according to the command.

Refer to the *FINS Command Reference Manual (W227)* for more details.

5-3-1 Processing Procedure 1

When a Request Command is Transmitted from the Personal Computer Unit to a Device that Offers a Service, and a Response is Received from that Device



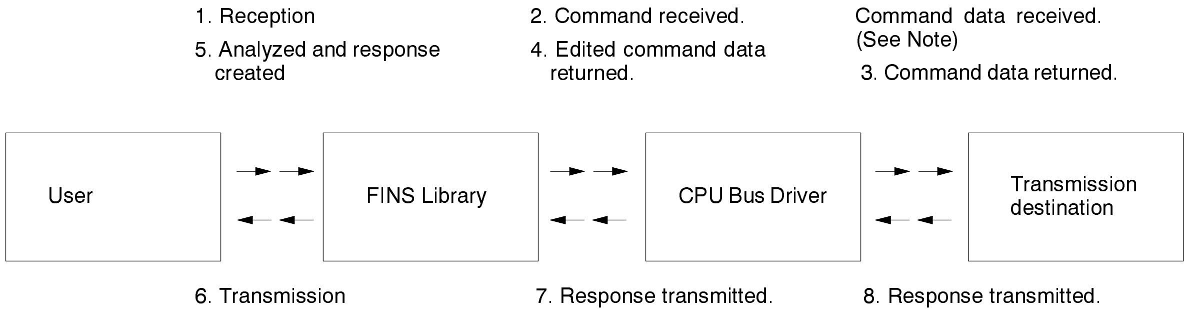
1, 2, 3...

1. The user creates a request command corresponding to the service that is to be requested, and then specifies the device offering the service and calls“transmit” from the FINS Library.
2. The BASIC interface library receives the request from the user and requests “transmit command” of the CPU Bus Interface Communications Driver.
3. The CPU Bus Driver receives the “transmit command” request from the FINS Library, and transmits the specified request command to the specified device.
4. In order to receive a response to the transmitted command, the BASIC user calls “receive response” from the FINS Library.
5. The FINS Library receives the request from the user, and requests “receive response” of the CPU Bus Driver.
6. In answer to the “receive response” requested by the FINS Library, the CPU Bus Driver returns to the FINS Library the response data it has received from the command destination.
7. The FINS Library receives the response data from the CPU Bus Driver and edits it into a form usable by BASIC, and then returns it to the user.
8. The BASIC user analyzes the contents of the response that was received.

Note The response reception from the transmission destination will be executed regardless of user request.

5-3-2 Processing Procedure 2

When a Request Command is Received from Another Device, and a Response is Transmitted to the Source of the Command Transmission



1, 2, 3...

1. In order to receive a request command from another device, the user calls “receive” from the FINS Library.
2. The FINS Library receives the request from the user, and requests “receive command” of the CPU Bus Driver.
3. In answer to the “receive command” requested by the FINS Library, the CPU Bus Driver returns to the FINS Library the command data it has received from the other device.
4. The FINS Library receives the command data from the CPU Bus Driver and edits it into a form usable by BASIC, and then returns it to the user.
5. The BASIC user analyzes the command data that is received, and creates corresponding response data.
6. Along with creating the response data, the BASIC user specifies the source of the request command and calls “transmit” from the FINS Library.
7. The FINS Library receives the request from the user, and requests “transmit response” of the CPU Bus Interface Communications Driver.
8. The CPU Bus Driver receives the “transmit response” request from the FINS Library, and transmits the specified response data to the specified device.

Note The command reception from the other device will be executed regardless of user request.

5-4 Using the FINS Library

Usage of the FINS Library will vary depending on the version of BASIC being used. Explanations are provided below both for Quick BASIC and other forms of BASIC.

5-4-1 Designing with Quick BASIC

Copying Files

Copy the applicable file from the System Floppy Disk to the System Disk.

`PCDGSUB.LIB` (When designing with compiler)
`PCDGSUB.QLB` (When designing with interpreter)

The way to start up with the interpreter is as follows:

```
QB/L PCDGSUB
```

For detailed instructions, refer to the Quick BASIC manual.

Linking to the Library

When compiling the BASIC program, link to PCDGSUB.LIB and create an executive module.

Example: Creating `SMP.EXE` from `SAMP.BAS`

```
BC SMP.BAS,,NUL,;
LINK/NOE SMP.OBJ+PCDGSUB.
   LIB,,NUL.;
```

Calling Method

Calling is performed by means of the following procedure.

```
DECLARE SUB func CDECL ALIAS ”realname”
CALL func(argument)
```

Note `func` is the name of the function called; the “real name” is the name used in the library.

The correspondences of function names and the names used in the library are shown in the table below. Use these names when calling.

Operation number	Function name	Name used in library
00H	SOPEN	_binit
01H	SCLOSE	_bclose
02H	SSEND	_xsend_dat
03H	SRECV	_xrecv_rs()
04H	SRCVT	_xrecv_rs1

Example

```
100   DECLARE SUB SOPEN CDECL ALIAS ”_binit”
110   DECLARE SUB SRCVT CDECL ALIAS ”_xrecv_rs1”
120   CALLS SOPEN(RST%)
.
.
180   D$=STRING$(255,””)
190   CALLS SRCVT(NA%,DA%,UN%,TYPE%,SID%,D$,CT%,RST%,RC%,TIM%)
.
.
```

5-4-2 Designing with BASIC Other than Quick BASIC

The form of BASIC available to the user through the FINS Library on the System Floppy Disk is Quick BASIC. The FINS Library must be modified, therefore, in order to utilize other forms of BASIC. The way to modify and use the FINS Library is explained below.

Copying Files

Copy the following files from the System Floppy Disk to the System Disk:

```
PCDGMAIN.OBJ
PCDGSUB.C
BASICIF.H
```

Modifying and Using the FINS Library

The Designed BASIC will be linked with the FINS Library. First, rewrite the format (`str` structure) of the string descriptor in `BASICIF.H` so that it agrees with the designed BASIC.

```
struct str  {  /*Quick BASIC*/
 int len;                /*String length*/
 char *adr;              /*Character-type data storage area*/
};                       /*Leading offset address*/
```

Compile based on the medium model, and create the library.

Example: Compiling with MS-C

```
cl/c/AM/Zp/Op/Gs pcdgsub.c
lib pcdgsub.lib–+pcdgsub.obj;
```

Use the library created above, when compiling the BASIC program according to the instructions given on page 124, *Designing with Quick BASIC*, to link with the created library and create an execution form.

5-5 FINS Library Operations

This section will explain FINS Library operations, and give the contents of arguments and return data when each of these operations is used.

Operation		Summary
00H	Initialize	Initializes the service. From that point on, transmission or reception requests cannot be accepted.
01H	Close	Carries out exit processing for the service. From that point on, transmission or reception requests cannot be accepted.
02H	Transmit	Transmits a service request command, or a response to a service request, to another device.
03H	Receive (without timer monitoring)	Receives commands from other devices, and receives responses to transmissions sent by the Unit itself. (Without timer monitoring: Reception is aborted by pressing the Escape Key.)
04H	Receive (with timer monitoring)	Receives commands from other devices, and receives responses to transmissions sent by the Unit itself. (With timer monitoring)

5-5-1 SOPEN: Initialize

Operation Opens the CPU Bus Driver, and initializes the service.

Operation Number 00

Format `CALLS SOPEN(RST%)(QuickBASIC)(_binit)`

Argument (Output) RST%: Return status

0:	Normal termination
–3:	Already open.
–16:	CPU Bus Driver not loaded.

Explanation When using this library operation, be sure to perform it at the beginning of the program. Also be sure to initialize when reusing library operations after the CPU Bus Driver has been closed. The reception buffers will be cleared at this time.

5-5-2 SCLOSE: Close

Operation Closes the CPU Bus Driver and ends the service.

Operation Number 01

Format `CALLS SCLOSE(QuickBASIC)(_bclose)`

Arguments None

Explanation Once “close” has been executed, the FINS Library operations cannot be used again until initialization.

5-5-3 SSEND: Transmit

Operation Transmits a service request command, or a response to a service request, to another device.

Operation Number 02

Format `CALLS SSEND(NA%,DA%,UN%,TYPE%,SID%,D$,RST%)(QuickBASIC)` `(_xsend-dat)`

Arguments

(Input) `NA%`	Transmission destination network address
(Input) `DA%`	Transmission destination node address

(Input) UN% Transmission destination unit address
(Input) TYPE% Command (0)/Response (1)
(Input) SID% Service ID (0 to 255)
(Input) D% Transmission data
(Input) RST% Command

00	Transmission end
01	Transmission continue
02	Transmission buffer problem

(Output) RST% Return status

0 Normal termination

−1 Argument error
Destination device address error
Other destination device exists
(With "transmission continue" specified, NA%, DA%, UN% are different from before.}
Command error (RST% is outside of 00 to 02 range.)

−2 Not open

−9 Number of transmission bytes error
(Outside of 2 to 2,002-byte range. This includes exceeding 2,002 bytes with "transmission continue" specified.)

−22 Abnormal termination

Explanation

Transmission data is sent to the specified destination device as data corresponding to the Command/Response (TYPE%) designation.

If the service ID (SID) is specified at the time the command is sent, that data will be returned from the destination device as the SID of the response to the command. Thus the correspondence between the command and the response can be determined by means of specifying the SID.

(When transmitting a response, specify the SID so that it is the same as the SID of the command.)

The transmission data conforms to the portion of the FINS command from MRC onwards.

The meanings of the communicating device address designations are explained below. Refer to the *FINS Command Reference Manual (W227)* for more details on the contents of FINS commands.

Network Address
This is the network address of the transmission destination, and indicates the final target location address. The following code has a special meaning.

$00: Same network address

Node address
This is the node address of the transmission destination, and indicates the final target location address. The following codes have special meanings.

$00: Same node address

$FF: Broadcast to all nodes on specified network

Unit address
This is the unit address of the transmission destination, and the final target location address is specified by the absolute address.

Example: Special I/O Unit #0 will have an address of $10.

The following codes have special meanings.

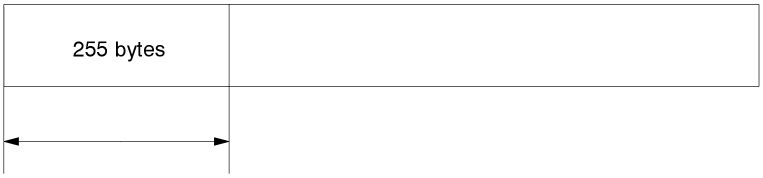
$00: Unit address of Programmable Controller
$10 to $2F: CPU Bus Units
$FD: Peripheral Tools (e.g., FIT)
$FE: Communications Units (e.g., SYSMAC NET, SYSMAC LINK)

When the data to be transmitted equals or exceeds the character-type variable size (255 bytes), the transmission procedure will be as follows. In this example, 700 bytes of data are transmitted.

1, 2, 3...

1. The first 255 bytes of data are transmitted with command `(RST%) = 01`.

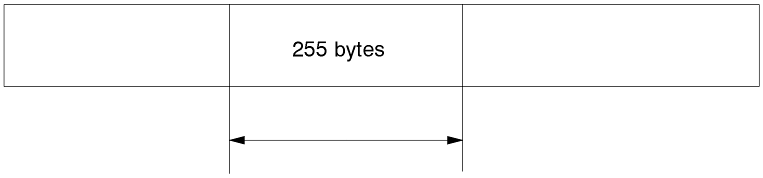
Transmission data



This portion of data is transmitted.

2. The next portion of data is transmitted with command `(RST%) = 01`.

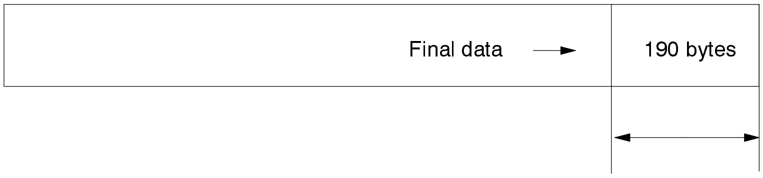
Transmission data



This portion of data is transmitted.

3. Step 2. is repeated until no more data remains to be transmitted.
4. When the last portion of data is reached, it is transmitted by command `(RST%) = 00`.

Transmission data



The final data is transmitted.

If the data to be transmitted is less than the character-type variable size, it can be transmitted by step 4. alone.

If the data transmission has already been started, and you want to abort the transmission and discard the data, then send any arbitrary data by command `(RST%) = 02`. Both the data that was already being transmitted by “transmission continue” and the arbitrary data that was sent at the end will be discarded.

To transmit a request command to another device and receive a response to that request command, use the receive operation.

5-5-4 SRECV: Receive (Without Timer Monitoring)

Operation

Receives commands from other devices, and receives responses to transmissions sent by the Unit itself. The waiting-to-receive time is not monitored, and the waiting-to-receive status is forcibly ended by pressing the Escape Key.

Operation Number

03

Format

```
CALLS SRECV (NA%,DA%,UN%,TYPE%,SID%,D$,RST%,RC%)
(QuickBASIC)(_xrcv_rso)
```

Arguments

(Output) NA%	Transmission source network address
(Output) DA%	Transmission source node address

(Output) `UN%`	Transmission source unit address
(Output) `TYPE%`	Type of data received: Command (0)/Response (1)
(Output) `SID%`	Service ID of data received (0 to 255)
(Input) `D$`	Reception buffer (Secure the area in advance.)
(Output) `D%`	Reception data
(Output) `CT%`	Number of reception data bytes stored in D$
(Output) `RST%`	Return status

0	Normal termination
−2	Not open.
−11	Forcibly ended by Escape Key.
−22	Abnormal termination

(Output) `RC%`	Reception buffer size insufficient: Number of bytes of data remaining that cannot be stored

Explanation

A command from another device, or a response to a request command transmissions sent by the Unit itself, is received. Along with the data type that is received, the data is stored in the reception buffer.

If some other data is already being received at the time of this reception request, the data is returned will be returned to the user according to the following rules.

When Only Command Data is Received
The command data that is received will be returned to the user in the order in which it is received.

When Only Response Data is Received
The response data that is received will be returned to the user in the order in which it is received.

When Command and Response Data are Received Together
First the response data that is received will be returned to the user in the order in which it is received. When there is no more response data to return, then the command data that is received will be returned to the user in the order in which it is received.

The service ID (`SID`) here is the same as that specified at the time of command transmission, and will be returned from the destination device as the `SID` of the response to that command. Thus the correspondence between the command and the response can be recognized by means of specifying the `SID`.

The transmission data conforms to the portion of the FINS command from MRC onwards. Refer to the *FINS Command Reference Manual (W227)* for more details on the contents of FINS commands.

In addition, the location of the source of the transmitted data will be stored in the transmission source address area, and the number of bytes actually stored in the buffer will be stored in the area for the amount of data received.

The meanings of the transmission source address designations are explained below.

Network Address
This is the network address of the transmission source. The following code has a special meaning.

$00: Same network address

Node address
This is the node address of the transmission source. The following code has a special meaning.

$00: Same node address

Unit address
This is the unit address of the transmission destination, and is stored by the absolute address.

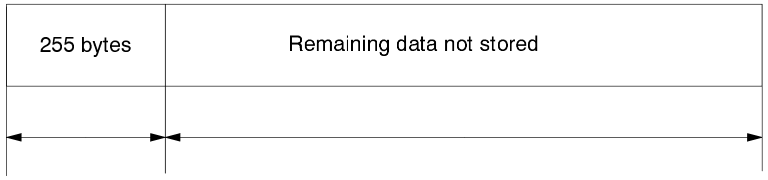
Example: Special I/O Unit #0 will have an address of $10.

The following codes have special meanings.

$00: Unit address of Programmable Controller
$10 to $2F: CPU Bus Units

When the data that is received equals or exceeds the character-type variable size (255 bytes), the character-type variable size portion will be stored in the reception buffer, and the remaining portion will be returned to the user as a data remainder amount. That remaining data can be returned to the user by means of requesting to receive the request command again.

Reception Data



This portion of data will be returned.

This data will be returned 255 bytes at a time, with subsequent requests to receive request commands.

To receive a request command from another device and transmit a response to that command, use "transmission request."

If no data is received from any device when this "receive command" request is executed, waiting-to-receive status will occur.

If a command is received during waiting-to-receive, the data that is received will be stored in the specified buffer. An exit can be forced during that period by pressing the Escape Key.

5-5-5 SRCVT: Receive (With Timer Monitoring)

Operation

Receives commands from other devices, and receives responses to transmissions sent by the Unit itself. The waiting-to-receive time is monitored.

Operation Number

04

Format

```
CALLS SRCVT(NA%,DA%,UN%,TYPE%,SID%,D$,CT%,RST%,RC%,TIM%)
(QuickBASIC)(_xrcv_rst)
```

Arguments

(Output) `NA%` Transmission source network address
(Output) `DA%` Transmission source node address
(Output) `UN%` Transmission source unit address
(Output) `TYPE%` Type of data received: Command (0)/Response (1)
(Output) `SID%` Service ID of data received (0 to 255)
(Input) `D$` Reception buffer (Secure the area in advance.)
(Output) `D%` Reception data
(Output) `CT%` Number of reception data bytes stored in D$
(Output) `RST%` Return status

0	Normal termination
–2	Not open.
–25	Reception timeout
–22	Abnormal termination

(Output) `RS%` Reception buffer size insufficient: Number of bytes of data remaining that cannot be stored
(Input) `TIM%` Waiting-to-receive timer value (unit: 110 ms)

Note If `TIM%` is set to 0, there will be an immediate return when there is no data in the system's reception buffer, and `RST% = –25` (reception timeout) will occur.

Explanation

A command from another device, or a response to a request command transmissions sent by the Unit itself, is received. Along with the data type that is received, the data is stored in the reception buffer.

If some other data is already being received at the time of this reception request, the data is returned will be returned to the user according to the following rules.

When Command Data Only is Received
The command data that is received will be returned to the user in the order in which it is received.

When Response Data Only is Received
The response data that is received will be returned to the user in the order in which it is received.

When Command and Response Data are Received Together
First the response data that is received will be returned to the user in the order in which it is received. When there is no more response data to return, then the command data that is received will be returned to the user in the order in which it is received.

The service ID (SID) here is the same as that specified at the time of command transmission, and will be returned from the destination device as the SID of the response to that command. Thus the correspondence between the command and the response can be recognized by means of specifying the SID.

The transmission data conforms to the portion of the FINS command from MRC onwards. Refer to the *FINS Command Reference Manual (W227)* for more details on the contents of FINS commands.

In addition, the location of the source of the transmitted data will be stored in the transmission source address area, and the number of bytes actually stored in the buffer will be stored in the area for the amount of data received.

The meanings of the transmission source address designations are explained below.

Network Address
This is the network address of the transmission source. The following code has a special meaning.

$00: Same network address

Node Address
This is the node address of the transmission source. The following code has a special meaning.

$00: Same node address

Unit Address
This is the unit address of the transmission destination, and is stored by the absolute address.

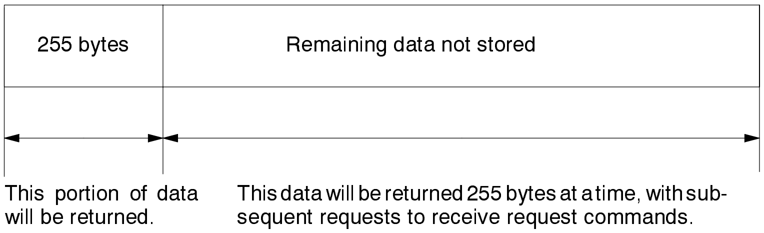
Example: CPU Bus Unit #0 will have an address of $10.

The following codes have special meanings.

$00: Unit address of Programmable Controller
$10 to $2F: CPU Bus Units
$FD: Peripheral tools (e.g., FIT)

When the data that is received from another device equals or exceeds the character-type variable size (255 bytes), the character-type variable size portion will be stored in the reception buffer, and the remaining portion will be returned to the user as a data remainder amount. That remaining data can be returned to the user by means of requesting to receive again.

Reception Data


255 bytes
Remaining data not stored
This portion of data will be returned.
This data will be returned 255 bytes at a time, with subsequent requests to receive request commands.

To receive a request command from another device and transmit a response to that command, use “transmission request.”

If no data is received from any device when this request to receive is executed, waiting-to-receive status will remain in effect until the time specified for the waiting-to-receive timeout has elapsed.

If a command is received during waiting-to-receive, the data that is received will be stored in the specified buffer. If data cannot be received, a reception timeout will occur.

5-6 Sample Programs

The following pages provide sample programs using the FINS Library with Quick BASIC.

```
'/*************************************************/
'/* FINS BASIC Library Sample Program            */
'/* QUICK BASIC                                  */
'/* (Receiving)                                  */
'/*************************************************/
      DEFINT A-Z
                                                          /*Operation number setting*/
      DECLARE SUB SOPEN CDECL ALIAS ”_binit”                        /*Initialization*/
      DECLARE SUB SCLOSE CDECL ALIAS ”_bclose”                     /*End processing*/
      DECLARE SUB SRECV CDECL ALIAS ”_xrecv_rs0”           /*Without timer monitoring*/
      DECLARE SUB SRCVT CDECL ALIAS ”_xrecv_rs1”              /*With timer monitoring*/
      DECLARE SUB SSEND CDECL ALIAS ”_xsend_dat”              /*Transmission request*/

      CLS
      CALLS SOPEN(RST%)                                             /*Initialization*/
      IF RST%<>0 THEN
            PRINT”Driver not installed”:GOTO FINISH
      END IF

      GOSUB RECV                                      /*Command reception processing*/
      GOSUB SEND                                    /*Response transmission processing*/
FINISH:
      CALLS SCLOSE                                   /*Close*/
      PRINT:PRINT”BASIC Interface Sample Program”
      END

'*********************************
'*  Command reception processing  *
'*********************************
RECV:
      TIM%=0
      RCVFLG%=0
      INPUT”Input waiting–to–receive timer value:”,TIM%
      INPUT”...processing to begin:”,X$
RECV1:
      D$=SPACE$(255)                             /*Buffer area secured for reception*/

'Command reception
      CALL SRCVT(NA%,DA%,UN%,TYP%,SID%,D$,CT%,RST%,RC%,TIM%)
      IF RST%<>0 THEN
            PRINT”Command reception error”;RST%:GOTO*FINISH
      END IF

      PRINT
      PRINT ”Command reception data”
      PRINT ”Transmission source network address:”;NA%
      PRINT ”Transmission source node address:”;DA%
      PRINT ”Transmission source Unit number:”;UN%
      PRINT ”Type of data received:”;TYP%
      PRINT ”Service ID:”;SID%
      PRINT ”Bytes of data received:”;CT%
      PRINT ”Bytes of data remaining:”;RC%
      PRINT ”Reception data.. ..0..1..2..3..4..5..6..7..8”;
      PRINT ”..9..a..b..c..d..e..f”
      PRINT ”    ..”;
      FOR I=1 TO CT%
IF(I MOD 16)=1 THEN PRINT:PRINT”    ..”;
B$=MID$(D$,I,1)
PRINT””;RIGHT$(”0”+HEX$(ASC(B$)),2);
      NEXT
      PRINT
      IF RCVFLG%=0 THEN          /*Command keep for response*/
            CMD$=LEFT$(D$,2)     /*(MRC,SRC)*/
            RCVFLG%=12
      END IF
      IF RC%>0 THEN GOTO RECV1   /*When there is data remaining*/
      PRINT ”Command reception  Normal termination”
      RETURN
'
'************************************
'* Response transmission processing *
'************************************
SEND:
      RST%=0                           /*Command end*/
      TYP%=1                           /*Response*/
      D$=CMD$+CHR$(0)+CHR$(0)
      CALL SSEND(NA%,DA%,UN%,TYP%,SID%,D$,RST%)        /*Response reception*/
      IF RST%<>0 THEN
            PRINT”Response transmission error”;RST%:GOTO*FINISH
      END IF
'
      PRINT:PRINT”Response transmission  Normal termination”
      RETURN
```

```
'/*************************************************/
'/* FINS BASIC Library Sample Program            */
'/* QUICK BASIC                                  */
'/* (Transmitting)                               */
'/*************************************************/
```

```
      DEFINT A-Z
                                                            /*Operation number setting*/
      DECLARE SUB SOPEN CDECL ALIAS "_binit"                      /*Initialization*/
      DECLARE SUB SCLOSE CDECL ALIAS "_bclose"                  /*End processing*/
      DECLARE SUB SRECV CDECL ALIAS "_xrecv_rs0" /*Reception request (without timer monitoring)*/
      DECLARE SUB SRCVT CDECL ALIAS "_xrecv_rs1" /*Reception request (with timer monitoring)*/
      DECLARE SUB SSEND CDECL ALIAS "_xsend_dat" /*Command/response transmission*/

      CLS
      CALLS SOPEN(RST%)                                         /*Initialization*/
      IF RST%<>0 THEN
            PRINT"Driver not installed":GOTO FINISH
      END IF

      GOSUB SEND                                      /*Command transmission processing*/
      GOSUB RECV                                      /*Response reception processing*/
FINISH:
      CALLS SCLOSE                                    /*Close*/
      PRINT:PRINT"BASIC Interface Sample Program END"
      END

'**********************************
'* Response reception processing  *
'**********************************
RECV:
      TIM%=100
      D$=SPACE$(255)                    /*Buffer area secured for reception*/

'Response reception
      CALL SRCVT(NA%,DA%,UN%,TYP%,SID%,D$,CT%,RST%,RC%,TIM%)
      IF RST%<>0 THEN
            PRINT"Response reception error";RST%:GOTO*FINISH
      END IF

      PRINT
      PRINT "Response reception data"
      PRINT "Transmission source network address:";NA%
      PRINT "Transmission source node address:";DA%
      PRINT "Transmission source Unit number:";UN%
      PRINT "Type of data received:";TYP%
      PRINT "Service ID:";SID%
      PRINT "Bytes of data received:";CT%
      PRINT "Bytes of data remaining:";RC%
      PRINT "Reception data.. ..0..1..2..3..4..5..6..7..8";
      PRINT "..9..a..b..c..d..e..f"
      PRINT "    ..";
      FOR I=1 TO CT%
IF(I MOD 16)=1 THEN PRINT:PRINT"    ..";
B$=MID$(D$,I,1)
PRINT"";RIGHT$("0"+HEX$(ASC(B$)),2);
      NEXT
      PRINT
      IF RC%>0 THEN GOTO RECV             /*When there is data remaining*/
      PRINT "Response reception  Normal termination"
      RETURN

'***********************************
'* Command transmission processing  *
'***********************************
SEND:
      PRINT
      INPUT"Transmission destination network address:",NA%
      INPUT"Transmission destination node address:",DA%
      INPUT"Transmission destination Unit number:",UN%
      INPUT"Input transmission data.",WRK$'
      INPUT"...processing to begin",x$

      RST%=0          'Command end
      TYP%=3          'Command transmission
      SID%=0          'Service ID
      D$=CHR$(&H9)+CHR$(&H1)+CHR$(0)+CHR$(LEN(WRK$))+WRK$
      CALLS SSEND(NA%,DA%,UN%,TYP%,SID%,D$,RST%)     /*Command transmission*/
      IF RST%<>0 THEN
            PRINT"Command transmission error";RST%:GOTO*FINISH
      END IF

      PRINT:PRINT"Command transmission  Normal termination"
      RETURN
```

SECTION 6
FINS Driver

This section describes the FINS Driver and FINS Driver operations, and provides sample programs using these operations.

6-1 Introduction 138
6-2 Processing Flow 138
6-2-1 Processing Procedure 1 139
6-2-2 Processing Procedure 2 140
6-3 Using the FINS Driver 141
6-3-1 Opening and Accessing the Driver through MS-DOS 141
6-4 FINS Driver Operations 142
6-4-1 Initialize (01) 142
6-4-2 Transmit (02) 143
6-4-3 Receive (03) 145
6-5 Sample Programs 147
6-5-1 Transmitting 147
6-5-2 Receiving 150

6-1 Introduction

When services are used via the CPU bus interface, the CPU Bus Driver can be used as the driver. When this driver is used for event service, however, it is necessary to be aware of the header's data format at the time of transmission or reception.

When the FINS Driver is used for event service, on the other hand, that is not required. Thus the FINS Driver makes it easy to use services via the CPU bus interface.

Note

1. When the FINS Driver is used, the only service that can be employed is event service. For cyclic service or CPU bus link service, use the CPU Bus Interface Communications Driver. For details concerning types of service, refer to *1-4 Communications/Control Services*.
2. The FINS Driver operates using the CPU Bus Driver, so the CPU Bus Driver must be installed before the FINS Driver.
3. The FINS Driver accesses by means directly requesting interrupt number 65H, so /V65 must be specified when the CPU Bus Driver is installed.
4. Sample programs must compiled as follows:
   `CL /Zp`<*source filename*>

Installing the CPU Bus Driver and FINS Driver

The CPU Bus Driver (SBUS.SYS) is recorded in the CONFIG.SYS file in drive F in advance. Be sure that SBUS.SYS is recorded in the active CONFIG.SYS file when a system disk is being modified or a new CONFIG.SYS file is being created.

Add the following line to the CONFIG.SYS file to record the CPU Bus Driver. Refer to *2-6 Installing Device Drivers* for details on other SBUS.SYS parameters. (The F:\CONFIG.SYS file is used when the Unit is started from Built-in ROM.)

```
DEVICE=E:\SBUS.SYS /V65
```

Drive name

The CPU Bus Driver is recorded in the Personal Computer Unit's Built-in ROM (drive E) in advance.

Be sure to include the /V65 parameter when using the FINS Driver (DGIOX.COM). Add the following line to the CONFIG.SYS file to record the FINS Driver. This line must be placed after the SBUS.SYS line.

```
DEVICE=n:\DGIOX.COM
```

Drive name

6-2 Processing Flow

This section will describe the communications procedure (flow) when communications are executed using the FINS Library.

When the FINS Library is used, the only service that can be employed is event service.

The communications procedure when event service is used is as follows:

1, 2, 3...

1. Service request command is transmitted to a device offering a service.
2. Response to the service requested is received from the device offering the service.

This is the communications procedure for event service, so the procedure will be the same when the FINS Library is used. In the case of the library, however, the service is processed using the CPU Bus Driver. Thus the communications data will be transmitted and received via the CPU Bus Driver.

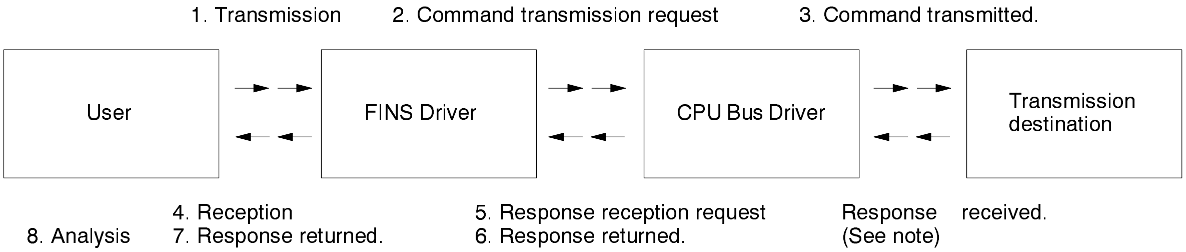
The communications data handled by the FINS Library conforms to the portion of the PC's FINS command from MRC onwards. A maximum of 2,002 bytes of data can be transmitted.

The contents of the PC's FINS commands will vary depending on the contents of the service requested. In addition, the contents of the response will vary according to the command.

Refer to the *FINS Command Reference Manual (W227)* for more details.

6-2-1 Processing Procedure 1

The following diagram shows the procedure followed when a Request Command is Transmitted from the Personal Computer Unit to a Device that Offers a Service, and a Response is Received from that Device



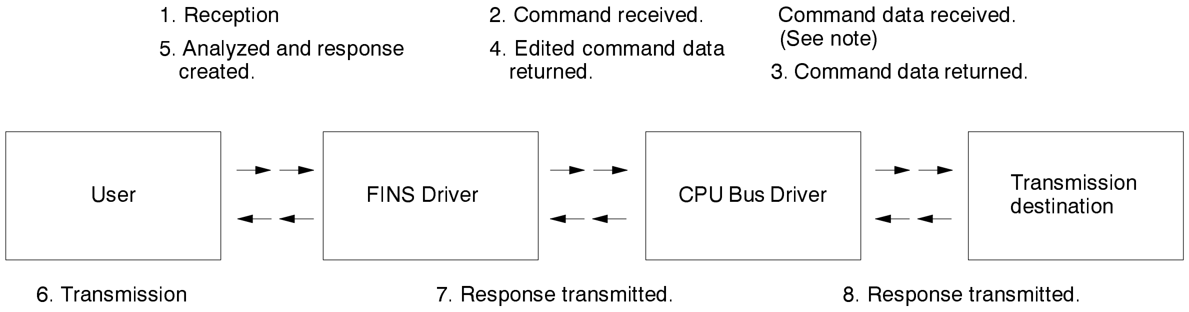
1, 2, 3...

1. The user creates a request command corresponding to the service that is to be requested, and then specifies the device requesting the service and requests "transmit command" from the FINS Driver.
2. The FINS Driver receives the request from the user and requests "transmit command" of the CPU Bus Driver.
3. The CPU Bus Interface Driver receives the "transmit command" request from the FINS Driver, and transmits the specified request command to the specified device.
4. In order to receive a response to the transmitted command, the user requests "receive response."
5. The FINS Driver receives the request from the user, and requests "receive" of the CPU Bus Driver.
6. In answer to the "receive response" requested by the FINS Driver, the CPU Bus Driver returns to the FINS Driver the response data it has received from the command destination.
7. The FINS Driver receives the response data from the CPU Bus Driver and returns it to the user.
8. The user analyzes the contents of the response that was received.

Note The response from the destination Unit will be received by the CPU Bus Driver even if the user doesn't request it.

6-2-2 Processing Procedure 2

The following diagram shows the procedure followed when a Request Command is Received from Another Device, and a Response is Transmitted to the Source of the Command Transmission



1, 2, 3...

1. In order to receive a request command from another device, the user requests "receive command" of the FINS Driver.
2. The FINS Driver receives the request from the user, and requests "receive command" of the CPU Bus Driver.
3. In answer to the "receive command" requested by the FINS Driver, the CPU Bus Driver returns to the FINS Driver the command data it has received from the other device.
4. The FINS Driver receives the command data from the CPU Bus Driver and edits it into a form usable by BASIC, and then returns it to the user.
5. The user analyzes the command data that is received, and creates corresponding response data.
6. Along with creating the response data, the user specifies the source of the request command and calls "transmit" from the FINS Driver.
7. The FINS Driver receives the request from the user, and requests "transmit response" of the CPU Bus Driver.
8. The CPU Bus Driver receives the "transmit response" request from the FINS Driver, and transmits the specified response data to the specified device.

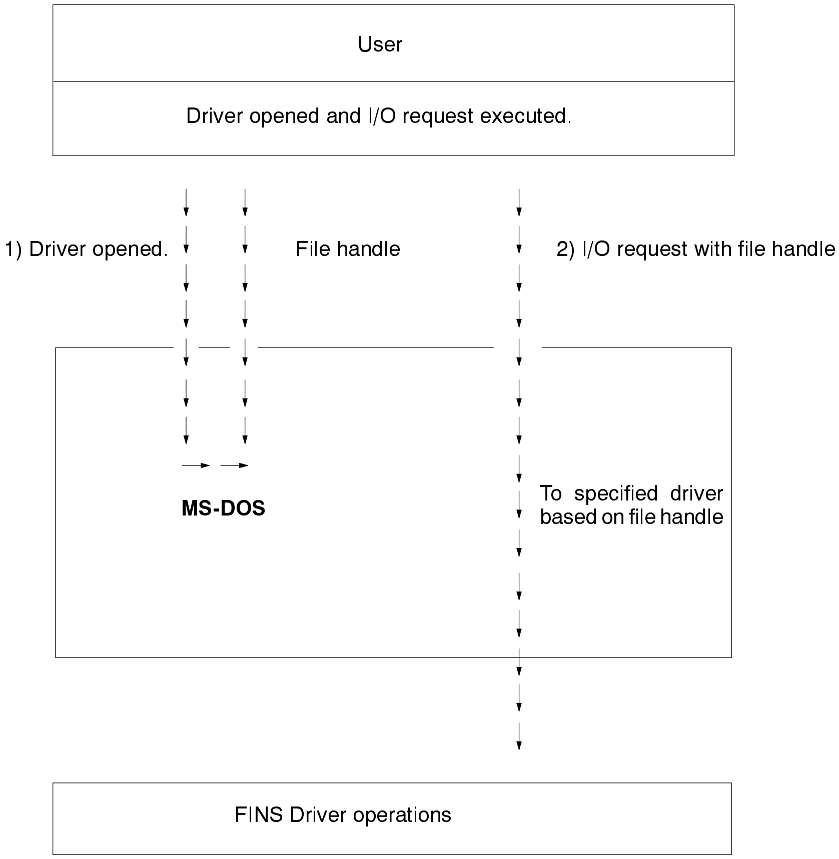
Note The command reception from the other device will be executed regardless of user request.

6-3 Using the FINS Driver

This section will explain how to use FINS Driver.

6-3-1 Opening and Accessing the Driver through MS-DOS

First of all, the driver is opened through MS-DOS, and the “file handle” file identifier is obtained. When the driver services are employed, I/O requests are executed based on this file handle.


User
Driver opened and I/O request executed.
1) Driver opened.
File handle
2) I/O request with file handle
MS-DOS
To specified driver based on file handle
FINS Driver operations

The procedure for using the FINS Driver is as outlined below.

Opening the Driver

File handle is found.

Call procedure:

AH = 3DH
AL = 02H File access mode
DS:DX = Leading address of path name (“FINS”)
INT 21H

Return:

When Carry is Set (Error)

AX = 02H	File does not exist.
AX = 03H	Path name is invalid.
AX = 04H	There are too many files open.
AX = 05H	Access was denied.
AX = 0CH	Access code is invalid.

When Carry is Not Set
AX = File handle (normal termination)

Closing the Driver

The driver is closed.

Call Procedure:

AH = 3EH
BX = File handle
INT 21H

Return:

When Carry is Set (Error)

AX = 06H	File handle is invalid.

When Carry is Not Set

AX = 00H	Normal termination

Requesting I/O

The prescribed I/O parameters are prepared, and the request is made using the file handle.

Call Procedure:

AH = 44H
AL = 03H
BX = File handle
CX = Number of bytes in data buffer
DS:DX = Data buffer segment: offset
(Refer to *6-4 FINS Driver Operations* for details on the data buffer.)
INT 21H

Return:

When Carry is Set (Error)

AX = 01H	Function is invalid.
AX = 05H	Access was denied.
AX = 06H	File handle is invalid.
AX = 0DH	Data is invalid (data buffer error).

When Carry is Not Set

AX = 00H	Normal termination
AX = 81H	Invalid operation to FINS Driver
AX = 85H	Access to FINS Driver denied.
AX = 86H	Invalid file handle to FINS Driver
AX = 8DH	Invalid data to FINS Driver (parameter error)
AX = FFH	Improper command
AX = Other	Return status (described later)

6-4 FINS Driver Operations

This section will explain FINS Driver operations, including contents of data buffers when these operations are used, as well as the return values, for each of the operations.

Operation		Summary
01H	Initialize	Initializes the service.
02H	Transmit	Transmits a request command, or a response, to another device.
03H	Receive	Receives request commands from other devices, and requests to receive responses.

6-4-1 Initialize (01)

Parameter Format

0	(Operation code) 01H

Operation Code Specify 01(HEX).

Return Status AX= 00H Normal termination

Operation This operation clears the reception buffers and initializes the service status.

6-4-2 Transmit (02)

Parameter Format

0	(Operation code) 02H	
1	Transmission buffer address	Offset
2		
3	”	Segment
4		
5	Transmission buffer size	
6		
7	Command (0)/Response (1)	
8	SID(0 to 255)	
9	Transmission destination network address	
10	Transmission destination node address	
11	Transmission destination unit address	

Operation Code:

Specify 02(HEX).

Transmission Buffer Address:

This specifies the leading address of the buffer where the transmission data is stored.

Transmission Buffer Size:

Transmission data size is expressed in number of bytes.

Transmitting a command:	2 to 2,002 bytes
Transmitting a response:	4 to 2,002 bytes

Command/Response:

This specifies either command data or response data to be transmitted.

SID:

This specifies the service ID (SID) when data is transmitted. The SID can be set within a range of 0 to 255. If the SID is specified at the time the command is sent, the same SID will be returned from the destination device in the response to the command. Thus the correspondence between the command and the response can be determined from the SID.

Transmission Destination Network Address:

This specifies the network address of the device to which data is to be transmitted. If “0” is specified, transmission will be within the same network.

Transmission Destination Node Address:

This specifies the node address of the device to which data is to be transmitted. If “0” is specified, transmission will be within the same node.

Transmission Destination Unit Address:

This specifies the unit address of the device to which data is to be transmitted. If “0” is specified, transmission will be within the same unit.

Return Status (AX)

00H:	Normal termination
41H:	Argument specification error
FFH:	Abnormal termination

Operation

This operation transmits data stored in the area specified by the transmission buffer address to the specified destination device as either a command or response, according to the specification. The amount of data transmitted is specified by the size of the transmission buffer.

Correspondence Between Command/Response and Types of Data:

Command/Response	Data
00(HEX)	Request command transmitted to another device
01(HEX)	Response data returned to another device

The transmission data conforms to the portion of the PC's FINS from MRC (main request) onwards. Refer to the *FINS Command Reference Manual (W227)* for more details on the contents of FINS commands.

The meanings of the transmission destination address designations are explained below.

Network Address

This is the network address of the transmission destination. The following code has a special meaning.

$00: Same network address

Node Address

This is the node address of the transmission destination. The following codes have special meanings.

$00: Same node address
$FF: Broadcast to all nodes on specified network

Unit Address

This is the unit address of the transmission destination. Add $10 to a CPU Bus Unit's unit number to calculate the Unit's unit address.

Example: CPU Bus Unit #0 will have an address of $10.

The following codes have special meanings.

$00: Unit address of Programmable Controller
$10 to $2F: CPU Bus Units
$FD: Peripheral Tools (e.g., FIT)
$FE: Communications Units (e.g., SYSMAC NET, SYSMAC LINK)

Note

1. When the data to be transmitted is a response, use the same address (transmission source address) that was specified when the command was received.
2. When transmitting a command that requires a response, use a reception request (response) to receive the response.

6-4-3 Receive (03)

Parameter Format

0	(Operation code) 03H	
1	Timer setting (0=enabled, 1=disabled)	
2	Reception buffer address	Offset
3		
4	”	Segment
5		
6	Reception buffer size	
7		
8	Timer set value	
9	Command (0)/Response (1)	
10	SID(0 to 255)	
11	Transmission source network address	
12	Transmission source node address	
13	Transmission source Unit address	
14	Number of bytes received	
15		

Operation Code (Input):

Specify 03(HEX).

Timer Setting (Input):

This setting indicates whether or not the Unit will wait for a specified time for data reception. The timer set value is described below.

0: Timer setting enabled
1: Timer setting disabled (When the timer setting is disabled, the Unit will wait to receive after the Escape Key is pressed.)

Reception Buffer Address (Input):

This specifies the leading address of the buffer where the reception data is stored.

Reception Buffer Size (Input):

Reception data size is expressed in bytes.
Receiving a response: 4 to 2,002 bytes
Receiving a command: 2 to 2,002 bytes

Timer Set Value (Input):

This setting (0 to 255) specifies the time that the Unit will wait to receive data. The timer set value is specified in 110-ms units, so the set value can be set from 0 to 28.05 s. If the set value is 0, the Unit won’t wait at all.

The set value is used only when the timer setting is enabled.

Command/Response (Output):

This setting indicates whether the data being received is a command from another device or a response to a command transmitted from the Personal Computer Unit.

0: Command
1: Response

SID (Output):

The service ID (SID) is used to match responses with their corresponding commands. If a command is received from another device, return the same SID to the source. The SID can be set within a range of 0 to 255.

Transmission Source Network Address (Output):

Returns the transmission source network address of the received data.

Transmission Source Node Address (Output):

Returns the transmission source node address of the received data.

Transmission Source Unit Address (Output):

Returns the transmission source unit address (absolute address) of the received data.

Number of Bytes Received:

Returns the number of bytes of data received.

Return Status (AX)

00H:	Normal termination
11H:	Timeout
12H:	Reception cancelled by pressing the Escape Key
41H:	Argument specification error
FFH:	Abnormal termination

Operation

This operation receives data and stores it along with a code identifying the type of data (command or response). The data is stored in the area specified by the reception buffer address. The amount of data is specified by the size of the reception buffer.

If data is being received at the time of this reception request, the data will be returned to the user according to the following rules.

When Only Command Data is Received

The command data that is received will be returned to the user in the order in which it is received.

When Only Response Data is Received

The response data that is received will be returned to the user in the order in which it is received.

When Command and Response Data are Received Together

First the response data that is received will be returned to the user in the order in which it is received. When there is no more response data to return, then the command data that is received will be returned to the user in the order in which it is received.

Correspondence Between Command/Response and Types of Data:

Command/Response	Data
00(HEX)	Response to a command transmitted from the Unit
01(HEX)	Command from another device

The reception data conforms to the portion of the FINS command from MRC onwards. Refer to the *FINS Command Reference Manual (W227)* for more details on the contents of FINS commands.

At this point, the location of the source of the transmitted data will be stored in the transmission source address area and the number of bytes actually stored in the buffer will be stored in the area for the number of bytes received.

The meanings of the transmission source address designations are explained below.

Network Address

This is the network address of the transmission source. The following code has a special meaning.

$00: Same network address

Node address.

This is the node address of the transmission source. The following code has a special meaning.

$00: Same node address

Unit Address

This is the unit address of the transmission destination, and is stored by the absolute address. Add $10 to a CPU Bus Unit's unit number to calculate the Unit's unit address.

Example: CPU Bus Unit #0 will have an address of $10.

The following codes have special meanings.

$00: Unit address of Programmable Controller
$10 to $2F: CPU Bus Units

When receiving a request command from another device and transmitting a response back, use a transmission request (response).

If no data has been received when reception is requested and the "timer setting" is enabled, the Unit will wait until data is received or the time specified in the "timer set value" elapses. If the timer times out, return status AX=11H will be returned.

If no data has been received when reception is requested and the "timer setting" is disabled, the Unit will wait until data is received or the Escape Key is pressed. If the Escape Key is pressed, return status AX=12H will be returned.

If even a portion of the data received is stored in the specified buffer, that data will be cleared.

6-5 Sample Programs

The following pages provide sample programs using the FINS Driver. These sample programs can be found on the program disk.

The sample programs will be outlined in the following order:

1, 2, 3...

1. Sample program name
2. Compiling command
3. Program outline

6-5-1 Transmitting

Sample Program Name `net_send.c`

Compiling Command `CL/Zp net_send.c`

Program Outline The program has the following parts:

1, 2, 3...

1. Opening the FINS Driver
2. Initializing the reception buffer
3. Transmitting the command (The network address, node address, and unit address are fixed in the program.)
4. Receiving the response
5. Displaying the response data received
6. Closing the FINS Driver

```
/*********************************************/
/*                                           */
/* FINS Driver Sample Program (Transmitting) */
/*                                           */
/*                                           */
/*                                           */
/*********************************************/

#include        <dos.h>
#include        <stdio.h>

```

```
#define         MAX        2002
#define         CMD        0
#define         RSP        1

typedef  void far      *farptr;
typedef  unsigned char  uchar;
typedef  unsigned int   unit;
typedef  unsigned short ushort;

struct   cmd02{                       /*Transmission*/
uchar    s_cmd;                       /*Command 02*/
uchar    far *bp;                     /*Transmission buffer address*/
ushort   req_byte;                    /*Transmission buffer size*/
uchar    wrtype;                   /*Type of data transmitted (0: Command/1: Response)*/
uchar    sid;                         /*SID*/
uchar    net_add;                     /*network add*/
uchar    nod_add;                     /*node add*/
uchar    port_add;                    /*port add*/
};

struct   cmd03{                       /*Reception*/
uchar    s_cmd;                       /*Command 03*/
uchar    timflg;                      /*Timer monitor flag (Monitoring: 0: Yes/1: No)*/
uchar    far *bp;                     /*Reception buffer address*/
ushort   req_byte;                    /*Reception buffer size*/
uchar    u_timer;                     /*Timer value*/
uchar    rdtype;                    /*Type of data received (0: Command/1: Response)*/
uchar    sid;                         /*SID*/
uchar    net_add;                     /*network add*/
uchar    nod_add;                     /*node add*/
uchar    port_add;                    /*port add*/
uchar    rd_byte;                     /*Number of bytes actually received*/
};

union    REGS   inregs,outregs;       /*I/O register structure*/
unit     fd;                          /*File handle*/
unit     length;                      /*Number of bytes to be transmitted*/
unit   *buf;
static   unsigned char rdt[MAX];      /*Reception data buffer*/
static   unsigned char sdt[MAX]=      /*Transmission data buffer*/
/*MRC SRC*/
  0x09,0x01,0,10,1,2,3,4,5,6,7,8,9,0
};


void     errclose();

void
main()
{
uchar    init;                        /*Initialization packet*/
struct   cmd02 snd;                   /*Transmission request packet*/
struct   cmd03 rcv;                   /*Reception request packet*/
int            i;

/*
```

```
** Initialization
*/
inregs.h.ah = 0x3d;                   /*Opens driver. (1)*/
inregs.h.al = 0x02;
inregs.x.dx =(uint)”FINS”;
intdos(&inregs, &outregs);
if(outregs.x.cflag!=0)
         printf(”Driver not loaded./n”);
         printf(”/tError code = 0x%x/n”outregs.x.ax);
         exit(1);
  }
fd = outregs.x.ax;                    /*Saves file handle.*/

init = 0x01;                          /*Initializes. (2)*/
length = 1;
buf = (uint*)&init;
if (ioctl())
         errclose();

/*
** Command data transmission
*/
snd.s_cmd = 0x02;                     /*Transmits command (3)*/
snd.bp = (farptr)sdt;
snd.req_byte = 14;                    /*Number of digits requested for transmission*/
snd.wrtype = CMD;                     /*Type of data transmitted*/
snd.sid = 0;                          /*Service ID*/
snd.net_add = 0x01;                   /*Network number*/
snd.nod_add = 0x02;                   /*Node number*/
snd.port_add = 0x15;                  /*Unit number*/
length = sizeof(struct cmd02);        /*Structure size*/
buf = (unit*)&snd;
if(ioctl())                           /*Requests command reception*/
         errclose();

/*
** Response data reception
*/
rcv.s_cmd = 0x03;                     /*Receives response (4)*/
rcv.timflg = 0;                       /*Timer monitor flag*/
rcv.bp = (farptr)rdt;                 /*Storage buffer address*/
rcv.req_byte = MAX;                   /*Maximum number of digits for reception*/
rcv.u_timer = 100;                    /*Timer value*/

length = sizeof(struct cmd03);        /*Structure size*/
buf = (unit*)&rcv;
if(ioctl())                           /*Reception request*/
         errclose();

/*
** Display of received data (5)
*/
printf(*The response data received is as follows:*/n”);
printf(*Transmission source is*/n”);
printf(*/tNetwork address:0x%x/n”,rcv.net_add);
printf(*/tNode address:0x%x/n”,rcv.nod_add);
printf(*/tUnit address:0x%x/n”,rcv.port_add);
printf(*/tType of data received:0x%x/n”,rcv.rd_byte);
printf(*/tService ID:0x%x/n”,rcv.sid);
printf(*/tNumber of bytes for reception:%d/n”,rcv.rd_byte);
printf(”Reception data is/n/t”);
for(i=0;i<rcv.rd_byte;i++)
         printf(”0x%02x”,rdt[i]);
printf(”/n”);

/*
**End processing
*/
inregs.h.ah = 0x3e;                   /*Closes driver (6)*/
inregs.x.bx = fd;                     /*File handle*/
intdos(&inregs, &outregs);            /*INT 21H*/

printf(”Normal termination/n”);
}

int
ioctl()
{
inregs.x.ax = 0x4403;               /*I/O request*/
inregs.x.bx = fd;                   /*File handle*/
         inregs.x.cx = length;      /*Sets number of bytes for transmission.*/
inregs.x.dx = (short)buf;           /*Sets parameter buffer address.*/
intdos(&inregs, &outregs);          /*INT 21H*/
         if(outregs.x.cflag||outregs.x.ax)
         return(1);
return(0);
}

void
errclose()
{
printf(”IOCTL error/n”);
printf(”/tcmd=0x%x carry=0x%x AX=0x%x/n”);
                     buf[0],outregs.x.cflag,outregs.x.ax);
inregs.h.ah = 0x3e;                   /*Closes driver.*/
inregs.x.bx = fd;                     /*File handle*/
intdos(&inregs, &outregs);            /*INT 21H*/
exit(2);
}
```

6-5-2 Receiving

Sample Program Name `net_recv.c`

Compiling Command `CL/Zp net_recv.c`

Program Outline The program has the following parts:

1, 2, 3...

1. Opening the FINS Driver
2. Initializing the reception buffer
3. Receiving a command
4. Displaying the command data received
5. Creating response data (MRES, SRES)
6. Transmitting the response

7. Closing the FINS Driver

```
/******************************************/
/*                                        */
/* FINS Driver Sample Program (Receiving) */
/*                                        */
/*                                        */
/*                                        */
/******************************************/

#include        <dos.h>
#include        <stdio.h>

#define         MAX        2002
#define         CMD        0
#define         RSP        1

typedef  void far    *farptr;
typedef  unsigned char  uchar;
typedef  unsigned int   unit;
typedef  unsigned short ushort;

struct   cmd02{                        /*Transmission*/
uchar    s_cmd;                        /*Command 02*/
uchar    far *bp;                      /*Transmission buffer address*/
ushort   req_byte;                     /*Transmission buffer size*/
uchar    wrtype;                  /*Type of data transmitted (0: Command/1: Response)*/
uchar    sid;                          /*SID*/
uchar    net_add;                      /*network add*/
uchar    nod_add;                      /*node add*/
uchar    port_add;                     /*port add*/
};

struct   cmd03{                        /*Reception*/
uchar    s_cmd;                        /*Command 03*/
uchar    timflg;                       /*Timer monitor flag (Monitoring: 0: Yes/1: No)*/
uchar    far *bp;                      /*Reception buffer address*/
ushort   req_byte;                     /*Reception buffer size*/
uchar    u_timer;                      /*Timer value*/
uchar    rdtype;                    /*Type of data received (0: Command/1: Response)*/
uchar    sid;                          /*SID*/
uchar    net_add;                      /*network add*/
uchar    nod_add;                      /*node add*/
uchar    port_add;                     /*port add*/
uchar    rd_byte;                      /*Number of bytes actually received*/
};

union    REGS   inregs,outregs;        /*I/O register structure*/
unit     fd;                           /*File handle*/
unit     length;                       /*Number of bytes to be transmitted*/
unit   *buf;
static   unsigned char rdt[MAX];       /*Reception data buffer*/
static   unsigned char sdt[MAX]=       /*Transmission data buffer*/
                      /*MRC SRC*/
                           0x00,0x00,0
                           };

```

```
void     errclose();

void
main()
{
uchar                 init;           /*Initialization packet*/
struct   cmd02 snd;                   /*Transmission request packet*/
struct   cmd03 rcv;                   /*Reception request packet*/
int            i;

/*
**Initialization
*/
inregs.h.ah = 0x3d;                   /*Opens driver (1)*/
inregs.h.al = 0x02;
inregs.x.dx =(uint)”FINS”;
intdos(&inregs, &outregs);
if(outregs.x.cflag!=0)
         printf(”Driver not loaded./n”);
         printf(”/tError code = 0x%x/n”outregs.x.ax);
         exit(1);
}
fd = outregs.x.ax;                    /*Saves file handle*/

init = 0x01;                          /*Initializes (2)*/
length = 1;
buf = (unit*)&init;
if (ioctl())
         errclose();

/*
**Command data transmission
*/
rcv.s_cmd = 0x03;                     /*Reception (3)*/
rcv.timflg = 0;                       /*Timer monitor flag*/
rcv.bp = (farptr)rdt;
rcv.req_byte = MAX;                   /*Maximum number of digits for reception*/
rcv.u_timer = 100;                    /*Timer value*/

length = sizeof(struct cmd03);        /*Structure size*/
buf = (uint*)&rcv;
if(ioctl())
         errclose();

printf(*Command data received*/n”);   /*(4)*/
printf(*Transmission source*/n”);
printf(*/tNetwork address:0x%x/n”,rcv.net_add);
printf(*/tNode address:0x%x/n”,rcv.nod_add);
printf(*/tType of data received:0x%x/n”,rcv.rdtype);
printf(*/tService ID:0x%x/n”,rcv.sid);
printf(*/tUnit address:0x%x/n”,rcv.port_add);
printf(*/tNumber of bytes for reception:%d/n”,rcv.rd_byte);
printf(”Reception data is/n/t”);
for(i=0;i<rcv.rd_byte;i++)
         printf(”0x%02x”,rdt[i]);
printf(”/n”);
```

```

sdt[0] = rdt[0];                    /*Main request (5)*/
sdt[1] = rdt[1];                    /*Sub-request*/
sdt[2] = 0;                         /*Main response*/
sdt[3] = 0;                         /*Sub-response*/

/*
**Response data transmission
*/
snd.s_cmd = 0x02;                     /*Transmission (6)*/
snd.net_add = rcv.net_add;            /*Network number*/
snd.nod_add = rcv.nod_add;            /*Node number*/
snd.port_add = rcv.port_add;          /*Unit number*/
snd.sid = rcv.sid;                    /*Service ID*/
snd.wrtype = RSP;                     /*Response*/
snd.bp = (farptr)sdt;
snd.req_byte = 4;                     /*Number of digits requested for transmission*/

length = sizeof(struct cmd02);        /*Structure size*/
buf = (uint*)&snd;
if(ioctl())
         errclose();

/*
**End processing
*/
inregs.h.ah = 0x3e;                   /*Closed driver. (7)*/
inregs.x.bx = fd;                     /*File handle*/
intdos(&inregs, &outregs);            /*INT 21H*/
printf(”Normal termination/n”);
}

int
ioctl()
{
inregs.x.ax = 0x4403;                 /*I/O request*/
inregs.x.bx = fd;                     /*File handle*/
inregs.x.cx = length;                 /*Sets number of bytes for transmission*/
inregs.x.dx = (uint)buf;              /*Sets parameter buffer address*/
intdos(&inregs, &outregs);            /*INT 21H*/
if(outregs.x.cflag||outregs.x.ax)
         return(1);
return(0);
}

void
errcloes()
{
printf(”IOCTL Error/n”);
printf(”/tcmd=0x%x carry=0x%x AX=0x%x/n”);
                     buf[0],outregs.x.cflag,outregs.x.ax);
inregs.h.ah = 0x3e;                   /*Closes driver.*/
inregs.x.bx = fd;                     /*File handle*/
intdos(&inregs, &outregs);            /*INT 21H*/
exit(2);
}
```

SECTION 7
RS-232C Communications

This section describes the RS-232C communications functions and how to use them.

7-1 RS-232C Port Specifications 156
7-1-1 Communications Specifications 156
7-1-2 Interface Specifications 156
7-2 Serial Communications Library 156
7-3 Communications Control Functions 157
7-3-1 LOPEN() (Open RS-232C Communications Port) 157
7-3-2 LCLOSE() (Close RS-232C Communications Port) 158
7-3-3 LREAD() (Store Data) 158
7-3-4 LWRITE() (Transmit Data) 159
7-3-5 LNCTL() (Control Signals) 160
7-3-6 LNSTS() (Check Signal Status) 161
7-3-7 LDCCTL() (Set XON/XOFF) 161
7-3-8 LDCRST() (Stop XON/XOFF) 162
7-3-9 LRXCNT() (Check Reception Buffer) 162
7-3-10 LTXCNT() (Check Amount of Data Transmitted) 163
7-4 RS-232C Baud Rate Limitations 163
7-4-1 Baud Rates for C (Microsoft C) 163
7-4-2 Baud Rates for Quick BASIC 163
7-5 RS-232C Response Errors 164
7-5-1 Programmable Terminals 164
7-5-2 Bar Code Reader, OCR, or Card Reader 164
7-5-3 Workstation or Personal Computer 165
7-5-4 Sample Program 165

7-1 RS-232C Port Specifications

7-1-1 Communications Specifications

The following table shows the Personal Computer Unit's RS-232C communications specifications.

Item	Specification
Number of ports	2 max.
Synchronization	Start /Stop
Wiring method	Dedicated line, full duplex
Baud rate (See note.)	75, 150, 300, 600, 1200, 2400, 4800, 9600, or 19200 bps
Connection method	RS-232C
Data length	7 or 8 bits
Stop bits	1 or 2 bits
Parity	None, even, odd
Reception buffer capacity	512 bytes/port

Note The communications baud rate is limited when the program that performs RS-232C communications also reads or writes data through the CPU Bus Driver or on the RAM disk. Refer to *7-4 RS-232C Baud Rate Limitations* for details.

7-1-2 Interface Specifications

RS-232C ports 1 and 2 both conform to IBM PC/AT serial interface standards. The model numbers of the compatible OMRON connector socket and hood are XM2D-0901 and XM2S-0913. The maximum cable length is 15 m.

Pin Allocation

Pin no.	Signal symbol	Signal name	Signal direction
1	CD	Carrier Detect	Input
2	RD (RXD)	Receive Data	Input
3	SD (TXD)	Send Data	Output
4	ER (DTR)	Data Terminal Ready	Output
5	SG (GND)	Signal Ground	---
6	DR (DSR)	Data Set Ready	Input
7	RS (RTS)	Request to Send	Output
8	CS (CTS)	Clear to Send	Input
9	CI (RI)	Call Incoming	Input
Base	FG	Protective ground	---

7-2 Serial Communications Library

A serial communications library with a Microsoft C compiler interface is provided on the Personal Computer Unit's program disk in directory \CSIO. The following table shows the files provided for each program memory model. Use the file appropriate for the program being used.

Memory model	File name
Small model	CSIOS.LIB
Medium model	CSIOM.LIB
Compact model	CSIOC.LIB
Large model	CSIOL.LIB
Huge model	

The following table lists the C functions provided in the RS-232C communications package. These functions are described in detail in *7-3 Communications Control Functions*.

Function	Description	Page
LOPEN()	Opens an RS-232C port.	157
LCLOSE()	Closes an RS-232C port.	158
LREAD()	Stores reception data.	158
LWRITE()	Transmits data.	159
LNCTL()	Controls the communications control signals.	160
LNSTS()	Checks the status of communications control signals.	161
LDCCTL()	Sets XON/XOFF control.	161
LDCRST()	Stops XON/XOFF control.	162
LRXCNT()	Checks the amount of data in the reception buffer.	163
LTXCNT()	Checks the amount of transmission data.	163

Note The function names are written in upper-case letters. If the functions are written in the program in lower-case, don't specify the /NOI option when linking the program.

7-3 Communications Control Functions

This section describes the operation of the communications control functions.

7-3-1 LOPEN() (Open RS-232C Communications Port)

Operation

The `LOPEN` function initializes the communications port that is to be used and starts RS-232C communications.

Format

```
unsigned int LOPEN (ipno, ibps, istp, ipari, ilng, iflg)
int ipno;
unsigned int ibps;
int istp;
int ipari;
int ilng;
unsigned int *iflg;
```

Parameters

ipno: port number
Select either “1” or “2.”

ibps: baud rate
Select the baud rate: 75, 150, 300, 600, 1200, 2400, 4800, 9600, or 19200 bps. The communications baud rate is limited when the same program also reads or writes data through the CPU Bus Driver or on the RAM disk. Refer to *7-4 RS-232C Baud Rate Limitations* for details.

istp: stop bits
Select the number of stop bits: 1 = 1 stop bit, 2 = 2 stop bits.

ipari: parity
Specify the parity: 1 = None, 2 = Even, 3 = Odd.

ilng: data length
Specify the data length: 7 = 7 bits, 8 = 8 bits.

iflg: termination status
This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

0: Normal termination
1: Parameter error
2: Open
16: Hardware error

Related Functions LCLOSE

7-3-2 LCLOSE() (Close RS-232C Communications Port)

Operation

The LCLOSE function stops RS-232C communications at the specified communications port.

Format

```
unsigned int LCLOSE (ipno, iflg)
int ipno;
unsigned int *iflg;
```

Parameters

ipno: port number
Select the port that is to be closed, either “1” or “2.”

iflg: termination status
This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

0: Normal termination
1: Incorrect port number specification
2: The specified port isn’t open.
16: Hardware error

Explanation

At the end of the program, be sure to close any ports that were opened by the LOPEN function. Failing to do so may result in faulty operation.

Related Functions LOPEN

7-3-3 LREAD() (Store Data)

Operation

This function reads data from the RS-233C reception buffer and stores it in the specified area.

Format

```
unsigned int LREAD (ipno, n, iary, iflg)
int ipno;
unsigned int n;
char *iary;
unsigned int *iflg;
```

Parameters

ipno: port number
Select the port number, either “1” or “2.”

n: number of bytes to read
Specify the number of bytes (number of characters requested for reception) to store in the specified area from the reception buffer.

If the reception buffer doesn’t contain the specified number of bytes of data, the Personal Computer Unit will wait for 30 seconds. A timeout will occur if the specified amount of data isn’t received within 30 seconds.

The LRXCNT() function can be used to determine the amount of data in the reception buffer before executing LREAD().

iary: storage area
This pointer indicates where the received data is to be stored.

iflg: termination status
This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

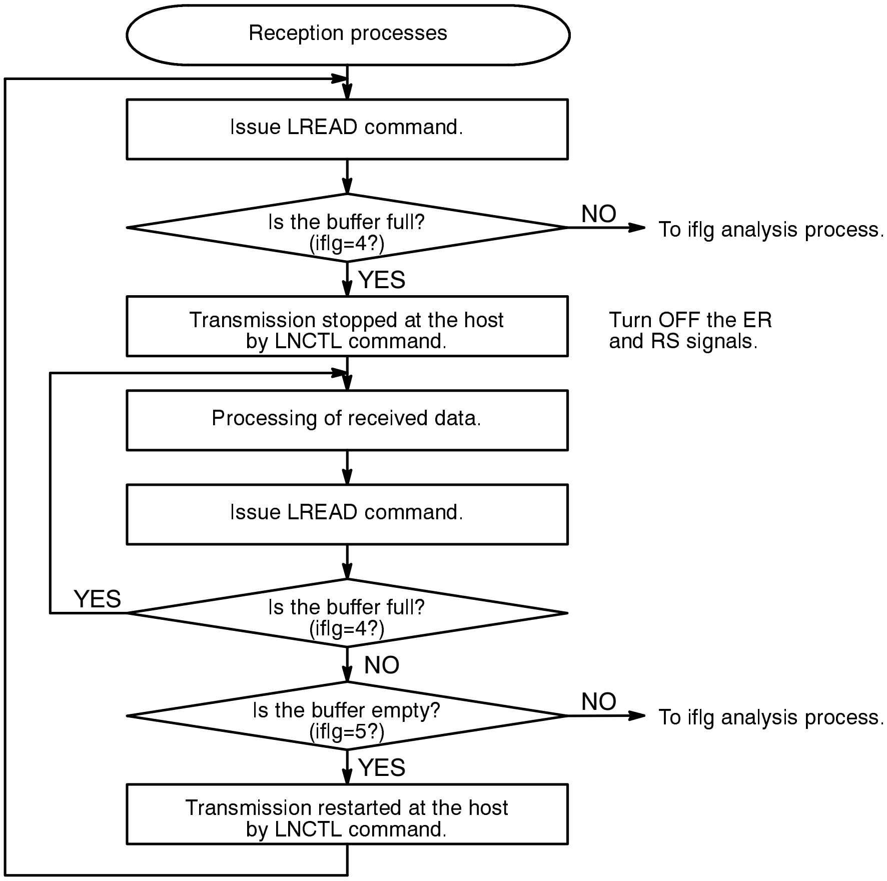
0: Normal termination
1: Parameter error
2: Specified port not open
3: Reception timeout (30 s)

4: Buffer full
5: Buffer empty
6: Buffer overflow
16: Hardware error

Explanation

The amount of free space in the reception buffer is increased by the amount of data read from the buffer with the LREAD() function. If XON/XOFF control is specified by executing the LDCCTL() function before the LREAD() function, XON and XOFF codes will be transmitted automatically in accordance with the amount of free space in the buffer.

There is sufficient space allocated to each port's reception buffer in the library, but the reception buffer might fill up if too much time passes before the LREAD() function is executed. The XON/XOFF control command or the following kind of processing must be used to prevent the buffer from filling up.


Reception processes
Issue LREAD command.
Is the buffer full? (iflg=4?)
NO
To iflg analysis process.
YES
Transmission stopped at the host by LNCTL command.
Turn OFF the ER and RS signals.
Processing of received data.
Issue LREAD command.
YES
Is the buffer full? (iflg=4?)
NO
Is the buffer empty? (iflg=5?)
NO
To iflg analysis process.
YES
Transmission restarted at the host by LNCTL command.

Related Functions `LDCCTL, LRXCNT`

7-3-4 LWRITE() (Transmit Data)

Operation

The `LWRITE` function transmits data to the specified port.

Format

```
unsigned int LWRITE (ipno, n, iary, iflg)
int ipno;
unsigned int n;
char *iary;
unsigned int *iflg;
```

Parameters

ipno: port number
Select the port number, either "1" or "2."

n: number of bytes to write

Specify the number of bytes (number of characters requested for transmission) of data that will be transmitted.

The LWRITE function won't end until the specified number of bytes has been transmitted unless XON/XOFF control is being used and an XON reception timeout occurs. (The LTXCNT() function can be used to determine the amount of data that was transmitted before the XON reception timeout occurred.)

iary: storage area

This pointer indicates where the transmission data is stored.

If XON/XOFF control is specified by executing the LDCCTL() function before the LWRITE() function, data transmission will be controlled automatically based on the XON and XOFF codes from the destination device.

iflg: termination status

This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

0: Normal termination
1: Parameter error
2: Specified port not open
3: XON reception timeout occurred
16: Hardware error

Related Functions

`LDCCTL, LRXCNT`

7-3-5 LNCTL() (Control Signals)

Operation

The `LNCTL` function controls communications signals for the specified port.

Format

```
unsigned int LNCTL (ipno, ictl, iflg)
int ipno;
int ictl;
unsigned int *iflg;
```

Parameters

ipno: port number

Select the number of the port, either “1” or “2,” where the communications signals are to be controlled.

ictl: signal control

Select the control code to be used. The control codes and their functions are shown below:

0: Turns ER signal OFF, and checks that DR signal is OFF.
1: Turns ER signal ON, and checks that DR signal is ON.
2: Turns RS signal OFF, and checks that CS signal is OFF.
3: Turns RS signal ON, and checks that CS signal is ON.
4: Turns BREAK signal OFF.
5: Turns BREAK signal ON.

iflg: termination status

This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

0: Normal termination
1: Parameter error
2: Specified port not open
8: After ER signal has been turned OFF, DR signal does not turn OFF.
9: After ER signal has been turned ON, DR signal does not turn ON.
10: After RS signal has been turned OFF, CS signal does not turn OFF.
11: After RS signal has been turned ON, CS signal does not turn ON.
16: Hardware error

Related Functions

`LNSTS`

7-3-6 LNSTS() (Check Signal Status)

Operation This function reads the status of specified port's signals.

Format

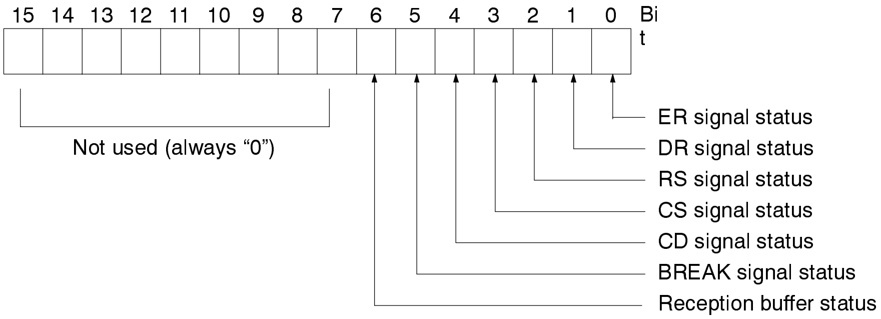
```
unsigned int LNSTS (ipno, ists, iflg)
int ipno;
unsigned int *ists;
unsigned int *iflg;
```

Parameters

ipno: port number
Select the number of the port, either "1" or "2," where the communications signal status is to be checked.

ists: signal status flags
This pointer indicates where the signal status flags of the port are to be stored. The status of each signal is indicated in this word as shown below:

Signal Status



A bit that is ON indicates that the signal is ON.

When the reception buffer status flag is ON, the buffer is empty; when it is OFF, there is data in the buffer.

Note The BREAK signal will remain ON until the Personal Computer Unit receives valid data.

iflg: termination status
This pointer indicates where the termination status of the function is to be stored. The meanings of the termination codes are shown below:

0: Normal termination
1: Parameter error
2: Specified port not open
16: Hardware error

Related Functions `LNCTL`

7-3-7 LDCCTL() (Set XON/XOFF)

Operation The `LDCCTL` function sets XON/XOFF control.

Format

```
unsigned int LDCCTL (ipno, imode, itout)
int ipno;
int imode;
unsigned int itout;
```

Parameters

ipno: port number
Select the number of the port, either "1" or "2," where the XON/XOFF control will take place.

imode: XON/XOFF control mode
This value indicates the control mode that will be used. The three modes are:

1: XON/XOFF control only during transmission.
2: XON/XOFF control only during reception.
3: XON/XOFF control during both transmission and reception.

itout: timeout value
Specify the set value of the timeout (0 to 65535), when transmitting data, from the time the XOFF code is received to the time the next XON code is received. A value can be specified within a range of 0.1 s to 6553.5 s (in units of 0.1 s). If "0" is specified, it will be considered 0.1 s.

Explanation

This function will remain in effect for the relevant port until either the `LCLOSE` function or the `LDCRST` function is executed. Be sure to execute LDCCTL() before the communications port is opened with LOPEN().

If a parameter error is made or XON/XOFF control isn't set correctly, XON/XOFF control won't take place and the XON code (DC1:11H) and XOFF code (DC3:13H) will be handled as ordinary data if they are received.

Related Functions `LREAD, LWRITE, LDCRST, LTXCNT`

7-3-8 LDCRST() (Stop XON/XOFF)

Operation The `LDCRST()` function stops XON/XOFF control.

Format

```
unsigned int LDCRST (ipno)
int ipno;
```

Parameters

ipno: port number
Select the number of the port, either "1" or "2," where XON/XOFF control will be stopped.

When this function is executed, XON/XOFF control is stopped and XON/XOFF codes (DC1:11H and DC3:13H) can't be transmitted. If these codes are transmitted after LDCRST() has been executed, they will be handled as ordinary data.

Related Functions `LDCCTL`

7-3-9 LRXCNT() (Check Reception Buffer)

Operation The `LRXCNT` function checks the reception buffer for the amount of data that has been received.

Format

```
unsigned int LRXCNT (ipno, n)
int ipno;
unsigned int *n;
```

Parameters

ipno: port number
Select the number of the port, either "1" or "2," where the amount of data in the reception buffer is to be checked.

n: number of bytes in buffer
This variable contains the number of bytes in the buffer.

Note The value returned here will be the number of data elements received by the Personal Computer Unit and stored in the reception buffer but not yet read from the buffer with the LREAD function. The value returned by the LRXCNT function will thus decrease as data is read out by the LREAD function.

Related Functions LREAD

7-3-10 LTXCNT() (Check Amount of Data Transmitted)

Operation

The LTXCNT function checks the amount of data successfully transmitted by the last LWRITE function.

Format

```
unsigned int LTXCNT (ipno, n)
int ipno;
unsigned int *n;
```

Parameters

ipno: port number
Select the number of the port, either “1” or “2,” where the amount of transmitted data is to be checked.

n: number of bytes transmitted
This variable contains the number of bytes transmitted by the last LWRITE function.

When XON/XOFF control is being used, LTXCNT() can be used to find out how much data was transmitted when transmission has been stopped by an XON reception timeout.

Related Functions LWRITE

7-4 RS-232C Baud Rate Limitations

The rate of communications through the RS-232C ports is limited when the program receiving data also reads or writes data with the CPU Bus Driver or on the RAM Disk. If the baud rates listed in the following tables are exceeded, data might not be received correctly.

7-4-1 Baud Rates for C (Microsoft C)

With C, the maximum baud rate is lower if two ports are being used than it is when one port is being used. Also, the maximum baud rate is lower for port #2 than port #1 if data is being transferred through the CPU bus.

CPU bus/RAM Disk Usage	Number of ports being used	Maximum baud rate	
		Port #1	Port #2
When reading/writing through the CPU bus	1	9600 bps	9600 bps
	2	4800 bps	2400 bps
When reading/writing on the RAM Disk	1	9600 bps	9600 bps
	2	4800 bps	4800 bps
When reading/writing through the CPU bus and on the RAM Disk	1	4800 bps	4800 bps
	2	4800 bps	2400 bps

7-4-2 Baud Rates for Quick BASIC

CPU bus/RAM Disk Usage	Number of ports being used	Maximum baud rate
		Ports #1 and #2
When reading/writing through the CPU bus	1 or 2	2400 bps
When reading/writing on the RAM Disk	1 or 2	9600 bps
When reading/writing through the CPU bus and on the RAM Disk	1 or 2	2400 bps

7-5 RS-232C Response Errors

7-5-1 Programmable Terminals

Data can be transmitted to the Personal Computer Unit by using either function keys or the touch panel. If a resend message is displayed, then input the data again.

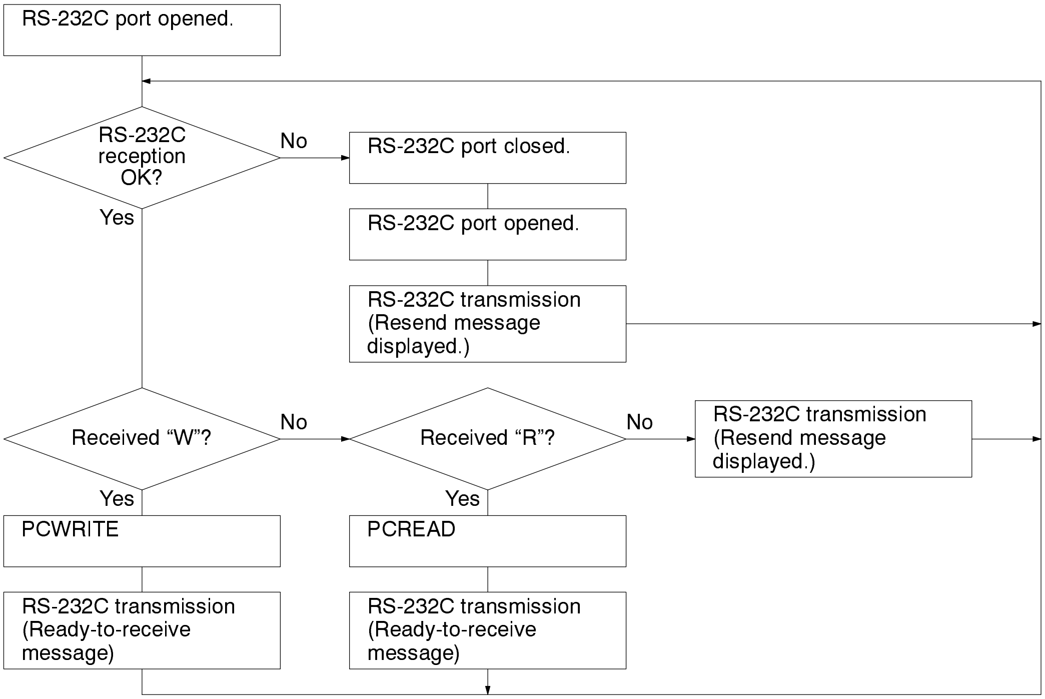
At the Personal Computer Unit, the following decisions will be made when the data is received.

Is there a hardware error?
A hardware error is a parity error or an overrun error.

Was data used by the program received?
This refers to input mistakes or missing data.

If those two questions check out okay, the received data will be used. If there is an error, a message will be displayed at the PT requesting that the data be input again. The Personal Computer Unit will wait for the data to be input.

Flowchart (at Personal Computer Unit)
In this example, PCWRITE will be executed if a “W” is received, and PCREAD will be executed if an “R” is received.

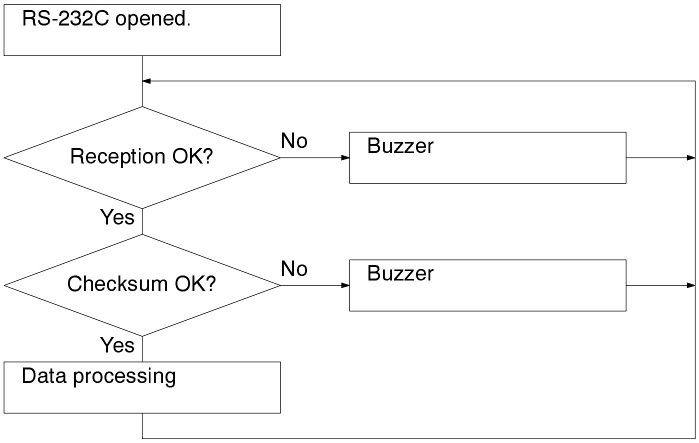

RS-232C port opened.
RS-232C reception OK?
No
Yes
RS-232C port closed.
RS-232C port opened.
RS-232C transmission (Resend message displayed.)
Received “W”?
No
Yes
Received “R”?
No
Yes
RS-232C transmission (Resend message displayed.)
PCWRITE
PCREAD
RS-232C transmission (Ready-to-receive message)
RS-232C transmission (Ready-to-receive message)

7-5-2 Bar Code Reader, OCR, or Card Reader

These devices themselves do not have the capability to resend or display. Therefore it is necessary for reception errors and data to be displayed at the Personal Computer Unit. In addition, checksums and other data checks must be conducted for data received.

Flowchart Example (at Personal Computer Unit)

In this example, data will be read, including the checksum. A buzzer will sound if there is a reception error.



7-5-3 Workstation or Personal Computer

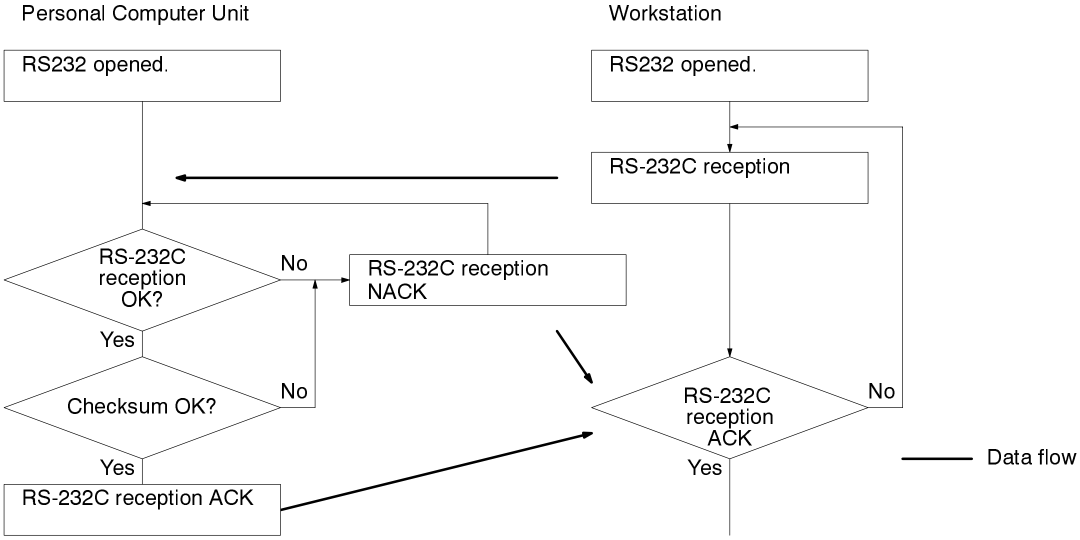
Depending on the program, there may or may not be a reciprocal error resend protocol. Suppose that the Personal Computer Unit is receiving and a workstation is sending. In the situations described below, the Personal Computer Unit will transmit NACK to the workstation and wait for a resend.

When there is a hardware error. A hardware error is an overrun error or a parity error.

When there is a checksum error for the data received.

If the transmission is normal, the Personal Computer Unit will send ACK to the workstation.

Flowchart Example: Personal Computer Unit and Workstation



7-5-4 Sample Program

A sample program, using the protocol described above with a BASIC Unit and a Personal Computer Unit, is provided below. The Personal Computer Unit is programmed with Quick BASIC.

Explanation of Program At the Personal Computer Unit, data sent from the BASIC Unit will be transmitted to the Programmable Controller at an RS-232C baud rate of 19.2 kbps. When the RS-232C data is received, a hardware error check and data checksum will be executed. If no errors are found, then ACK will be transmitted to the BASIC Unit. If an error is found, then NACK will be sent.

Personal Computer Unit In the main routine, data will be written to the DM area of the Programmable Controller using the *PCWRITE* function. When the RS-232C data is received, a data checksum will be executed. If no errors are found, then ACK will be transmitted and reflected in the PCWRITE transmission buffer. If there is a checksum error, then NACK will be sent.

If there is a reception error, the RS-232C port will close once and then reopen. In addition, by modifying the received data a checksum error will result and NACK will be transmitted.

```
'**************************************************
'*           RS-232C Sample Program               *
'*                 QUICK BASIC                    *
'**************************************************

DIM BUF%(100)

DECLARE SUB pcwrite CDECL ALIAS "_b_pcwrite"
DECLARE SUB pcopen CDECL ALIAS "_b_pcopen"
DECLARE SUB pcmsrd CDECL ALIAS "_b_pcmsrd"
DECLARE SUB pcclose CDECL ALIAS "_b_pcclose"

ON ERROR GOTO errproc
ON KEY(10) GOSUB eproc
KEY(10) ON

      OPEN "COM1:19200,N,8,1,ASC,RB32767" FOR RANDOM AS #4 LEN = 256

      ON COM(1) GOSUB rsin
      COM(1) ON

      sub$ = "@D,0,100,%100H" 'DM word 100
      CALLS pcopen(rtn%)
      IF rtn%<> 0 THEN PRINT "pcopen error rtn= "; rtn%: GOTO eproc
      CALLS pcwrite(ne%, no%, sub$, BUF%(0), rtnb%)
      IF rtnb% = 3 GOTO flg'Programmable Controller busy
      IF rtnb%<> 0 THEN PRINT "pcwrite error rtn= "; rtnb%: GOTO eproc
GOTO flg

rsin:
      COM(1) OFF

'232 reception
      INPUT #4,reccv$
      reclen% = LEN(reccv$)

'SUM calculation
      sum% = 0
      FOR i = 1 TO LEN(reccv$) - 3
            sum% = sum% + ASC(MID$(reccv$,i,1))
      NEXT i
'Checksum
```

```
      IF RIGHT$(reccv$,3) = RIGHT$(STR$(sum%),3) THEN
            PRINT #4, ”ACK”: eee% = 0’Data normal
      ELSE
          PRINT #4, ”NACK”: eee% = eee% + 1 ’Checksum error
      END IF

’Transmission data creation
      IF eee%=0 THEN
            FOR I=1 TO LEN(reccv$)–3
                  IF I=100 GOTO LB1
                  BUF%(I–1)=ASC(MID$(reccv$,I,1))
            NEXT I
       END IF

LB1:
       COM(1) ON
RETURN

’Error processing
errproc:
       count = count + 1
  reccv$ = ”ERROR JUMP!!!!”Checksum processing resulting in NACK
       CLOSE #4
       OPEN ”COM1:19200,N,8,1,ASC,RB32767” FOR RANDOM AS #4 LEN = 256
RESUME NEXT

eproc:
       pcclose (rtn%)
END
```

BASIC Unit

With the present time as the data to be transmitted, a 3-character checksum will be performed. After transmission, data will be received from the other Unit. If the data is ACK, then the next data will be transmitted. If it is NACK, then the same data will be sent again.

```
 10 REM TEST
 20 OPTION LENGTH 538
 30 PARACT 0 WORK 1000
 40 OPEN ”COM2:19200,8,N,1,CS50000,X,DS0” AS #2
 50 D$=TIME$
 60 SUM%=0
 70 ’SUM calculation
 80 FOR I=1 TO LEN(D$)
 90 SUM%=SUM%+ASC(MID$(D$,I,1))
100 NEXTI
110 D$=D$+RIGHT$(STR$(SUM%),3)
120 PRINT #2,D$
130 INPUT #2,ANS$
140 IF ANS$= ”ACK” THEN *LB
150 FOR I=0 TO 100 : NEXT I
160 GOTO 120
170 *LB
180 GOTO 50
190 END PARACT
```

SECTION 8
Controlling User Indicators

The user can control the status of some of the Personal Computer Unit's LED indicators. This section describes how to use these indicators.

8-1 The User Indicators 170

8-1-1 User Indicator Location 170

8-1-2 Controlling User Indicators 170

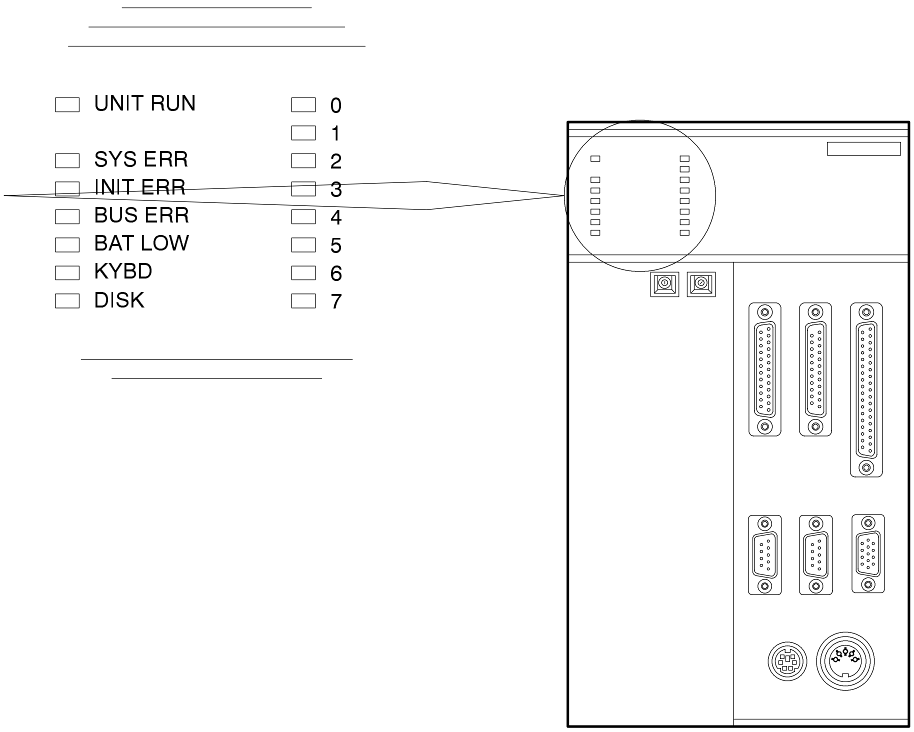
8-1-3 Checking the Indicators 171

8-2 Sample Program 171

8-1 The User Indicators

8-1-1 User Indicator Location

There are two types of indicators in the Personal Computer Unit's display area: system indicators and user indicators. The indicators numbered from 0 to 7 are the user indicators.

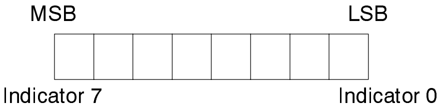


The user indicators can be turned on and off from the program. These indicators can be used to check the operating status when the Personal Computer Unit is being operated without a monitor or other display device.

Note The user indicators will light to indicate CPU bus errors that occur when the Personal Computer Unit boots up. The indicators can be turned off by the user in this case, too.

8-1-2 Controlling User Indicators

User indicator lights are turned on and off by controlling an I/O port at the Personal Computer Unit. The eight bits of I/O Port 390H correspond to indicators 0 to 7.



An indicator will be OFF when its corresponding bit is ON (1), and ON when its corresponding bit is OFF (0). The initial value of this byte is FF (all indicators off).

The indicators can be turned on and off by outputting the values shown here to the I/O port. To output to the I/O port, use the OUT instruction.

8-1-3 Checking the Indicators

You can use the DEBUG command, which is included in system debugging, to actually turn indicators on. The operation is as follows:

1, 2, 3...

1. Display the MS-DOS prompt.
2. Input `DEBUG` and then press the Enter Key. The prompt (–) for the DEBUG command will be displayed.
3. Input `0 390 FE` and then press the Enter Key. User indicator 0 will light.
4. Input `Q` and then press the Enter Key to quit the DEBUG command.

8-2 Sample Program

A program example using Quick BASIC is given below. In this example, the corresponding indicators will light when the numbers 0 to 7 are input from the keyboard. If all seven of the numbers are input, then all of the indicators will light.

```
LED:
INPUT ″Input a number from 0 to 7″, IN%
IF IN%>7 OR IN%<0 THEN PRINT ″Out of range″:GOTO LED .... (1)
SUM% = SUM% OR (2 ^ IN%) ................................. (2)
OUT &H390, (SUM% XOR &HFF) ............................... (3)
IF SUM% <> &HFF GOTO LED ................................. (4)
OUT &H390, &HFF .......................................... (5)
END
```

1, 2, 3...

1. Checks the range of the number that was input.
2. Turns ON the bit corresponding to the number.
3. Inverts the bit and outputs it.
4. Repeats the operation as long as all of the indicators are not lit.
5. Turns off all of the indicators and quits.

SECTION 9
Using Error Logs

This section describes how to use the error log, which records errors that occur during operation of the Unit.

9-1 Error Logs 174
- 9-1-1 Recording Error Records 174
- 9-1-2 Error Log Recording Area 174
- 9-1-3 Record Format 174
- 9-1-4 Reading Error Records 174

9-2 Writing Error Records 174
- 9-2-1 Procedure 174
- 9-2-2 Function Calls 175
- 9-2-3 Writing Error Log Contents 175

9-3 Reading Error Records 176
- 9-3-1 Reading Procedure 176
- 9-3-2 Function Calls 176
- 9-3-3 Writing Error Record Information 177

9-4 Errors at BIOS Startup 177

9-1 Error Logs

The error log is the part of the Built-in RAM Disk (drive F) used to record any errors that occur when the Unit is started or during execution of the program.

9-1-1 Recording Error Records

Error records will be recorded in the following two cases.

BIOS Startup

The results of the self-diagnosis when BIOS is started will be automatically recorded as an error record. The following errors will be recorded.

- Battery errors
- Checksum errors of the built-in ROM disk (drive E), BIOS-ROM, and the built-in RAM disk (only the error log area of drive F)
- System configuration
- Other hardware errors

Application Execution

From an application program, function calls can be used to record an error record. Consider recording the following kinds of errors.

- I/O errors
- CPU errors at the PC
- Other program execution errors

9-1-2 Error Log Recording Area

The error log is recorded in a 512-byte memory area beginning from the top of the memory (64-K RAM) used as built-in RAM (drive F). Each error record takes up 8 bytes and 63 records can be logged. The remaining 8 bytes are used for the checksum.

9-1-3 Record Format

Error records are recorded in FIFO (First In, First Out) format in a memory area of the built-in RAM disk. If the memory area becomes full, the data that was recorded earliest is deleted as new data is added.

When more than 63 errors are logged in the Personal Computer Unit, the oldest records are deleted as new errors are logged.

9-1-4 Reading Error Records

You can display the error log by inputting the ERRLOG command at the MS-DOS prompt. The ERRLOG command is stored in the ROM disk. While the MS-DOS prompt is being displayed, enter `errlog` and then hit the Enter Key. The following display will then appear on the screen.

```
             DATE          TIME          TYPE        CODE
   No.00   11-04-94      14:20:56         02          03
   No.01   11-04-94      15:10:58         02          06
               ↑             ↑             ↑           ↑
     Date of occurrence  Time of occurrence  Type of error  Error code
```

9-2 Writing Error Records

This section will explain how to write error records from an application program.

9-2-1 Procedure

The procedure for writing an error record is as follows:

1, 2, 3...

1. Create in the memory a buffer for writing.
2. Store in the buffer the error record data that is to be written.
3. Execute a function call by means of a software interrupt, and write the error record.

9-2-2 Function Calls

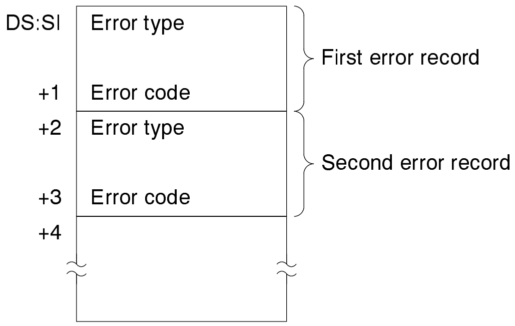
Error codes can be written by using software interrupt INT15H to execute a function call.

- **Software Interrupt Number:** INT15H
- **Input Parameters**
  AH = 21H: Function number
  AL = 01H: Write designation
  CX = Number of error records written
  DS:SI = Error record buffer address
- **Output Parameters**
  CF = Error flag

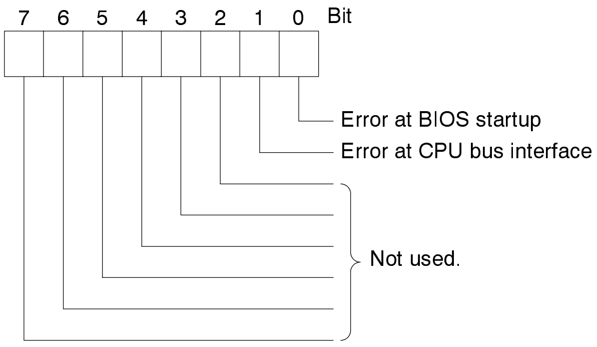
When a function call is executed with the above settings, the error record data stored in the designated error record buffer will be written, with date and time information added, in the specified number of error records.

9-2-3 Writing Error Log Contents

Error log contents can be written as shown below to the area designated as the error log buffer.



- **Type of Error**
  The code that distinguishes error types is recorded by turning ON and OFF the bits in one byte of data.



- **Error Code**
  The error code is recorded with one byte of data. It will differ according to the error type code. For details on the error code for error type O1H (when BIOS is started up), refer to *9-4 Errors at BIOS Startup*.

9-3 Reading Error Records

This section will explain how to read error records from an application program.

9-3-1 Reading Procedure

The procedure for reading an error record is as follows:

1, 2, 3...

1. Create in memory the buffer for reading.
2. Execute a function call by means of a software interrupt, and store the error record data in the buffer.
3. Read the error record data stored in the buffer.

9-3-2 Function Calls

Error codes can be read by using software interrupt INT15H to execute a function call. The reading will be executed in order, beginning with the oldest error record. There are two methods for reading error records. One is to read new error records one by one, and the other is to repeatedly read the same error record.

Reading New Error Records

When this method is used, the error record pointer is refreshed moves from the newly read data to the next data written. Therefore, data read by this method is not read when the next function call is executed.

- **Software Interrupt Number:** INT15H
- **Input Parameters**
  AH = 21H: Function number
  AL = 00H: Read designation
  CX = Number of error records read
  ES : DI = Error record buffer address
  When this operation is used, a buffer of CX × 8 bytes must be protected.
- **Output Parameters**
  [When input parameter CX = 0]
  CX = Present number of error records
  [When input parameter CX ≠ 0]
  CX = Number of error records read
  CF = Error flag

When a function call is executed with the above settings, the error record information, with date and time information added, will be read for the designated number of error records and stored in the buffer.

Reading the Same Record

When this method is used, the error record pointer does not move. Therefore, data read by this method can also be read when the next read operation is executed.

- **Software Interrupt Number:** INT15H
- **Input Parameters**
  AH = 21H: Function number
  AL = 02H: Read designation
  CX = Number of error records read
  ES : DI = Error record buffer address
  When this operation is used, a buffer of CX × 8 bytes must be protected.
- **Output Parameters**
  [When input parameter CX = 0]
  CX = Present number of error records
  [When input parameter CX ≠ 0]
  CX = Number of error records read
  CF = Error flag

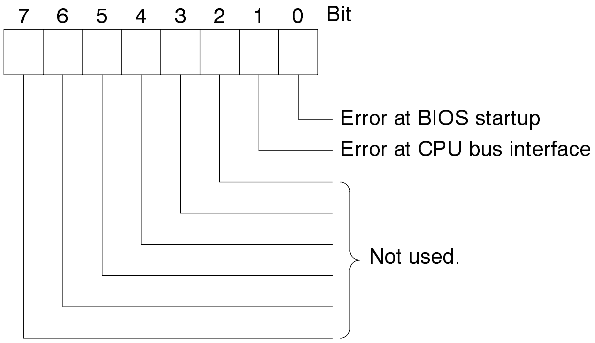
When a function call is executed with the above settings, the error record information, with date and time information added, will be read for the designated number of error records and stored in the buffer.

9-3-3 Writing Error Record Information

The format of error record information stored in the area designated as the error record reading buffer is as shown below.

DS:SI	Year (last 2 digits)	One error record
+1	Month	
+2	Day	
+3	Time	
+4	Minute	
+5	Second	
+6	Error type	
+7	Error code	
.	Year	
.	.	
.	.	

- **Type of Error**
  The code that distinguishes error types is recorded by turning ON and OFF the bits in one byte of data.



- **Error Code**
  The error code is recorded by one byte of data. It will differ according to the error type code. For details on the error code for error type O1H (when BIOS is started up), refer to *9-4 Errors at BIOS Startup*.

9-4 Errors at BIOS Startup

If an error should occur when the Personal Computer Unit's BIOS is started, it will be recorded in an error record as an error code for error 01H. The meanings for error codes at that time are as follows:

Error code	Meaning
01h	CPU register test in progress
02h	CMOS write/read error
03h	ROM BIOS checksum error
04h	Programmable interval timer error
05h	DMA initialization error
06h	DMA page register write/read error
08h	RAM refresh verification error
09h	First 64K RAM test in progress
0Ah	First 64K RAM chip or data line error, multi-bit
0Bh	First 64K RAM odd/even logic error
0Ch	Address line error first 64K RAM
0Dh	Parity error first 64K RAM

Error code	Meaning
10h	Bit 0 first 64K RAM error
11h	Bit 1 first 64K RAM error
12h	Bit 2 first 64K RAM error
13h	Bit 3 first 64K RAM error
14h	Bit 4 first 64K RAM error
15h	Bit 5 first 64K RAM error
16h	Bit 6 first 64K RAM error
17h	Bit 7 first 64K RAM error
18h	Bit 8 first 64K RAM error
19h	Bit 9 first 64K RAM error
1Ah	Bit 10 first 64K RAM error
1Bh	Bit 11 first 64K RAM error
1Ch	Bit 12 first 64K RAM error
1Dh	Bit 13 first 64K RAM error
1Eh	Bit 14 first 64K RAM error
1Fh	Bit 15 first 64K RAM error
20h	Slave DMA register error
21h	Master DMA register error
22h	Master interrupt mask register error
23h	Slave interrupt mask register error
25h	Interrupt vector loading in progress
27h	Keyboard controller test error
28h	CMOS power error and checksum calculation in progress
29h	CMOS configuration validation in progress
2Bh	Screen initialization error
2Ch	Screen retrace test error
2Dh	Search for video ROM in progress
2Eh	Screen running with video ROM
30h	Screen operable
31h	Monochrome monitor operable
32h	Color monitor (40 column) operable
33h	Color monitor (80 column) operable
34h	Timer tick interrupt test in progress or error
35h	Shutdown test in progress or error
36h	Gate A20 error
37h	Unexpected interrupt in protected
38h	RAM test in progress or address error > FFFFh
3Ah	Interval timer channel 2 test or error
3Bh	Time-of-Day clock test or error
3Ch	Serial port test or error
3Dh	Parallel port test or error
3Eh	Math coprocessor test or error
41h	System board select error
42h	Extend CMOS RAM error
80h	OS-ROM checksum error
82h	RAM DISK #1 power error
83h	RAM DISK #2 power error
84h	OS-RAM DISK checksum error
85h	Error log area checksum error

SECTION 10
Precautions

This section describes some important differences between the CV500-VP1□1-E and CV500-VP2□□-E versions of Personal Computer Unit and general precautions when setting up and using the Units.

10-1 Operating Environment 180
10-2 Differences from the CV500-VP111/121-E Units 180
10-3 Other Precautions 181
10-3-1 Installing the CPU Bus Driver 181
10-3-2 Resetting the Personal Computer Unit 181
10-3-3 Installing CVSS/SSS 182

10-1 Operating Environment

Observe the following precautions when using a Hard Disk Unit or Personal Computer Unit.

When using a Personal Computer Unit, the ambient operating temperature must be between 0°C and 50°C for the Backplane and all CV-series Units.

When using a Hard Disk Unit, the ambient conditions must be within the specifications for the Personal Computer Unit, Backplane, and all CV-series Units, as shown in the following table.

Item	Specification
Temperature	5°C to 45°C (maximum temperature change: 15°C/h)
Maximum Vibration	When operating: 0.3G (10 to 150 Hz) When not operating: 1.0G (150 to 300 Hz)
Maximum Shock	When operating: 5G 3 times each in X, Y, and Z directions When not operating: 30G 3 times each in X, Y, and Z directions

10-2 Differences from the CV500-VP111/121-E Units

The following table shows the differences between the CV500-VP1□1-E and CV500-VP2□□-E versions of Personal Computer Unit.

Item	Difference (CV500-VP2□□ specification)
OS	The operating system was changed from AX-DOS to MS-DOS and the DOS version was changed from DOS 3.21 to DOS 5.0.
Program	MS-DOS function calls are upward compatible. BIOS calls are compatible except for the Japanese portion of the video BIOS. Hardware is upward compatible except for the video portion.
CPU bus	The calling interface is unchanged. Application programs can be used with recompiling, but an ICF setting is required in the CPU Bus Driver. Currently, a 0 setting doesn't need to be changed.
CV500-MR261	Can't be used unless the device driver is installed.
3-mode FDD	The Japanese 1.2MB floppy format can't be used unless the driver (3MODEFDD.SYS) is installed.
CV500-MR262 CV500-MP601	Can't be used.
Startup errors	The DIP switch can be set so that system startup will continue even if an error occurs during startup. Refer to the *Operation Manual* for details on "battery-less" systems.
CPU Bus Driver	The options have been increased. Previously, the number of reception buffers was fixed at 14, but can now be set from 1 to 14. This reduces the amount of memory occupied by SBUS.SYS. The ICF can be specified now. Commands that don't require responses can be created.
Hard disk	The hard disk type must be set with the hardware setup utility. Refer to the *Operation Manual* for details on System Parameters.
CPU Bus Library	The quick library for QuickBASIC corresponds to Version 4.5. When using Version 4.2, refer to *2-4 Changing the Quick Library Version*.

10-3 Other Precautions

10-3-1 Installing the CPU Bus Driver

Escape Key Inputs

The keyboard driver must be installed before the CPU Bus Driver (SBUS.SYS) in order for the Escape Key to be used to interrupt the Unit when it is waiting to receive data.

The MS-DOS keyboard driver (KEYB.COM) is a .COM file, so it can't be installed in the CONFIG.SYS file. In this case, the ADDDRV.EXE command can be used, allowing the CPU Bus Driver (SBUS.SYS) to be installed after the keyboard driver.

Example: AUTOEXEC.BAT

```
.
.
PATH E:\
KEYB.COM JP, 932, E:\KEYBOARD.SYS
ADDDRV F:\SBUS.ADD
```

Example: SBUS.ADD

```
DEVICE=E:\SBUS.ADD
```

Saving Memory

When the CPU Bus Driver is loaded without any options, it takes up 50K-bytes of resident memory. This occurs because memory is allocated to reception buffers in the driver, and these can be reduced with the /b option.

When it isn't necessary to collect reception responses or commands, save memory by reducing the number of reception buffers.

With MS-DOS/V, the user memory can be increased up to 640K-bytes by placing device drivers in UMB.

Example: CONFIG.SYS

```
DOS=HIGH,UMB
DEVICE=E:\HIMEM.SYS
DEVICE=E:\EMM386.EXE RAM I=E000–EFFF FRAME=E000
DEVICEHIGH=E:\SBUS.SYS /V65 /B1
DEVICEHIGH=DGIOX.COM
```

10-3-2 Resetting the Personal Computer Unit

The Personal Computer Unit can be reset by the following four methods:

1, 2, 3...

1. Resetting with RESET.EXE
   When the CPU Bus Driver is installed, the Personal Computer Unit can be reset by itself with RESET.EXE. The CPU bus will be initialized.
2. Power Supply Reset
   Reset the power supply. The PC and I/O Units will also be reset.
3. Reset from the PC
   The Personal Computer Unit can be reset by turning ON the corresponding Reset Bit in the Auxiliary Area.
4. Keyboard Reset, Reset from the program

   There are other forms of resets, such as pressing the Control+Alt+Delete Keys, or executing HWSET, FDISK, or SWITCH commands. These resets will reset the Personal Computer Unit but won't reset the CPU bus, so the CPU bus will become unusable afterwards. In this event, restart the Unit with a Power Supply Reset or Reset from the PC.

Note Initialization of the CPU bus refers to initialization with respect to the Personal Computer Unit. To completely initialize the CPU bus, perform a Power Supply Reset.

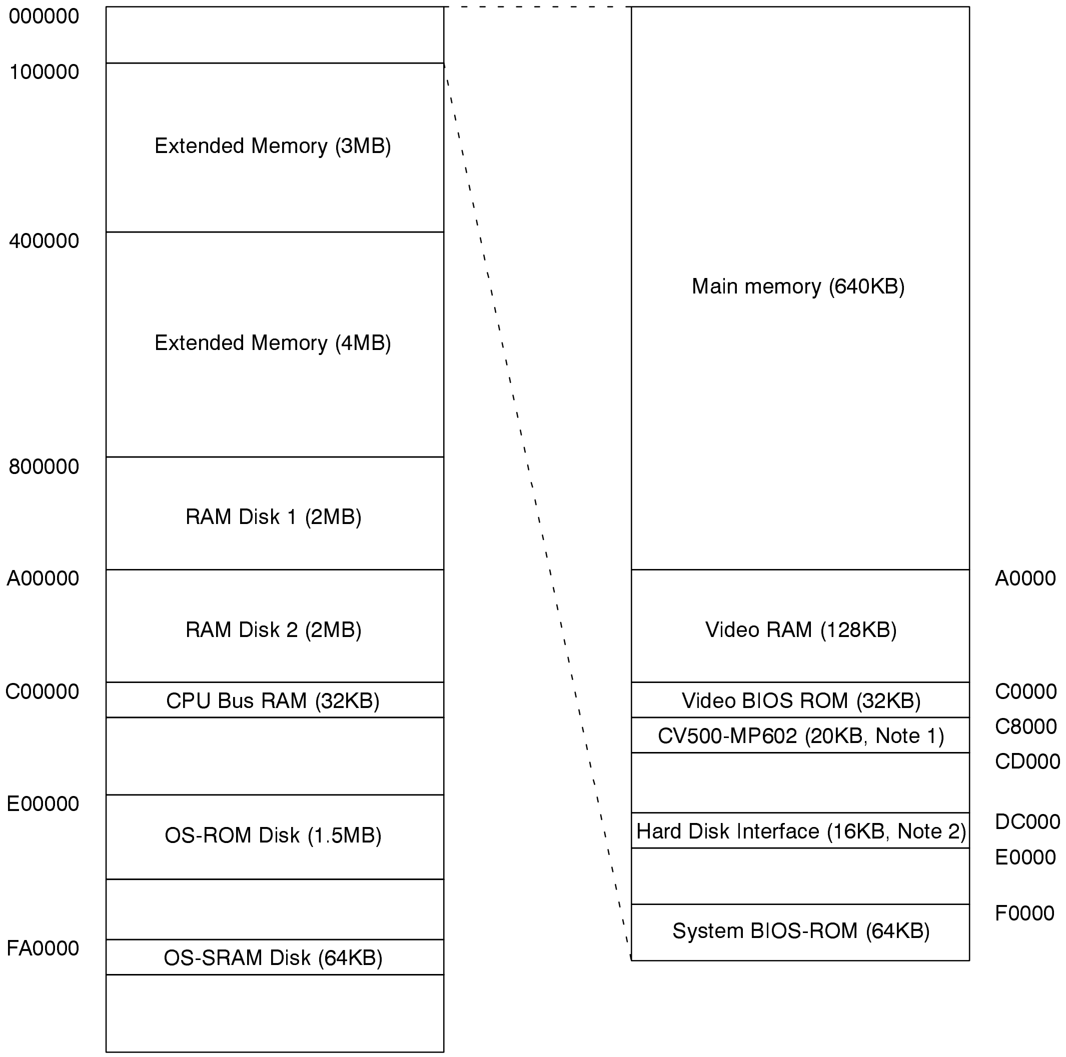
10-3-3 Installing CVSS/SSS

Specify install on a hard disk when installing the CV Support Software or SYSMAC Support Software.

If use of a RAM disk is specified, the CVSS/SSS uses the VDISK.SYS for the RAM disk driver. The Personal Computer Unit normally supports RAMDRIVE.SYS rather than VDISK.SYS, so be sure to change the CONFIG.SYS file to specify the correct RAM disk driver.

Appendix A
Memory Configuration

The memory configuration for the CV500-VP2□□-E Personal Computer Units is as follows:



Note 1. Can be changed with software settings. Refer the Personal Computer Unit's *Operation Manual* for details.

2. Can be changed with jumper pin settings. Refer the Personal Computer Unit's *Operation Manual* for details.

Glossary

ANSI escape sequence A screen control method standardized by the ANSI (American National Standard Institute). This method allows sophisticated screen control with the Esc key and a character code.

backup A copy of data or a program made to protect against loss of the original data or program.

BIOS An acronym for basic input/output system. The BIOS is a part of the control program constructing the operating system of CV500-VP111/121-E Personal Computer Units. This part of the control program relies on the hardware and its main role is to control input and output data exchanged between a Personal Computer Unit and peripheral devices.

BIOS call One method for using the subroutine functions included in BIOS. A number can be set in the CPU register so that a subroutine will be executed when a software interrupt occurs.

buffer A portion of the memory of a personal computer used to store data temporarily, e.g., often during communications.

command interpreter The program that interfaces user commands with the operating system. (COMMAND.COM)

COMMAND.COM A program that executes an MS-DOS command that is input to a Personal Computer Unit. The COMMAND.COM program usually located in the main memory is active when an MS-DOS prompt appears on screen.

compiler Software for translating programs from programming language to machine language. Execution is faster with a compiler than with an interpreter.

CPU bus A transmission path used between CPU Bus Units on a Programmable Controller.

CPU bus link One method for communicating between CPU Bus Units using the CPU bus. A portion of the memory for each Unit is maintained for communications, and communications are conducted by refreshing all the Units so that the contents will always be the same. Although it enables communications without complex procedures, it has the disadvantage of placing a load on the Programmable Controller.

CPU Bus Unit A Unit that connects to the CPU bus of a Programmable Controller. A Personal Computer Unit, SYSMAC NET Link Unit, and BASIC Unit are CPU Bus Units.

current directory The directory which is the subject of data input and output. If an I/O instruction is given without designating a directory, the input or output will be executed with respect to the current directory.

current drive The location in a computer from which data is input or output. If a data input command or data output command is executed without designating the drive, the data will be input to or output from the current drive. The drive may be a physical device or a part of memory defined as a drive.

directory A ‘container’ within memory used to store files. It is possible to make another directory in a directory.

EMS
An acronym for expanded memory specification. The EMS eliminates the limitation of 640K-byte memory controlled by the MS-DOS so that the MS-DOS can use more memory.

environmental parameter
Character string data with a name stored on the memory of a personal computer that includes conditions required to execute the program. An environmental parameter can be set using the SET command.

event
A function of the BASIC language. When an event occurs such as a special key being pressed or a timer setting elapsing, the operation currently in progress can be interrupted and a special operation executed.

FAT
An acronym for file allocation table. The FAT indicates the locations of files in the disk. Whenever a file is written to the disk, the contents of the FAT is refreshed. If the FAT is damaged, there is no way to find the locations of data in the disk.

hard disk
A high-capacity disk drive which data can be read from or written to at high speed. A hard disk is vulnerable to shock, vibration, and dust.

I/O table
A table created within the memory of the Programmable Controller that lists the IR area words allocated to each Unit in the Programmable Controller System. The I/O table can be created by, or modified from, a Programming Device.

LED
An acronym for light emitting diode. An LED is lit with a low current at a low voltage and mainly used for an indicator.

memory card
A card that incorporates ROM and RAM.

MONITOR Mode
The mode in which memory contents can be changed while the Programmable Controller is operating.

network address
A number assigned to a network in order to distinguish it from other networks to which it is connected. When designating a node within the same network, “0” is specified as the network address.

node address
A number assigned to a node to distinguish it from other nodes connected within the same network.

PROGRAM mode
The mode in which the Programmable Controller can be programmed. While the Programmable Controller is in this mode, operation is stopped and instructions can be input.

Programmable Terminal
A display device sometimes used with a Programmable Controller. When a Programmable Controller, Personal Computer Unit, and so on, are connected, it can be used as a simple display and I/O device.

RAM
An acronym for random access memory. Data can be read from and written to the designated location of the RAM of a personal computer according to the command input. All data will be lost if the computer is turned power off. Most memory of the computer consists of RAM.

RAM disk
RAM in a personal computer used as a disk. Data can be read from and written to a RAM disk at high speed compared with a hard disk. All data in the RAM disk of a personal computer will be lost if the personal computer is turned off. The RAM disk of a Personal Computer Unit is backed up by a battery so that data will not be lost if the Personal Computer Unit is turned off.

resident program
A program that stays in the memory of the personal computer. The available space of the memory for other programs is reduced according to the size of the resident programs.

ROM

An acronym for read-only memory. Data can be read from a ROM but usually no data can be written to it. Data in the ROM disk of a personal computer is not lost if the personal computer is turned off. A ROM is used to store programs and data that are frequently used.

ROM disk

ROM used as a disk. Data can be read from a ROM disk at high speed and no data will be lost by mistake. To write data to a ROM disk, a special device such as a PROM writer is required.

RS-232C

A standard interface established by the Electronic Industries Association (EIA) for connecting and exchanging data among computers or between computers and peripheral devices.

RUN mode

The mode in which the Programmable Controller normally operates. Programs stored in the Programmable Controller are executed in this mode.

SCSI

An acronym for small computer systems interface. The SCSI specifications are standardized specifications used to connect a personal computer and a disk drive or printer.

unit address

A number assigned to a Unit connected to the Programmable Controller in order to distinguish it from other Units in network communications. For CPU Bus Units, $10 (hexadecimal 10) is added to the Unit number.

unit number

An identification number for each Unit connected to a CPU bus.

word

A unit of data. There are two bytes of data in one word, eight bits in each byte, for a total of 16 bits per word.

Appendix B
I/O Configuration

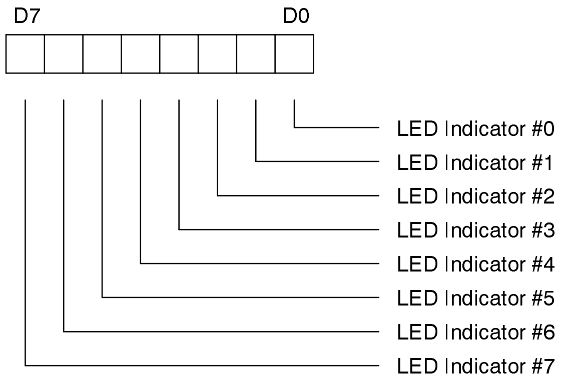
I/O Configuration

The I/O configuration for the CV500-VP2□□-E Personal Computer Units is shown in the following table.

I/O addresses	Function
000 to 01F	DMA controller, channels 0 to 3
020 to 021	Interrupt controller #1 (Master)
040 to 043	System timer
060 to 06F	Keyboard
070 to 071	RT/CMOS NMI controller
081 to 09F	DMA page register
0A0 to 0A1	Interrupt controller #1 (Slave)
0C0 to 0DF	DMA controller, channels 4 to 7
0F0 to 0FF	Math coprocessor
2F8 to 2FF	Serial port #2
300 to 30F	Hard Disk Interface Board (Can be changed to 340 to 34F.)
378 to 37A	Parallel port #1
390 to 394	CPU bus, LEDs, DIP switch
3C0 to 3DF	Video subsystem
3E0 to 3EF	CV500-MP602 (PC Card Interface Board)
3F0 to 3F8	Disk controller
3F8 to 3FF	Serial port #1

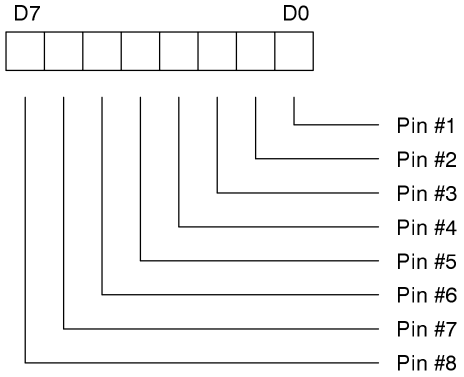
Port 390 (Read/Write)

These bits turn the user indicators on and off. Turn a bit off (0) to turn on the corresponding LED indicator. Turn a bit on (1) to turn off the corresponding LED indicator.



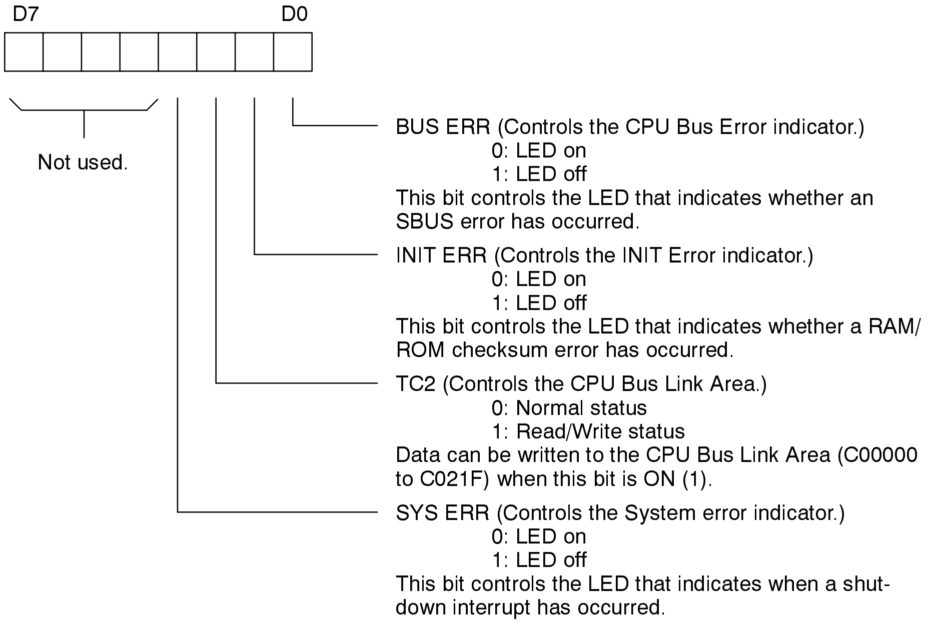
Port 391 (Read)

These flags reflect the status of the pins on the DIP switch. A flag is off (0) when its corresponding pin is ON, and on (1) when its corresponding pin is OFF.



Port 392 (Read/Write)

Usually these bits aren't changed by the user.



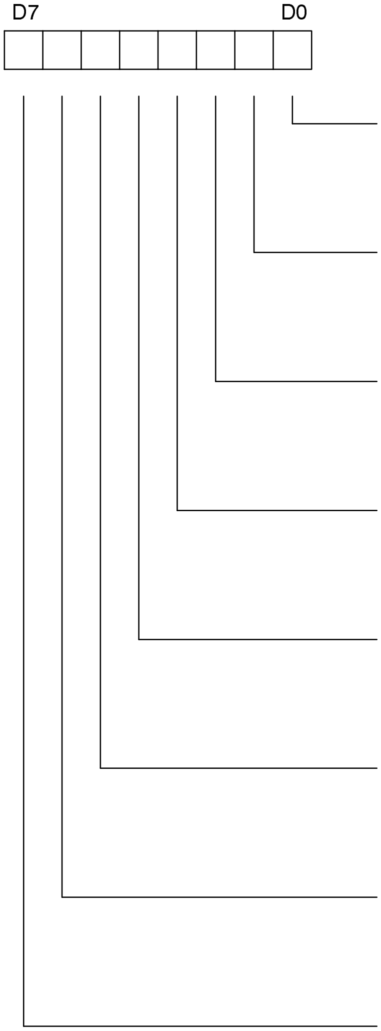
BUS ERR (Controls the CPU Bus Error indicator.)
0: LED on
1: LED off
This bit controls the LED that indicates whether an SBUS error has occurred.

INIT ERR (Controls the INIT Error indicator.)
0: LED on
1: LED off
This bit controls the LED that indicates whether a RAM/ROM checksum error has occurred.

TC2 (Controls the CPU Bus Link Area.)
0: Normal status
1: Read/Write status
Data can be written to the CPU Bus Link Area (C00000 to C021F) when this bit is ON (1).

SYS ERR (Controls the System error indicator.)
0: LED on
1: LED off
This bit controls the LED that indicates when a shutdown interrupt has occurred.

Port 394 (Read)



INT CPU Bus interrupt status
0: Interrupt occurred
1: Normal status
This flag indicates whether an interrupt has been generated from the CPU bus.

PF Power failure interrupt status
0: Power failure interrupt occurred
1: Normal status
This flag indicates whether a power failure interrupt has occurred. (Write to port 394 to clear this flag to 1.)

SHTDWN Shutdown interrupt status
0: Shutdown occurred
1: Normal status
This flag indicates whether a shutdown interrupt has occurred. (Write to port 394 to clear this flag.)

WDTU CPU WDT error status
0: Watchdog timer error occurred
1: Normal status
This flag indicates whether a WDT error has occurred in the CPU.

BL1 Battery low 1 (Lower)
0: Battery error occurred
1: Normal status
This flag indicates that the voltage has dropped in the battery used for the clock/System RAM.

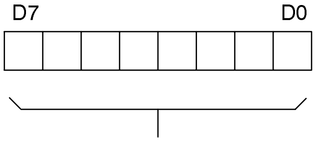
BL2 Battery low 2 (Middle)
0: Battery error occurred
1: Normal status
This flag indicates that the voltage has dropped in the battery used for the RAM Disk.

BL3 Battery low 3 (Upper)
0: Battery error occurred
1: Normal status
This flag indicates that the voltage has dropped in the battery used for the RAM Disk.

PFS Power failure status
0: Power failure occurred
1: Normal status
This flag indicates the line status of the power failure signal.

Port 394 (Write)

Write data to this register to clear the shutdown interrupt status flag (port 394, flag D2) and power failure interrupt status flag (port 394, flag D1). The data is dummy data.



Any data can be written.

Appendix C
Interrupts

Master

Interrupt header	IRQ	Function
08H	0	Timer interrupt
09H	1	Keyboard hardware interrupt
0AH	2	Slave controller’s connection
0BH	3	Serial port #2 interrupt
0CH	4	Serial port #1 interrupt
0DH	5	Open
0EH	6	Disk controller interrupt
0FH	7	Parallel port interrupt

Slave

Interrupt header	IRQ	Function
70H	8	Real time clock interrupt
71H	9	VGA interrupt
72H	10	PC’s power interruption interrupt
73H	11	CPU bus interrupt
74H	12	(PS2 mouse interrupt)
75H	13	Math coprocessor interrupt
76H	14	IDE hard disk interrupt
77H	15	Open (Used by CV500-MP602.)

Appendix D
Error Codes

The table below lists the error codes that might be read from the Personal Computer Unit's error log when event servicing is used and "Error Log Read" is performed.

Error type	Error code	Contents
$0302	$0001	Posted unit address does not match recognized unit address.
	$0002	Not used.
	$0003	Unit addresses duplicated.
	$0004	Improper unit address recognized.
	$0005	Hardware test unit recognized.
	$0006	Unit address could not be found in registered I/O table.
	$0007	Routing table error
	$0008	Routing table reading error
	$0009	Parity error: Cyclic service
	$000A	Parity error: Event service
	$000B	Parity error: CPU bus link service
	$000C	Access rights return error: Cyclic service
	$000D	Access rights return error: Event service
	$000E	Access rights return error: CPU bus link service
	$000F	CPU bus area memory access error
	$0010	Communications command discarded due to buffer overflow.
	$0011	Communications response discarded due to buffer overflow.
	$0012	Reception packet unit address error
	$0013	Programmable Controller routing process error (command)
	$0014	Programmable Controller routing process error (response)
	$0015	Reception packet size error
	$0016	Relay center routing process error
	$0017	Error in CPU bus of transmission destination unit.
	$0018	Transmission destination unit error
	$0019	Transmission destination unit could not be found.
	$001A	Voltage drop: Battery no. 1 (clock/system RAM backup battery)
	$001B	Voltage drop: Battery no. 2 (RAM Disk backup battery)
	$001C	Voltage drop: Battery no. 3 (RAM Disk backup battery)
	$001D	Improper interrupt occurred.
	$001E	Buffer full: Could not transmit response.
	$00FF	POWER FAIL signal was received.

Appendix E
FINS Commands to the CV500-VP2□□-E

The following table shows the commands to which responses are returned by the CPU Bus Driver with event servicing. The Personal Computer Unit will return a response automatically when it receives one of these FINS commands. Refer to *4-6 FINS Commands Serviced by Drivers* for more details on these commands.

Commands for which Response Processing is Executed by the Driver

MRC	SRC	Command contents
$05	$01	Read Controller Information
$07	$01	Read Time Information
$07	$02	Write Time Information
$08	$01	Loopback Test
$21	$02	Read Error Log
$21	$03	Clear Error Log

Appendix F
FINS Commands to CV-series PCs

The following table shows the FINS commands addressable to CV-series PCs. Refer to the*FINS Command Reference Manual* for more details.

Command code		PC mode				Name
		RUN	MONITOR	DEBUG	PROGRAM	
01	01	OK	OK	OK	OK	MEMORY AREA READ
	02	OK	OK	OK	OK	MEMORY AREA WRITE
	03	OK	OK	OK	OK	MEMORY AREA FILL
	04	OK	OK	OK	OK	MULTIPLE MEMORY AREA READ
	05	OK	OK	OK	OK	MEMORY AREA TRANSFER
02	01	OK	OK	OK	OK	PARAMETER AREA READ
	02	OK	OK	OK	OK	PARAMETER AREA WRITE
	03	OK	OK	OK	OK	PARAMETER AREA CLEAR
03	04	OK	OK	OK	OK	PROGRAM AREA PROTECT
	05	OK	OK	OK	OK	PROGRAM AREA PROTECT CLEAR
	06	Not usable	OK	OK	OK	PROGRAM AREA READ
	07	Not usable	Not usable	Not usable	OK	PROGRAM AREA WRITE
	08	OK	OK	OK	OK	PROGRAM AREA CLEAR
04	01	OK	OK	OK	OK	RUN
	02	OK	OK	OK	OK	STOP
05	01	OK	OK	OK	OK	CONTROLLER DATA READ
	02	OK	OK	OK	OK	CONNECTION DATA READ
06	01	OK	OK	OK	OK	CONTROLLER STATUS READ
	02	OK	OK	Not usable	Not usable	CYCLE TIME READ
07	01	OK	OK	OK	OK	CLOCK READ
	02	OK	OK	OK	OK	CLOCK WRITE
09	20	OK	OK	OK	OK	MESSAGE READ
						MESSAGE CLEAR
						FAL/FALS MESSAGE READ
0C	01	OK	OK	OK	OK	ACCESS RIGHT ACQUIRE
	02	OK	OK	OK	OK	ACCESS RIGHT FORCED ACQUIRE
	03	OK	OK	OK	OK	ACCESS RIGHT RELEASE
21	01	OK	OK	OK	OK	ERROR CLEAR
	02	OK	OK	OK	OK	ERROR LOG READ
	03	OK	OK	OK	OK	ERROR LOG CLEAR

Command code		PC mode				Name
		RUN	MONITOR	DEBUG	PROGRAM	
22	01	OK	OK	OK	OK	FILE NAME READ
	02	OK	OK	OK	OK	SINGLE FILE READ
	03	OK	OK	OK	OK	SINGLE FILE WRITE
	04	OK	OK	OK	OK	MEMORY CARD FORMAT
	05	OK	OK	OK	OK	FILE DELETE
	06	OK	OK	OK	OK	VOLUME LABEL CREATE/DELETE
	07	OK	OK	OK	OK	FILE COPY
	08	OK	OK	OK	OK	FILE NAME CHANGE
	09	OK	OK	OK	OK	FILE DATA CHECK
	0A	OK	OK	OK	OK	MEMORY AREA FILE TRANSFER
	0B	OK	OK	OK	OK	PARAMETER AREA FILE TRANSFER
	0C	(See note.)	OK	OK	OK	PROGRAM AREA FILE TRANSFER
23	01	Not usable	OK	OK	OK	FORCED SET/RESET
	02	Not usable	OK	OK	OK	FORCED SET/RESET CANCEL

Note When the PC is in RUN mode, data transfers from files to the program area aren’t possible.

Appendix G
Troubleshooting with FINS Response Codes

The following table lists the response codes (main and sub-codes) returned after execution of FINS commands, the probable cause of the errors, and recommended remedies.

Main code	Sub-code	Probable cause	Remedy
00: Normal completion	00	---	---
	01	Service was interrupted	Check the contents of the destination transmission area of third node.
01: Local node error	01	Local node not part of Network	Add to Network.
	02	Token time-out, node address too high	Set the local node's node address below the maximum node address
	03	Number of transmit retries exceeded	Check communications with internode echo test. If the test fails, check network.
	04	Maximum number of frames exceeded	Either check the execution of events in the network and reduce the number of events occurring in one cycle, or increase the maximum number of frames.
	05	Node address setting error (range)	Make sure the node address is within specified range and that there are no duplicate node addresses.
	06	Node address duplication error	Make sure that there are no duplicate node addresses.
02: Destination node error	01	Destination node not part of Network	Add to Network.
	02	No node with the specified node address	Check the destination node's node address.
	03	Third node not part of Network	Check the third node's node address.
	04	Busy error, destination node busy	Increase the number of transmit retry attempts or re-evaluate the system so that the destination node is not so busy receiving data.
	05	Response time-out, message packet was corrupted by noise	Increase the number of transmit retry attempts. Perform an internode echo test to check noise level.
		Response time-out, response watchdog timer interval too short	Increase the value for the response watchdog timer interval.
03: Communications controller error	01	Error occurred in the communications controller, ERC indicator is lit	Take corrective action, referring to communications controller errors and remedies table at end of this section
	02	CPU error occurred in the PC at the destination node	Clear the error in the CPU (refer to the PC's operation manuals)
	03	Controller error: A response was not received because an error occurred on the controller board. Confirm the error on the indicators on the board.	Check the communications status on the network and restart the board. If the error repeats, replace the board.
	04	Node address setting error	Make sure the node address is within specified range and that there are no duplicate node addresses.
04: Not executable	01	An undefined command has been used.	Check the command code.
	02	Cannot process command because the specified unit model or version is wrong.	Check the unit model and version.

Main code	Sub-code	Probable cause	Remedy
05: Routing error	01	Destination node address is not set in the routing table.	Set the destination node address in the routing table.
	02	Routing table isn't registered.	Set the source nodes, destination nodes, and relay nodes in the routing table.
	03	Routing table error	Set the routing table correctly.
	04	The maximum number of relay nodes (2) was exceeded in the command.	Redesign the network or reconsider the routing table to reduce the number of relay nodes in the command.
10: Command format error	01	The command is longer than the max. permissible length.	Check the command format of the command and set it correctly.
	02	The command is shorter than min. permissible length.	Check the command format of the command and set it correctly.
	03	The designated number of data items differs from the actual number.	Check the number of items and the data, and make sure that they agree.
	04	An incorrect command format has been used.	Check the command format of the command and set it correctly.
	05	An incorrect header has been used. (The local node's relay table or relay node's local network table is wrong.)	Set the routing table correctly.
11: Parameter error	01	A correct memory area code has not been used or Expansion Data Memory is not available.	Check the command's memory area code and set the appropriate code.
	02	The access size specified in the command is wrong, or the first address is an odd number.	Set the correct access size for the command.
	03	The first address is in an inaccessible area.	Set a first address that is in an accessible area.
	04	The end of specified word range exceeds the acceptable range.	Check the acceptable limits of the data area and set the word range within the limits.
	06	A non-existent program no. has been specified.	Check the program number and be sure that it is set correctly.
	09	The sizes of data items in the command block are wrong.	Check the command data and be sure that the sixes of the data items are correct.
	0A	The IOM break function cannot be executed because it is already being executed.	Either abort the current IOM break function processing, or wait until it is completed and execute the command.
	0B	The response block is longer than the max. permissible length.	Check the command format and set the number of items correctly.
	0C	An incorrect parameter code has been specified.	Check the command data and reenter it correctly.
20: Read not possible	02	The program area is protected.	Execute the instruction again after issuing the PROGRAM AREA PROTECT CLEAR command.
	03	The registered table does not exist or is incorrect.	Set or reset the registered table.
	04	The corresponding data does not exist.	---
	05	A non-existing program no. has been specified.	Check the program number and be sure that it is set correctly.
	06	A non-existing file has been specified.	Check whether the correct file name was used.
	07	A verification error has occurred.	Check whether the memory contents are correct and replace if incorrect.

Main code	Sub-code	Probable cause	Remedy
21: Write not possible	01	The specified area is read-only or is write-protected.	If the specified area is read-only, the write cannot be performed. If it is write-protected, turn off the write-protect switch and execute the instruction again.
	02	The program area is protected.	Execute the instruction again after issuing the PROGRAM AREA PROTECT CLEAR command.
	03	The number of files exceeds the maximum permissible.	Write the file(s) again after erasing unneeded files, or use different disk or Memory Card that has free space.
	05	A non-existing program no. has been specified.	Check the program number and be sure that it is set correctly.
	06	A non-existent file has been specified.	---
	07	The specified file already exists.	Change the name of the file and execute the instruction again.
	08	Changing the memory contents is not possible because doing so would result in an inconsistency.	Check the memory content that you are attempting to change.
22: Not executable in current mode	01	The mode is wrong (executing).	Check the operating mode.
	02	The mode is wrong (stopped).	Check the operating mode.
	03	The PC is in the PROGRAM mode.	Check the PC's mode.
	04	The PC is in the DEBUG mode.	Check the PC's mode.
	05	The PC is in the MONITOR mode.	Check the PC's mode.
	06	The PC is in the RUN mode.	Check the PC's mode.
	07	The specified node is not the control node.	Check which node is the control node.
	08	The mode is wrong and the step cannot be executed.	Check whether the step has active status or not.
23: No Unit	01	A file device does not exist where specified.	The Memory Card or disk is not installed.
	02	The specified memory does not exist.	Check the specifications of the installed file memory.
	03	No clock exists.	Check the model number.
24: Start/stop not possible	01	The data link table either hasn't been created or is incorrect.	Set the data link table correctly.

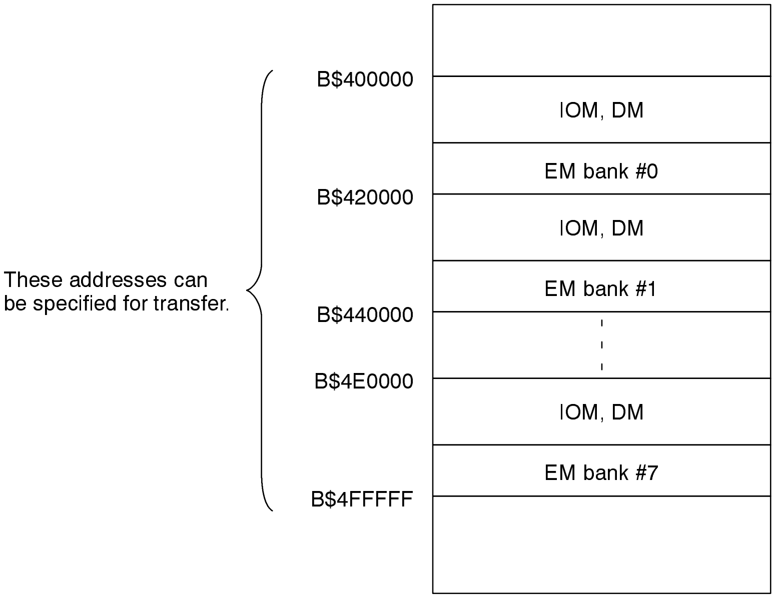
Main code	Sub-code	Probable cause	Remedy
25: Unit error	02	Parity/checksum error occurred because of incorrect data.	Transfer correct data into memory.
	03	I/O setting error (The registered I/O configuration differs from the actual.)	Either change the actual configuration to match the registered one, or generate the I/O table again.
	04	Too many I/O points	Redesign the system to remain within permissible limits.
	05	CPU bus error (An error occurred during data transfer between the CPU and a CPU Bus Unit.)	Check the unit and cable connections and issue the ERROR CLEAR command.
	06	I/O duplication error (A rack number, unit number, or I/O word allocation has been duplicated.)	Check the system's settings and eliminate any duplication.
	07	I/O bus error (An error occurred during data transfer between the CPU and an I/O Unit.)	Check the unit and cable connections and issue the ERROR CLEAR command.
	09	SYSMAC BUS/2 error (An error occurred during SYSMAC BUS/2 data transfer.)	Check the unit and cable connections and issue the ERROR CLEAR command.
	0A	Special I/O Unit error (An error occurred during CPU Bus Unit data transfer.)	Check the unit and cable connections and issue the ERROR CLEAR command.
	0D	Duplication in SYSMAC BUS word allocation.	Check and regenerate the I/O table.
	0F	A memory error has occurred in internal memory, in the Memory Card, or in Expansion DM during the error check.	If the error occurred in internal memory or the EM Unit, correct the data in the command an execute it again. If the error occurred in a Memory Card or EM used for file memory, the file data has been corrupted. Execute the MEMORY CARD FORMAT command. If the above remedies do not eliminate the error, replace the faulty memory.
	10	Terminator not connected in SYSMAC BUS System.	Connect the terminator correctly.

Main code	Sub-code	Probable cause	Remedy
26: Command error	01	The specified area is not protected. This response code will be returned if an attempt is made to clear protection on an area that is not protected.	The program area is not protected, so it isn’t necessary to clear protection.
	02	An incorrect password has been specified.	Specify a password that is registered.
	04	The specified area is protected.	Execute the command again after the PROGRAM AREA PROTECT CLEAR command.
	05	The service is being executed.	Execute the command again after the service has been completed or aborted.
	06	The service is not being executed.	Execute the service if necessary.
	07	Service cannot be executed from local node because the local node is not part of the data link.	Execute the service from a node that is part of the data link.
	08	Service cannot be executed because necessary settings haven’t been made.	Make the necessary settings.
	09	Service cannot be executed because necessary settings haven’t been made in the command data.	Check the command format of and make the necessary settings.
	0A	The specified action or transition number has already been registered.	Execute the command again using an action or transition number that hasn’t been registered.
	0B	Cannot clear error because the cause of the error still exists.	Eliminate the cause of the error and execute the ERROR CLEAR command.
30: Access right error	01	The access right is held by another device.	Execute the command again after the access right has been released. (The command can be executed after the ACCESS RIGHT FORCED ACQUIRE or ACCESS RIGHT RELEASE command is completed. Releasing the access right might affect processes in progress at the node that held the access right.)
40: Aborted	01	The ABORT command was executed.	---

Appendix H
PC Memory Configuration

The PC memory addresses which the user can specify with cyclic service are those in the IOM, DM, and EM Areas. These Areas are specified with the absolute PC memory addresses. (The UM Area cannot be specified.)

The following diagram shows the addresses that can be specified.



The PC's memory configuration is shown on the following page. This memory configuration is the same as the one listed in the *CV-series PC Operation Manual: Ladder Diagrams*, but the memory addresses are listed in word units. Memory addresses 0000 through FFFF on the next page correspond to addresses B$400000 through B$41FFFF in the diagram above.

PC Memory Configuration

Memory addresses	PC Memory	Data area addresses
0000 to 0FFF	IOM (I/O Memory) 4K-words	(See expanded view.)
1000 to 13FF	Timer PVs 1K-word	T0000 to T1023
1800 to 1BFF	Counter PVs 1K-word	C0000 to C1023
2000 to 27CF	Data link area 2K-words	D00000 to D01999
27D0 to 2E0F	CPU Bus Unit area 1600 words	D02000 to D03599
2E10 to 7FFF	24K-words of DM (CV500 up to D08191.)	D03600 to D24575
8000 to FFFD	EM banks 0 to 7 32K-words each (CV1000 only)	D03600 to D24575
FFFE to FFFF	Used by the system.	---

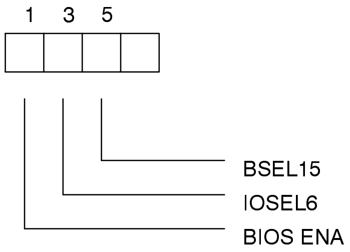
Memory addresses	PC Memory	Data area addresses
0000 to 00C7	I/O Area 200 words	CIO 0000 to CIO 0199
00C8 to 03E7	SYSMAC BUS/2 Area 800 words	CIO 0200 to CIO 0999
03E8 to 04AF	Link Area 200 words	CIO 1000 to CIO 1199
04B0 to 05DB	Holding Area 300 words	CIO 1200 to CIO 1499
05DC to 076B	CPU Bus Unit Area 400 words	CIO 1500 to CIO 1899
076C to 08FB	Work Area 400 words	CIO 1900 to CIO 2299
08FC to 09FB	SYSMAC BUS Area 256 words	CIO 2300 to CIO 2555
09FC to 09FE	Used by the system.	---
09FF	Temporary Relay Area	TR0 to TR7
0A00 to 0AFF	CPU Bus Link Area 256 words	G000 to G255
0B00 to 0CFF	Auxiliary Area 512 words	A000 to A511
0D00 to 0D3F	Transition Area 64 words	TN0000 to TN1023
0E00 to 0E3F	Step Area 64 words	ST0000 to ST1023
0F00 to 0F3F	Timer Area 64 words	T0000 to T1023
0F80 to 0FBF	Counter Area 64 words	C0000 to C1023

CIO Area

Note The CIO acronym is included for clarity; just input the address (0000 to 2555) when specifying words in the CIO Area.

Appendix I
Hard Disk Interface Board Settings

W2 Jumper Pin Settings



BIOS ENA	
Shorted	ROM valid
Open	ROM invalid

The following table shows the possible I/O address and BIOS address settings. (The addresses are all hexadecimal.)

Pin Settings		I/O Address	BIOS Address
IOSEL6	BSEL15		
Open	Open	340	D0000
Open	Shorted	340	D8000
Shorted	Open	300	D4000
Shorted	Shorted	300	DC000

Index

B

bar code reader, RS-232C response errors, 164

BASIC, 24
  compiler, 25
  interpreter, 25
  library functions, 24

BASIC functions
  PCCLOSE, 44
  PCMODE, 43
  PCMSRD, 26
  PCMSWR, 29
  PCOPEN, 26
  PCREAD, 31
  PCSTAT, 40
  PCWRITE, 35

BIOS, error codes, 177

Branching
  after FINS command reception, 93
  after FINS command response reception, 93
  after FINS command transmission, 92
  after FINS response transmission, 92

C

C, 45
  library functions, 45

C functions
  pcclose( ), 75
  pcmode( ), 73
  pcmsrd( ), 46
  pcmswr( ), 49
  pcopen( ), 46
  pcread( ), 51
  pcstat( ), 65
  pcwrite( ), 58

card reader, RS-232C response errors, 164

clear error log, 103

close, 126

commands, communications control, 157
  LCLOSE(), 158
  LDCCTL(), 161
  LDCRST(), 162
  LNCTL(), 160
  LNSTS(), 161
  LOPEN(), 157
  LREAD(), 158
  LRXCNT(), 162
  LTXCNT(), 163
  LWRITE(), 159

communications processing
  other devices, 8
    network address, 9
    node address, 9
    unit address, 9
  Personal Computer Unit, 8
  reception, 9
    buffers, 9

compiler, 25

control signals, LNCTL(), 160

copying, files, 16

CPU bus, measuring performance, 117

CPU Bus Driver, 78, 82
  access, 82
  closing, 83
  opening, 83
  operations, 85
    branching after FINS command reception, 93
    branching after FINS command response reception, 93
    branching after FINS command transmission, 92
    branching after FINS response transmission, 92
    cyclic area read, 86
    cyclic area write, 87
    cyclic service transmission address, 85
    cyclic service transmission direction, 85
    cyclic service transmission length, 85
    FINS command reception, 90
    FINS command transmission, 88
    FINS response reception, 91
    FINS response transmission, 89
    link area read, 95
    link area write, 96
    Personal Computer Unit resetting, 97
    reading information reserved for system, 94
    reception buffer flushing, 94
    reception status inquiry, 98
    timeout value settings, 94
    unit address inquiry, 97
    user timer service processing, 97
  operations list, 85
  requesting directly, 84
  requesting I/O, 83
  sample programs, 104

CPU bus driver, installation, 122

CPU Bus Interface, services, 2

CPU bus library
  BASIC, 24
    compiler, 25
    functions, 24
    interpreter, 25
  C, 45
    functions, 45
  disk configuration, 23
  programming procedure, 23

CPU Bus Link
  area, 3
  area contents, 11
  service, 2, 11, 78

CPU Bus Link service, processing time, 119

CPU Bus Unit, library, 20
- contents, 23
- functions, 23
- PCCLOSE, 22
- PCOPEN, 20
- PCREAD, 22
- PCWRITE, 21

CVSS, 182

cyclic area, 3

cyclic area read, 86

cyclic area write, 87

cyclic service, 2, 6, 78
- cyclic area, 6
- PC area, 6
- processing time, 118

cyclic service transmission address, 85

cyclic service transmission direction, 85

cyclic service transmission length, 85

D

DA1, 79

DA2, 79

DATA, 80, 81

data
- LREAD(), 158
- LWRITE(), 159

device drivers. *See* drivers

disks, configuration, System Disk 1, 15

DNA, 79

DNA to SA2, 80

drivers, 5
- installation, CPU bus driver, 78
- installing, 16

E

environment, precautions, 180

error codes, 191
- BIOS, 177

error logs, 174
- reading, error records, 174
- recording
  - area, 174
  - error records, 174
  - record format, 174

error records
- reading, 176
  - function calls, 176
  - procedure, 176
  - writing information, 177
- writing, 174
  - error log contents, 175
  - function calls, 175
  - procedure, 174

event area, 3

event service, 2, 7, 78
- processing time, 118
- service request command, 7
- service request response, 7

F

files
- copy, 16
- copying required files, 16

FINS, installing driver, 17

FINS command reception, 90

FINS command transmission, 88

FINS commands
- DA1, 79
- DA2, 79
- DATA, 80
- data format, 79
- DNA, 79
- GCNT, 79
- ICF, 79
- MRC, 80
- Personal Computer Unit processing, 193
- responses, 80
  - DATA, 81
  - DNA to SA2, 80
  - GCNT, 80
  - ICF, 80
  - MRC, 81
  - MRES, 81
  - RSV, 80
  - SID, 81
  - SRC, 81
  - SRES, 81
- RSV, 79
- SA1, 79
- SA2, 80
- sample programs, 132
- serviced by drivers, 98
  - clear error log, 103
  - loopback test, 101
  - read controller information, 99
  - read error log, 102
  - read time information, 100
  - write time information, 100
- SID, 80
- SNA, 79
- SRC, 80

FINS Driver, 138
  access, through MS-DOS, 141
  operations, 142
    initialize, 142
    receive, 145
    transmit, 143
  processing flow, 138
  sample programs, 147
  use, 141

FINS format, 78

FINS Library, 122
  design
    other than Quick BASIC, 125
    Quick BASIC, 124
  operations, 126
    close, 126
    initialize, 126
    receive with timer monitoring, 130
    receive without timer monitoring, 128
    transmit, 126
  processing flow, 123
  structure, 122
  use, 124

FINS response codes, 197

FINS response reception, 91

FINS response transmission, 89

functions, CPU bus library, 23

G

GCNT, 79, 80

H

Hard Disk Interface Board, settings, 205

I

I/O configuration, 185

ICF, 79, 80

initialize, 126, 142

installation
  CPU bus driver, 78, 122, 138, 181
  device drivers, 16
    options, 17

interpreter, 25

interrupts, 189

L

LCLOSE(), 158

LDCCTL(), 161

LDCRST(), 162

libraries, 5

library, serial communications, 156

link area read, 95

link area write, 96

LNCTL(), 160

LNSTS(), 161

loopback test, 101

LOPEN(), 157

LREAD(), 158

LRXCNT(), 162

LTXCNT(), 163

LWRITE(), 159

M

memory, configuration, 183, 203

MRC, 80, 81

MRES, 81

O

OCR, RS-232C response errors, 164

P

PCCLOSE, 22, 44

pcclose( ), 75

PCMODE, 43

pcmode( ), 73

PCMSRD, 26

pcmsrd( ), 46

PCMSWR, 29

pcmswr( ), 49

PCOPEN, 20, 26

pcopen( ), 46

PCREAD, 22, 31

pcread( ), 51

PCSTAT, 40

pcstat( ), 65

PCWRITE, 21, 35

pcwrite( ), 58

performance, CPU bus, 117

Personal Computer Unit, RS-232C response errors, 165

Precautions, 181

precautions, operating environment, 180

processing time, measuring, 118

program, design
  Assembler, 16
  C, 16
  Quick BASIC, 16

Programmable Terminals, RS-232C response errors, 164

programming languages, 5

programs, samples
  CPU Bus Driver, 104
  FINS commands, 132
  FINS Driver, 147

Q

Quick Library, changing version, 16

R

read controller information, 99

read error log, 102

read time information, 100

reading information reserved for system, 94

receive, 145
  with timer monitoring, 130
  without timer monitoring, 128

reception, LRXCNT(), 162

reception buffer flushing, 94

reception status inquiry, 98

resetting, Personal Computer Unit, 181

resetting the Personal Computer Unit, 97

response errors, RS-232C
  bar code reader, 164
  card reader, 164
  OCR, 164
  Personal Computer Unit, 165
  Programmable Terminals, 164
  sample program, 165, 167
  workstation, 165

response time, CPU bus service, 117

RS-232C communications
  LCLOSE(), 158
  LOPEN(), 157

RS-232C response errors, sample program, 165, 167

RSV, 79, 80

S

SA1, 79

SA2, 80

service request
  command, 7
  response, 7

services
  CPU Bus Link, 2, 78
  cyclic, 2, 78
  event, 2, 78
  ranges, 3
  selecting, 4

SID, 80, 81

SNA, 79

SRC, 80, 81

SRES, 81

SSS, 182

status, LNSTS(), 161

T

timeout value setting, 94

transmission, LTXCNT(), 163

transmit, 126, 143

U

unit address inquiry, 97

user indicators, 170
  checking, 171
  controlling, 170

user timer service processing, 97

W

workstation, RS-232C response errors, 165

write time information, 100

X

XON/XOFF
  LDCCTL(), 161
  LDCRST(), 162

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. W252-E1-1A

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	February 1995	Original production
2	October 1995	**Page 85:** Parameter format corrected. **Pages 104 to 106:** Sample program replaced. **Page 205:** *Appendix A Hard Disk Interface Board Settings* added.