生産終了
DISCONTINUED

C200H-ASC11/ASC21/ASC31 ASCII Units

Operation Manual

Revised June 2000

Read and Understand this Manual

Please read and understand this manual before using the product. Please consult your OMRON representative if you have any questions or comments.

Warranty and Limitations of Liability

WARRANTY

OMRON's exclusive warranty is that the products are free from defects in materials and workmanship for a period of one year (or other period if specified) from date of sale by OMRON.

OMRON MAKES NO WARRANTY OR REPRESENTATION, EXPRESS OR IMPLIED, REGARDING NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR PARTICULAR PURPOSE OF THE PRODUCTS. ANY BUYER OR USER ACKNOWLEDGES THAT THE BUYER OR USER ALONE HAS DETERMINED THAT THE PRODUCTS WILL SUITABLY MEET THE REQUIREMENTS OF THEIR INTENDED USE. OMRON DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED.

LIMITATIONS OF LIABILITY

OMRON SHALL NOT BE RESPONSIBLE FOR SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES, LOSS OF PROFITS OR COMMERCIAL LOSS IN ANY WAY CONNECTED WITH THE PRODUCTS, WHETHER SUCH CLAIM IS BASED ON CONTRACT, WARRANTY, NEGLIGENCE, OR STRICT LIABILITY.

In no event shall the responsibility of OMRON for any act exceed the individual price of the product on which liability is asserted.

IN NO EVENT SHALL OMRON BE RESPONSIBLE FOR WARRANTY, REPAIR, OR OTHER CLAIMS REGARDING THE PRODUCTS UNLESS OMRON'S ANALYSIS CONFIRMS THAT THE PRODUCTS WERE PROPERLY HANDLED, STORED, INSTALLED, AND MAINTAINED AND NOT SUBJECT TO CONTAMINATION, ABUSE, MISUSE, OR INAPPROPRIATE MODIFICATION OR REPAIR.

Application Considerations

SUITABILITY FOR USE

OMRON shall not be responsible for conformity with any standards, codes, or regulations that apply to the combination of products in the customer's application or use of the products.

At the customer's request, OMRON will provide applicable third party certification documents identifying ratings and limitations of use that apply to the products. This information by itself is not sufficient for a complete determination of the suitability of the products in combination with the end product, machine, system, or other application or use.

The following are some examples of applications for which particular attention must be given. This is not intended to be an exhaustive list of all possible uses of the products, nor is it intended to imply that the uses listed may be suitable for the products:

- Outdoor use, uses involving potential chemical contamination or electrical interference, or conditions or uses not described in this manual.
- Nuclear energy control systems, combustion systems, railroad systems, aviation systems, medical equipment, amusement machines, vehicles, safety equipment, and installations subject to separate industry or government regulations.
- Systems, machines, and equipment that could present a risk to life or property.

Please know and observe all prohibitions of use applicable to the products.

NEVER USE THE PRODUCTS FOR AN APPLICATION INVOLVING SERIOUS RISK TO LIFE OR PROPERTY WITHOUT ENSURING THAT THE SYSTEM AS A WHOLE HAS BEEN DESIGNED TO ADDRESS THE RISKS, AND THAT THE OMRON PRODUCTS ARE PROPERLY RATED AND INSTALLED FOR THE INTENDED USE WITHIN THE OVERALL EQUIPMENT OR SYSTEM.

PROGRAMMABLE PRODUCTS

OMRON shall not be responsible for the user's programming of a programmable product, or any consequence thereof.

Disclaimers

CHANGE IN SPECIFICATIONS

Product specifications and accessories may be changed at any time based on improvements and other reasons.

It is our practice to change model numbers when published ratings or features are changed, or when significant construction changes are made. However, some specifications of the products may be changed without any notice. When in doubt, special model numbers may be assigned to fix or establish key specifications for your application on your request. Please consult with your OMRON representative at any time to confirm actual specifications of purchased products.

DIMENSIONS AND WEIGHTS

Dimensions and weights are nominal and are not to be used for manufacturing purposes, even when tolerances are shown.

PERFORMANCE DATA

Performance data given in this manual is provided as a guide for the user in determining suitability and does not constitute a warranty. It may represent the result of OMRON's test conditions, and the users must correlate it to actual application requirements. Actual performance is subject to the OMRON Warranty and Limitations of Liability.

ERRORS AND OMISSIONS

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for clerical, typographical, or proofreading errors, or omissions.

Notice:

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

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

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

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

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

OMRON Product References

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

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

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

Visual Aids

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

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

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

© OMRON, 1998

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.

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

TABLE OF CONTENTS

PRECAUTIONS xi

1 Intended Audience xii
2 General Precautions xii
3 Safety Precautions xii
4 Operating Environment Precautions xii
5 Application Precautions xiii

SECTION 1
Introduction 1

1-1 Overview 2
1-2 System Configuration 3
1-3 Internal Configuration 5
1-4 Specifications 7
1-5 Comparison with Previous ASCII Units 10
1-6 Basic Operating Procedures 12

SECTION 2
Installation 17

2-1 Nomenclature and Functions 18
2-2 Installation 22

SECTION 3
IR/CIO and DM Area Allocations 29

3-1 IR/CIO Area Allocations 30
3-2 DM Area Allocations 38

SECTION 4
Data Exchange with General-purpose External Devices 43

4-1 Overview 44
4-2 Opening a Communications Port 44
4-3 Sending Data 47
4-4 Receiving Data 51
4-5 Closing a Communications Port 53
4-6 Communications Errors 59

SECTION 5
Processing 61

5-1 Processing Character Strings 62
5-2 Processing Bits 62
5-3 Processing Receive Buffers 62
5-4 Time Processing 63
5-5 Interrupt Functions 64
5-6 Loop Processing 67

SECTION 6
Data Exchange with the CPU Unit 69

6-1 Overview of Data Exchanges 70

TABLE OF CONTENTS

6-2 Selecting the Data Exchange Method 74
6-3 Details of the Data Exchange Methods 75
6-4 Data Exchange Time Charts 87
6-5 IOWR/IORD Instruction Specifications 95
6-6 Restrictions for CS1-series PCs 108

SECTION 7
Editing BASIC Programs 111

7-1 Programming Procedure 112
7-2 Character Variable Space Allocations 118
7-3 Starting/Stopping the BASIC Program 120
7-4 Program Configuration 121
7-5 List of BASIC Commands 129
7-6 BASIC Commands 139
7-7 User-defined BASIC Functions 231
7-8 Debugging 232

SECTION 8
Data Exchange Application Programs 235

8-1 Asynchronous Processing 236
8-2 High-speed Data Exchanges 237
8-3 High-volume Data Exchanges 239
8-4 Data Transfer: ASCII Unit to CPU Unit 239
8-5 Bit Data Exchanges 240

SECTION 9
Clearing Error Messages 241

9-1 List of Error Messages 242
9-2 Troubleshooting 248
9-3 CPU Unit Error Indicators 252
9-4 Reading and Clearing the Error Log 254
9-5 Maintenance 255

Appendices

A Operating Precautions 257
B Comparison with C200H-ASC02 263
C PC Format 265
D Formats for Storing Variables in Memory 275
E Command Execution Time Samples 279
F Sample Programs 283
G Wiring RS-232C or RS-422A/485 Cable Connectors 287
H Reserved Words 289

Index 291

Revision History 295

About this Manual:

This manual describes the installation and operation of the C200H-ASC11/ASC21/ASC31 ASCII Units and includes the sections described below. These ASCII Units can be used with the C200HX/HG/HE, ZE-version C200HX/HG/HE, C200H, C200HS, and CS1-series PCs as a Special I/O Unit. Unless specified otherwise, references to the "C200HX/HG/HE" include the ZE-version C200HX/HG/HE PCs.

It has been assumed in the writing of this manual that the reader is already familiar with the hardware, programming, and terminology of OMRON PCs. If a review of this information is necessary, the reader should refer to the appropriate OMRON PC manuals for assistance.

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

Section 1 provides basic information on the features of the ASCII Unit, internal configurations, and operating procedures. It also provides the Unit's specifications, and a comparison of the ASCII Unit models covered in this manual with the previous ASCII Unit.

Section 2 describes how to install and connect the ASCII Unit, including mounting to the PC Backplane and connecting to external devices/computers.

Section 3 describes the methods used to allocate words to the ASCII Unit in the IR and DM Areas and the applications of the bits and words in these areas.

Section 4 describes the ASCII Unit commands in conjunction with transmission control signals for opening and closing communications ports and for sending and receiving data between the ASCII Unit and external devices.

Section 5 provides examples of processing character strings, bits, receive buffers, looping, and interrupts.

Section 6 describes procedures for exchanging data between the ASCII Unit and the CPU Unit, the transfer sequence, instruction/command timing, and the use of various interrupts.

Section 7 describes the procedures for editing, saving, starting, and stopping BASIC programs. An overview of BASIC program configuration, language definitions, and the use of various BASIC commands, statements, and functions is provided as well.

Section 8 provides examples of data exchange applications.

Section 9 provides a list of error messages, probable causes and possible corrections.

The ***Appendices*** provide important operating precautions, differences with the command language used for the ASC02 ASCII Unit, the PC format, and the formats for storing variables in memory.

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

PRECAUTIONS

This section provides general precautions for using the Programmable Controller (PC) and related devices.

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

1 Intended Audience xii
2 General Precautions xii
3 Safety Precautions. xii
4 Operating Environment Precautions xii
5 Application Precautions xiii

1 Intended Audience

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

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

2 General Precautions

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

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

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

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

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

3 Safety Precautions

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

⚠Caution Do not turn OFF the power supply to the PC while the BASIC program is being written to the flash ROM of the ASCII Unit by executing ROMSAVE command from the terminal.

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

4 Operating Environment Precautions

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

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidity outside the range specified in the specifications.

- Locations subject to condensation as the result of severe changes in temperature.
- Locations subject to corrosive or flammable gases.
- Locations subject to dust (especially iron dust) or salts.
- Locations subject to exposure to water, oil, or chemicals.
- Locations subject to shock or vibration.

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

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

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

5 Application Precautions

Observe the following precautions when using the ASCII Unit or PC System.

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

- Always turn OFF the power supply to the PC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting I/O Units, CPU Units, Memory Cassettes, or any other Units.
  - Assembling the Units.
  - Setting DIP switches or rotary switches.
  - Connecting or wiring the cables.
  - Connecting or disconnecting the connectors.

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

- There are restrictions in the data areas and addresses that can be specified for the PC READ, PC WRITE, PC READ@, and PC WRITE@ commands in the BASIC program when using an ASCII Unit with a CS1-series CPU Unit. Be particularly careful when using existing BASIC programs in ASCII Units for CS1-series PCs.

- Fail-safe measures must be taken by the customer to ensure safety in the event of incorrect, missing, or abnormal signals caused by broken signal lines, momentary power interruptions, or other causes.
- Install the Unit properly as specified in the operation manual. Improper installation of the Unit may result in malfunction.
- Be sure that all the mounting screws, terminal screws, and cable connector screws are tightened to the torque specified in the relevant manuals. Incorrect tightening torque may result in malfunction.
- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Double-check all the wiring before turning ON the power supply. Incorrect wiring may result in burning.
- Mount the Unit only after checking the connectors and wring.
- Be sure that the terminal blocks, Memory Units, expansion cables, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.
  - Changing the operating mode of the PC.
  - Force-setting/force-resetting any bit in memory.
  - Changing the present value of any word or any set value in memory.
- Before touching the Unit, be sure to first touch a grounded metallic object in order to discharge any static built-up. Not doing so may result in malfunction or damage.

SECTION 1
Introduction

This section provides basic information on the features of the ASCII Unit, internal configurations, and operating procedures. It also provides the Unit's specifications, and a comparison of the ASCII Unit models covered in this manual with the previous ASCII Unit.

1-1 Overview 2
1-2 System Configuration 3
1-3 Internal Configuration 5
1-4 Specifications 7
1-5 Comparison with Previous ASCII Units 10
1-6 Basic Operating Procedures 12
1-6-1 Editing BASIC Programs 12
1-6-2 BASIC Programming Procedure 14
1-6-3 Connection and Transfer to External Devices 14

1-1 Overview

The ASCII Unit is a Special I/O Unit that can exchange ASCII data with personal computers, board computers, bar code readers, ID systems, and non-OMRON Programmable Controllers (PCs) via an RS-232C or RS-422A/485 interface. Because it is programmed in BASIC, the ASCII Unit can independently process read/write data from general-purpose peripheral devices, making it easy to combine sequence control and information processing on the same PC.

ASCII Communications with Peripheral Devices

The following functions are provided. These functions are not possible with protocol macros and host link non-procedural communications alone.

- Flexible processing of character strings (sampling, comparison, combination, and length determination of portions of character strings).
- Reception processing for only a portion of the data from a receive buffer.
- Processing of interrupt subroutines, such as those for communications, time processing, and PC interrupts.
- Flexible response processing, such as retries and time monitoring.

These functions enable the ASCII Unit to handle all kinds of communications frames and protocols. It can handle essentially all data exchanges with other devices equipped with an RS-232C or RS-422A/485 port.

Multiport Communications

Up to either 10 or 16 ASCII Units can be mounted on each PC (depending on the model of the CPU Unit), making ASCII Units ideal for monitoring a large number of general-purpose peripheral devices via communications.

Special Processing

Apart from communicating with general-purpose peripheral devices, ASCII Units can also be used for special operations. Data operations, character string processing, logic flow processing, and other operations that are difficult to carry out in the PC can be handled by using the ASCII Units as “sub-CPUs” for the CPU Unit. This aids high-speed processing in the system as a whole, and reduces the load on the CPU Unit. In particular, by carrying out special operations and transmitting back the results when an interrupt is received from the CPU Unit, ASCII Units can act as co-processors for the CPU Unit.

C200H-ASC11/ASC21/ASC31 Features

High-capacity User Memory

ASCII Units each provide 200 Kbytes of user memory (BASIC program area and variable area), enabling them to handle complicated applications.

High-speed Processing

A 32-bit MPU (C200H-ASC02 used an 8-bit processor) enables high-speed processing.

RS-422A/485 Port

The RS-422A/485 port enables long-range communications and 1:N communications.

Various Formats for CPU Unit Data Transfers

In addition to previous commands (PC READ/WRITE, PC READ@/WRITE@, and PC PUT/PC GET) it is also possible to execute PC QREAD/QWRITE triggered by the ASCII Unit, as well as the PC EPUT/EGET commands, which allow the ASCII Unit to read/write the memory shared between the ASCII Unit and the CPU Unit. For the C200HX/HG/HE or CS1-series PCs, the CPU Unit can immediately read/write the ASCII Unit using the IOWR/IORD instructions. This means that the user can select the data exchange method to match the application, including the read/write trigger, the Unit executing the read/write, the read/write timing, the transmitted data size, and whether or not special programming is required.

High-speed Data Exchange Using Shared Memory

There is an internal shared memory shared with the CPU Unit, and the ASCII Unit and the CPU Unit can access this shared memory asynchronously. Previously, in data exchange using the I/O refresh period, the CPU Unit or ASCII Unit had to wait until the I/O refresh timing arrived, but now shared memory can be read/written as soon as the command/instruction is executed. High-speed data transfer is also possible by combining an instantaneous interrupt with the IOWR instruction addressed to the ASCII Unit from the CPU Unit.

Bit ON/OFF Command

By treating variables as bit strings, individual bits out of 16 or 32 bits can be turned ON/OFF in integer variables, simplifying bit input processing to the PC.

Extensive Interrupt Processing

There are two ways that the CPU Unit can interrupt the ASCII Unit: by I/O refreshing or by IOWR instruction. There are 5 types of interrupts for receiving data from the communications circuit: General data reception, specified character reception, specified byte size reception, leading and final character reception, leading and specified number of characters reception. Interrupts can also occur for key inputs, when an error occurs, and at specified times. Timers can also generate either one-shot or interval interrupts.

Built-in Transmission Control Signal Control

Commands for transmission control signals (RTS, DTR) and commands to monitor transmission control signals (CTS, DSR) are provided. These commands simplify handling all types of transmission protocols.

Error Check Calculation Commands

The FCS instruction can be used to calculate error check codes, including those for LRC (longitudinal parity), CRC-CCITT (XMODEM), CRC-16 (MODBUS), and SUM (1-byte or 2-byte). This allows you to eliminate the programming previously required to calculate error check codes.

Special Operations

Calculations are possible for exponential function (e^x), square root ($\sqrt{}$), trigonometric functions (TAN, SIN, and COS), inverse trigonometric functions (TAN^{-1}, SIN^{-1}, COS^{-1}), and logarithmic functions (LOG) in both single and double precision.

More Commands

Support is provided for conditional repeating commands (WHILE/WEND), IF commands enabling structured programming (@IF...ENDIF), and commands that record the interrupted line number when an interrupt occurred (INTRB).

Full BASIC Debugging

The following commands are also supported: BRKPT commands to suspend operation at specified lines, single or multiple line STEP commands, and WATCH commands to display the contents of variables when operation is suspended. This makes it easy to confirm that the BASIC program is running correctly.

A TRON command to monitor tracing of the execution lines is also available. The results of this trace are saved in the trace buffer, and can be monitored afterwards by means of the TRACE command. This allows checking the operation of the BASIC program.

Error Log

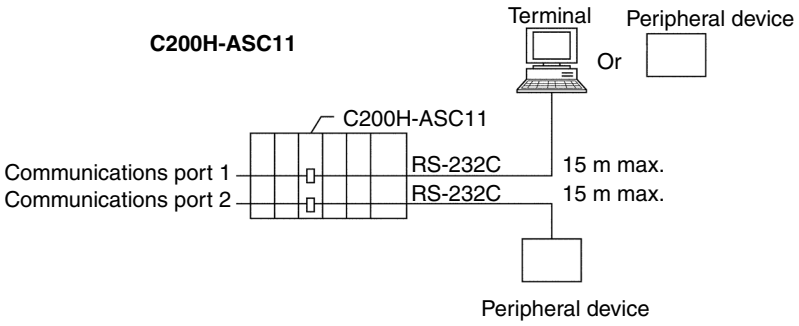
Up to 30 error records are stored in an error log. This is useful for troubleshooting.

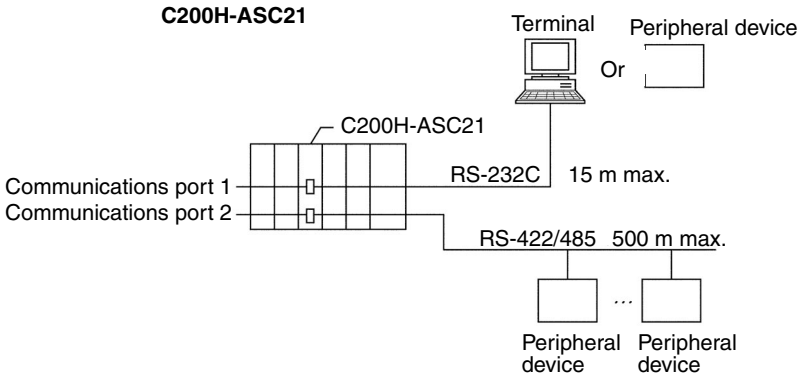
1-2 System Configuration

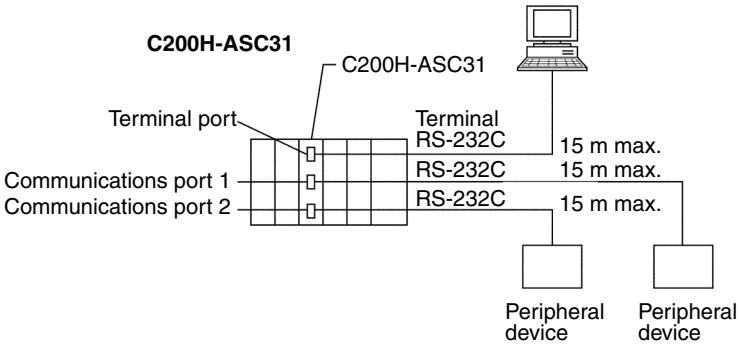
The C200H-ASC11/ASC21/ASC31 ASCII Units are Special I/O Units for the C200H, C200HS, C200HX/HG/HE, and CS1-series Programmable Controllers.

The ASCII Unit's communications port #1 (C200H-ASC11/ASC21) or terminal port (C200H-ASC31) can be connected to a computer terminal using an RS-232C cable to input BASIC commands, compile BASIC programs, debug, and display RUN results.

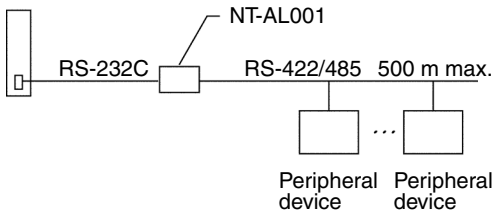
The ASCII Unit's communications port can be connected to a peripheral device using an RS-232C cable or twisted-pair cable (RS-422A/485) to send/receive communications frames and other data via the BASIC program.







Note Use an NT-AL001 Link Adapter (RS-232C to RS-422/485) to connect more than one terminal/device to an RS-232C port (except for the terminal port, port 3, on C200H-ASC31).



Connectable CPU Units

PC	CPU Unit	Total no. of Units on CPU Rack, Expansion I/O Racks, and SYSMAC BUS Slave Racks	Mounting restrictions
C200HX/HG/HE	C200HE-CPU11-E/32-E/42-E/11-ZE/32-ZE/42-ZE C200HG-CPU33-E/43-E/33-ZE/43-ZE C200HX-CPU34-E/44-E/34-ZE/44-ZE	10	None
	C200HG-CPU53-E/63-E/53-ZE/63-ZE C200HX-CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE	16	
C200HS	C200HS-CPU01-E/01-EC/21-E/21-EC/31-E/03-E/23-E/33-E	10	
CS1	CS1H-CPU66-E/65-E/64-E/63-E CS1G-CPU45-E/44-E/43-E/42-E	16	None
C200H	C200H-CPU01-E/02-E/03-E/11-E/21-E/23-E/31-E	10	Do not mount Unit to rightmost two slots on CPU Rack.

Restrictions on SYSMAC BUS Slave Racks

The following restrictions apply when mounting Units to SYSMAC BUS Slave Racks.

Group	A	B	C	D
Unit	ASCII Unit, High-speed Counter Unit, Position Control Unit (NC111/112/113/213) Analog I/O Unit, ID Sensor Unit, Fuzzy Logic Unit	Multi-point I/O Unit	Temperature Sensor Unit, Voice Unit	Position Control Unit (NC211) Motion Control Unit
Number of Units mountable for each group per Remote I/O Master Unit	4 Units	8 Units	6 Units	2 Units
Number of Units mountable for all groups combined per Remote I/O Master Unit	3A + B + 2C + 6D ≤ 12, and A + B + C + D ≤ 8			

Note The ASCII Unit can also be mounted to an C200H Expansion Rack connected to a Mini H-type PC (C20H/C28H/C40H/C60H).

1-3 Internal Configuration

ASCII Units are each allocated 10 words in the IR/CIO area of the PC. These allow the exchange of data read/write triggers, bit data, status, and interrupts with the CPU Unit during I/O refreshing without the need for special programming.

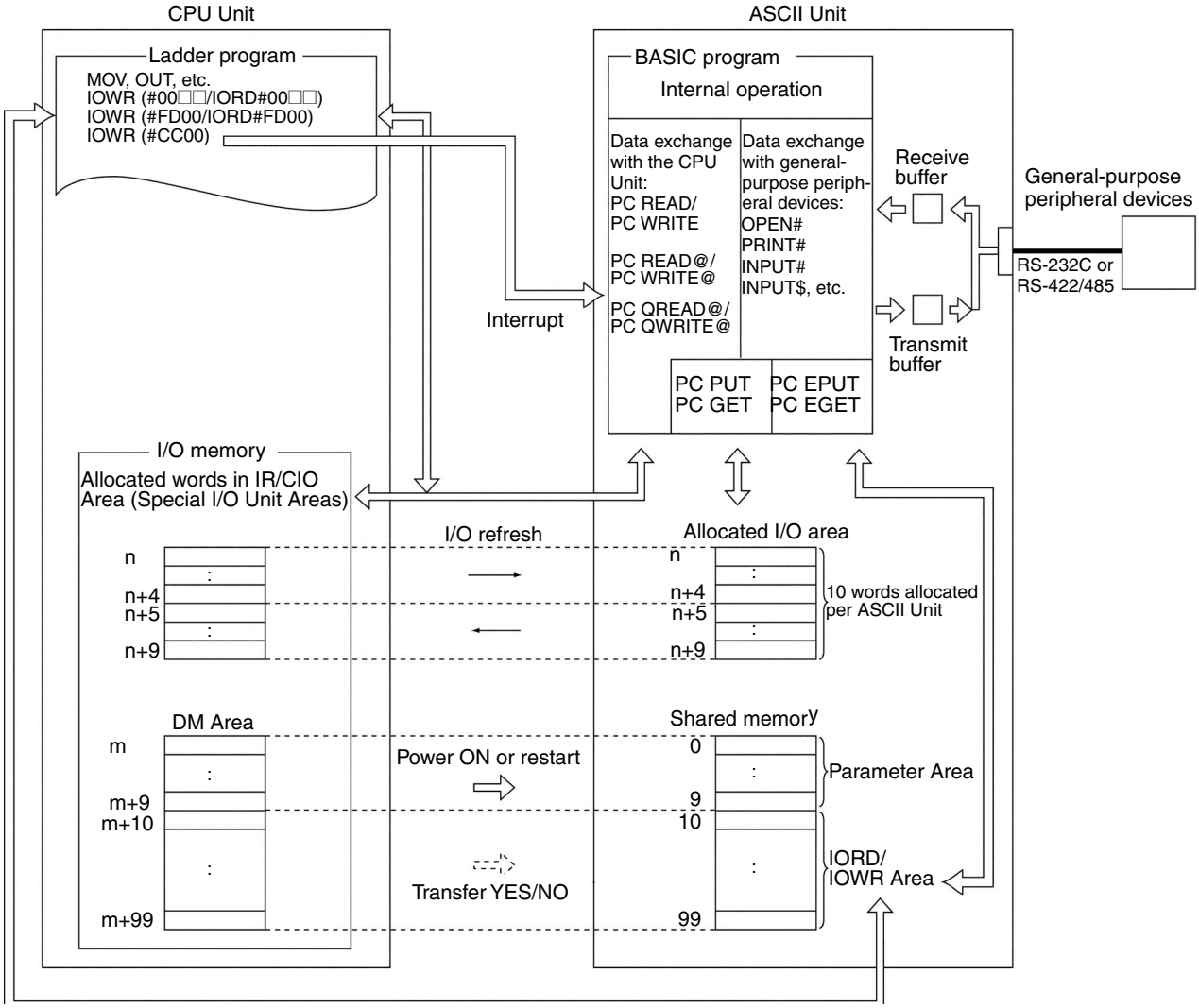
Each Unit is also allocated 10 words as a Setup Area in the data memory area (DM area). The Setup Area contains operating parameters, such as the communications parameters for communications ports and the starting program number. These parameters are automatically transferred from the DM area to the ASCII Unit when the power supply is turned on or the ASCII Unit is restarted.

Internally, the ASCII Unit provides 100 words of shared memory with the CPU Unit (10 Setup Area words and 90 general-purpose words). The ASCII Unit and the CPU Unit can access these areas independently (asynchronously).

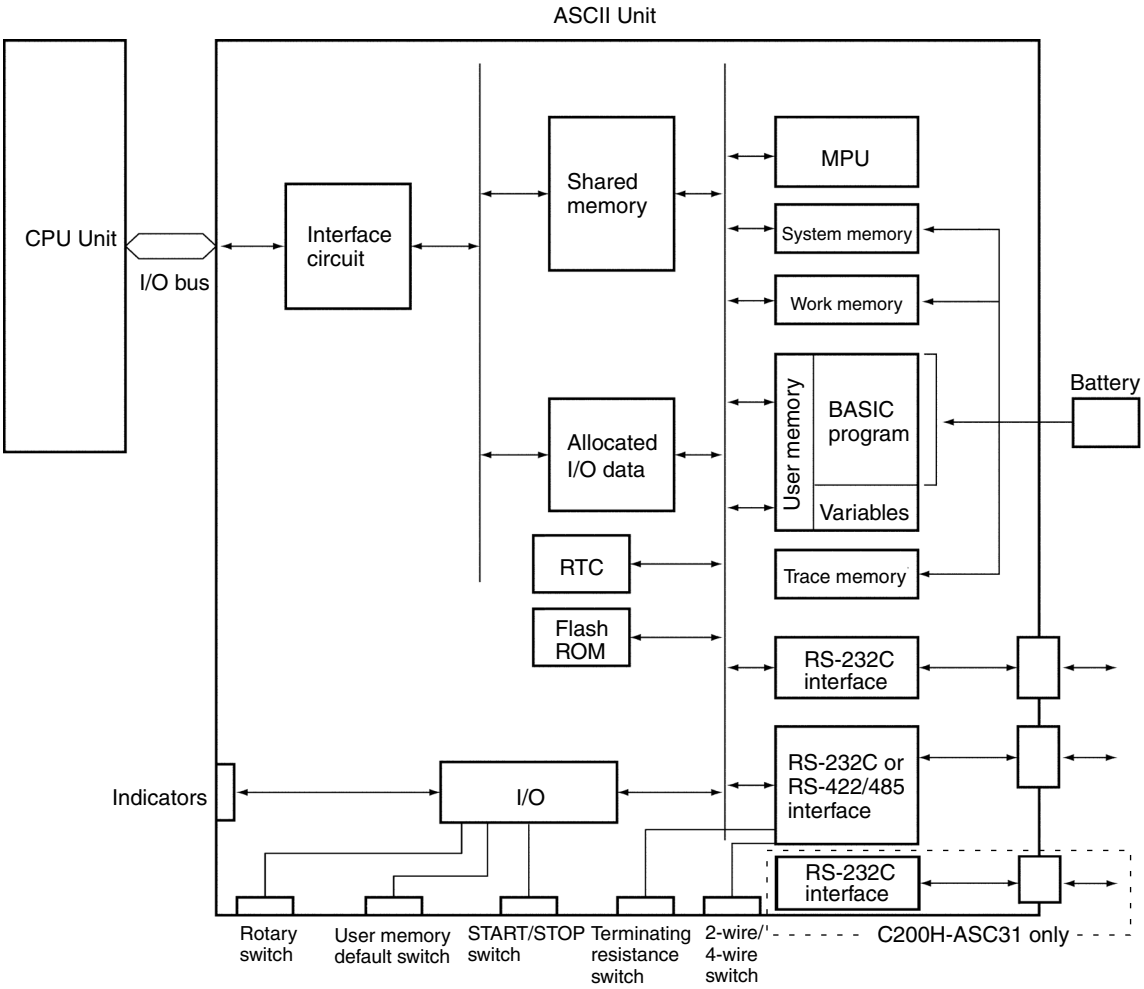
Data read/write that does not depend on the cycle time can be executed using the IOWR (SPECIAL I/O UNIT WRITE) or IORD (SPECIAL I/O UNIT READ) instructions from the CPU Unit (C200HX/HG/HE or CS1).

Interrupts that do not depend on the cycle time can be sent from the CPU Unit to the ASCII Unit using IOWR (SPECIAL I/O UNIT WRITE) instructions (C200HX/HG/HE or CS1).

Data Exchange



Internal Layout



Note 1. All memory, including RTC, is backed up by battery.
2. Work memory is part of user memory.
3. Trace memory is part of system memory.

1-4 Specifications

General Specifications Conform to those of the SYSMAC C200H, C200HS, C200HX/HG/HE (-Z), and CS1-series CPU Units.

Functional and Performance Specifications

Item		Specifications		
Model		**C200H-ASC11**	**C200H-ASC21**	**C200H-ASC31**
Communications port	**Port 1**	RS-232C (peripheral device or terminal connection)	RS-232C (peripheral device or terminal connection)	RS-232C (peripheral device connection)
	Port 2	RS-232C (peripheral device connection)	RS-422A/485 (peripheral device connection)	RS-232C (peripheral device connection)
	Terminal port	None	None	RS-232C (terminal connection)

Item		Specifications		
Model		**C200H-ASC11**	**C200H-ASC21**	**C200H-ASC31**
Communications parameters	**Communications mode**	Half duplex (full duplex available for RS-232C)		
	Synchronization	Start-stop		
	Baud rate	300/600/1,200/2,400/4,800/9,600/19,200/38,400 bps (terminal port does not support 38,400 bps) **Note** Using interrupts will limit the baud rates that can be used.		
	Transmission mode	RS-232C: Point-to-point (1:1) RS-422A/485: Point-to-multipoint (1:N)		
	Transmission distance	RS-232C: 15 m RS-422A/485: 500 m		
	Communications buffers	Receive buffer: 512-bytes Transmit buffer: 512-bytes		
	Response flow control	RTS/CTS flow (for receive buffer of external device) **Note** ASCII also asserts RTS when sending data. Xon, Xoff flow (for receive buffers of external device and ASCII Unit)		
Program	**Language**	BASIC		
	No. of tasks	Single task		
Memory capacity	**User program memory area (BASIC program + variable area)**	200 Kbytes Contents: BASIC source program (up to 4 programs can be loaded) intermediate code, variables. **Note** The BASIC program has battery backup. Make sure the battery is installed before operating.		
	Flash ROM	200 Kbytes The BASIC program can be stored in the flash ROM from the user memory area. It can be loaded into the user memory area when the power supply is turned on or the Unit is restarted.		
CPU Unit and data exchange areas	**Allocated I/O words**	Special I/O Unit Area: 10 words		
	Shared memory area	100 words (no battery backup) as follows: 10 word: Setup Area (can be transferred from the DM area words allocated in the CPU Unit to the ASCII Unit when the power supply is turned on or when the Unit is restarted). 90 wordGeneral-purpose area (IORD/IOWR area). **Note** Read/write is possible using IORD/IOWR instructions from the CPU Unit or PC_EPUT/PC_EGET commands from the ASCII Unit. The position and size of the IORD area/IOWR area can be adjusted. When the power is turned on or the Unit is restarted, the data in the DM words allocated in the CPU Unit can be transferred to the ASCII Unit. (The PC_EPUT and PC_EGET commands are not supported by the C200H and C200HS.)		

Item		Specifications		
Method of data exchange with CPU Unit		**Item**	**CPU Unit instructions**	**ASCII Unit commands**
		ASCII Unit reads/writes I/O memory on trigger from CPU Unit.	MOV, LD, etc.	PC READ/PC WRITE 255 words max. per command.
		ASCII Unit independently reads/ writes I/O memory.	Not required	PC READ@/PC WRITE@ 255 words max. per command.
		Read/write allocated IR/CIO area words.	MOV, LD etc.	PC PUT/PC GET 1 word per command for PC PUT; 2 words for PC GET.
		CPU Unit executes IOWR/IORD(#FD00) on trigger from ASCII Unit.	IOWR (#FD00) /IORD (#FD00) **Note** C200HX/HG/HE or CS1. 128 words max. per instruction.	PC QREAD/PC QWRITE 128 words max. per command.
		CPU Unit and ASCII Unit read/write to shared memory asynchronously.	IOWR (#00□□) /IORD (#00□□) **Note** C200HX/HG/HE or CS1. 90 words max. per instruction.	PC EPUT/PC EGET 90 words max. per command.
RTC		Year, month, day, date, hour, minute, second (leap year adjusted for) Accuracy: At 25°C: +70 s/–45 s per month At 0°C: +5 s/–110 s per month At 55°C: +0/–140 s per month		
		Battery backup. Can be synchronized with the CPU Unit time.		
Battery	**Model**	C200H-BAT09		
	Life	5 years at 25°C (The life of the battery is shortened if the ASCII Unit is used at higher temperatures.)		
Diagnostic functions		CPU watchdog timer, battery voltage drops		
Accessories		Connector plug: XM2A-0901 or equivalent (C200H-ASC11/21: 2, C200H-ASC31: 3) Hood: XM2S-0911 or equivalent (C200H-ASC11/21: 2, C200H-ASC31: 3)		
Internal current consumption (See note.)		250 mA max. at 5 VDC	300 mA max. at 5 VDC	300 mA max. at 5 VDC
Dimensions		130 x 35 x 100.5 mm (H x W x D)		
Weight		400 grams max.		

Note
1. If an NT-AL001 RS-232C/RS-422 Link Adapter is used at the RS-232C port, an additional 150 mA of current will be consumed.
2. Always connect the battery to the ASCII Unit before using it.

1-5 Comparison with Previous ASCII Units

The following table shows a comparison between the C200H-ASC11/ASC21/ASC31 and the C200H-ASC02 ASCII Units. Before using the ASCII Unit, refer to *Appendix A Operating Precautions*, and for further details, refer to *Appendix B Comparison with C200H-ASC02*.

Item	C200H-ASC02	C200H-ASC11 /ASC21/ASC31	Comments
Applicable PCs	C200H, C200HS, C200HX/HG/HE, CS1-series PCs		---
Unit No.	0 to 9	0 to 15	---
MPU	8 bits	32 bits	---
Interfaces (ports)	Two RS-232C	Two RS-232C OR One RS-232C and one RS-422A/485 OR One terminal port and two RS-232C	---
Baud rate	19,200 bps max.	38,400 bps max.	---
User program memory area (BASIC program + variable area)	24 Kbytes	200 Kbytes	---
BASIC program storage	EEPROM: 24 Kbytes	Flash ROM: 200 Kbytes	---
Number of BASIC programs in user area of Unit	Up to 3	Up to 4	---
Number of BASIC program lines	0 to 63,999	1 to 65,535	---
ASCII Unit system settings	DIP switch on back of Unit	Setup Area (10 words) (transferred from DM area at power ON or restart)	---
Allocated IR/CIO area words	4 words	10 words	---
Shared memory	None	Yes (general-purpose words: 90 words)	---

Item			C200H-ASC02	C200H-ASC11 /ASC21/ASC31	Comments
Commands/ Instructions	**ASCII Unit BASIC commands for data exchange with CPU Unit**	**PC READ/ PC WRITE**	Yes	Yes (multiple format strings supported)	On trigger from CPU Unit, ASCII Unit reads/writes I/O memory.
		PC READ@/ PC WRITE@	Yes	Yes (multiple format strings supported)	ASCII Unit reads/ writes I/O memory independently.
		PC QREAD/ PC QWRITE	None	Yes	On trigger from ASCII Unit, CPU Unit reads/ writes via IOWR/ IORD instructions.
		PC PUT/ PC GET	Yes	Yes	Reads/writes allocated IR/CIO area words.
		PC EPUT/ PC EGET	None	Yes	Reads/writes its own shared memory.
	Exchanging data with ASCII Unit from CPU Unit	**IOWR/IORD instructions (read/write to ASCII Unit's shared memory, IR/CIO area transfer, and interrupts)**	None	Yes (C200HX/HG/HE or CS1)	---
	Error check code calculation instruction		None	Yes (FCS)	---
	Bit ON/OFF command		None	Yes (BITON, BITOFF)	---
	Conditional repeat command		None	Yes (WHILE/WEND)	---
	Label branching		None	Yes	---
	Integer search		None	Yes (SEARCH)	---
	Debugging function	**Break point command**	None	Yes (BRKPT)	---
		Trace command	None	Yes (TRACE)	---
		Variable display command	None	Yes (WATCH)	---
	Transmission control code: Control command		None	Yes (RTS/DTR control, CTS/DSR monitor)	---
	Special operations	**Exponential function**	None	Yes	---
		Square root	None	Yes	---
	Floating point calculations		Single	Double	---
Machine language program			Yes	None	---

Item			C200H-ASC02	C200H-ASC11 /ASC21/ASC31	Comments
Inter-rupts	**PC interrupts**	**Using allocated IR/CIO area words**	Yes	Yes	---
		Using IOWR (CC##) instruction	None	Yes	---
		Maximum number	15	99	---
	Communica-tions interrupts (via received data)	**Interrupts when data is received in the receive buffer**	Yes	Yes	---
		Interrupt on reception of designated character	None	Yes	---
		Interrupt on reception of designated number of bytes	None	Yes	---
		Interrupt on reception of designated start and end characters	None	Yes	---
		Interrupt on reception of designated start character and desig-nated number of bytes	None	Yes	---
	Key interrupts		Yes	Yes	---
	Timed interrupts		None	Yes	---
	Interval timer interrupts		None	Yes	---
	One-shot timer interrupts		None	Yes	---
	Error interrupts		Yes	Yes	---
	Line code storage command on interrupt		None	Yes (INTRB, INTRR, INTRS)	---
Error log			None	Yes (Can record up to 30 error records. Time and date of occurrence not recorded.)	---
Internal current consumption			5 VDC 200 mA	5 VDC 250 mA or 300 mA	

1-6 Basic Operating Procedures

1-6-1 Editing BASIC Programs

The following example uses WINDOWS 95 on a personal computer as a terminal (hyperterminal VT-100 emulation).

1,2,3...

1. Mount the ASCII Unit to the Backplane. Refer to *2-2 Installation*.
2. Connect the Unit's terminal port to the personal computer's RS-232C port via an RS-232C cable. Refer to *2-2 Installation*.
3. Set the switches on the front of the Unit.
4. Turn on the PC power supply.
5. Perform the following steps if the ASCII Unit's default setup is used.
   a) Clear all the DM words allocated to the ASCII Unit in the CPU Unit using a PC Peripheral Device (such as a Programming Console).
   b) Toggle the PC power switch off and then back on, or restart the ASCII Unit.

6. Perform the following steps if the ASCII Unit's default setup is not being used.
   a) Set the DM words allocated to the ASCII Unit in the CPU Unit using a PC peripheral device (such as Programming Console).
   b) Select whether to set the terminals ports' communications parameters to the default settings or not, and whether to set the terminal emulation to VT100 or not. Refer to *3-2 DM Area Allocations*.
   c) Toggle the PC power switch off and then back on, or restart the ASCII Unit.
7. In WINDOWS 95, click **Start** and then point to **Programs**.
8. Point to **Accessories** and then click **Hyperterminal**. Refer to *7-1 Programming Procedure*.
9. Start the hyperterminal, and specify the connection settings and properties.
10. Press the Ctrl+X Keys on the computer keyboard.

Inputting a BASIC Program Directly into the ASCII Unit

1,2,3...

1. Input **New** to clear the ASCII Unit's program.
2. Input **Auto** to automatically display the line numbers.
3. Input the BASIC program and press the Enter Key.

Transferring a BASIC Program from a Text Editor to the ASCII Unit

1,2,3...

1. Input a program using a text editor such as Note Pad, and save it in text format.
2. Input **New** to clear the ASCII Unit's program.
3. Input the LOAD command.
4. Point to **Hyperterminal** and then **Transfer**, and click **Transmit text file** to transfer the saved file to the ASCII Unit. Specify a transmission delay after each line when transmitting data (or use XON/XOFF flow control).
5. Press the Ctrl+C Keys on the computer keyboard.

Transferring the ASCII Unit's BASIC program to the Personal Computer

1,2,3...

1. Using the SAVE command, read the program. Input SAVE #□, and press the Enter Key.
2. Point to **Hyperterminal** and then **Transfer**, and click **Capture Text.**
3. Input the file name to save the program and switch to reception mode.
4. Press the Ctrl+C Keys to begin the transfer.
5. Point to **Hyperterminal**, **Transfer**, and then **Capture Text**, and then click **STOP**.
6. Press the Ctrl+C Keys.

1-6-2 BASIC Programming Procedure

1,2,3...

1. Study communications procedures with the external devices. For further details on creating flowcharts for communications procedures, refer to *SECTION 4 Data Exchange with General-purpose External Devices*.
2. Study the method of data exchange with the CPU Unit, including the following:
   - Data exchange timing
   - Data exchange trigger to be used (i.e., whether to trigger the exchange from the CPU Unit or the ASCII Unit)
   - Interrupts from the CPU Unit
   - Data transfer method based on the volume of data

   For further details, refer to *SECTION 6 Data Exchange with the CPU Unit*.
3. Study BASIC internal processing. For further details, refer to *SECTION 5 Processing*.
4. Write the program. Refer to *7-4 Program Configuration*, *7-5 List of BASIC Commands*, and *7-6 BASIC Commands*.
5. Debug the program. Refer to *7-3 Starting/Stopping the BASIC Program* and *7-8 Debugging*.

Note

1. When storing the program in flash ROM, enter the ROMSAVE command from the terminal.
2. When automatically reading the program to user memory at power ON or when the Unit is restarted, set bits 08 to 15 of the DM word m to $5A_{hex}$ in the DM Setup Area allocated to the Unit.
3. Set the port number to 3 when using the LOAD or SAVE command for the C200H-ASC31.

1-6-3 Connection and Transfer to External Devices

1,2,3...

1. Mount the ASCII Unit to the Backplane.
2. Set the following switches on the front of the Unit.
   - Unit number switch
   - RS-422/485 2-wire/4-wire switch
   - RS-422/485 terminating resistance switch

   Refer to *2-1 Nomenclature and Functions*.
3. Connect the external devices using an RS-232C or RS-422/485 port depending on the type of device. Refer to *2-2 Installation.*
4. Turn on the PC power supply.
5. Using a PC peripheral device (such as Programming Console) set the DM area words allocated in the CPU Unit. In particular, set DM word m+1 bits 00 to 07 to specify whether the ASCII Unit will run when power is turned on or the Unit is restarted. Refer to *3-2 DM Area Allocations*.
6. Transfer the ladder program to the CPU Unit.
7. Toggle the PC power off and then back on, or restart the ASCII Unit. The BASIC program will be executed if the ASCII Unit is set to start operation at power ON or when the Unit is restarted (in the DM Setup Area allocated to the Unit).
8. If the ASCII Unit is set for manual starting at power ON or when the Unit is restarted (in the DM Setup Area allocated to the Unit), perform one of the following steps.

a) Set the START/STOP switch on the front panel of the Unit to START.

b) Input RUN from the terminal to run the BASIC program. (The START/STOP switch on the front panel of the Unit must be set to START first.)

Note

1. Use one of the following steps to pause the BASIC program.
   - Press the Ctrl+X Keys from the terminal.
   - Set the START/STOP switch on the front panel of the Unit to STOP.
2. Use one of the following steps to pause the BASIC program at a particular line.
   - Set a break point from the terminal.
   - Place a STOP command in the program.
   - Input the RUN TO line number command from the terminal.

SECTION 2
Installation

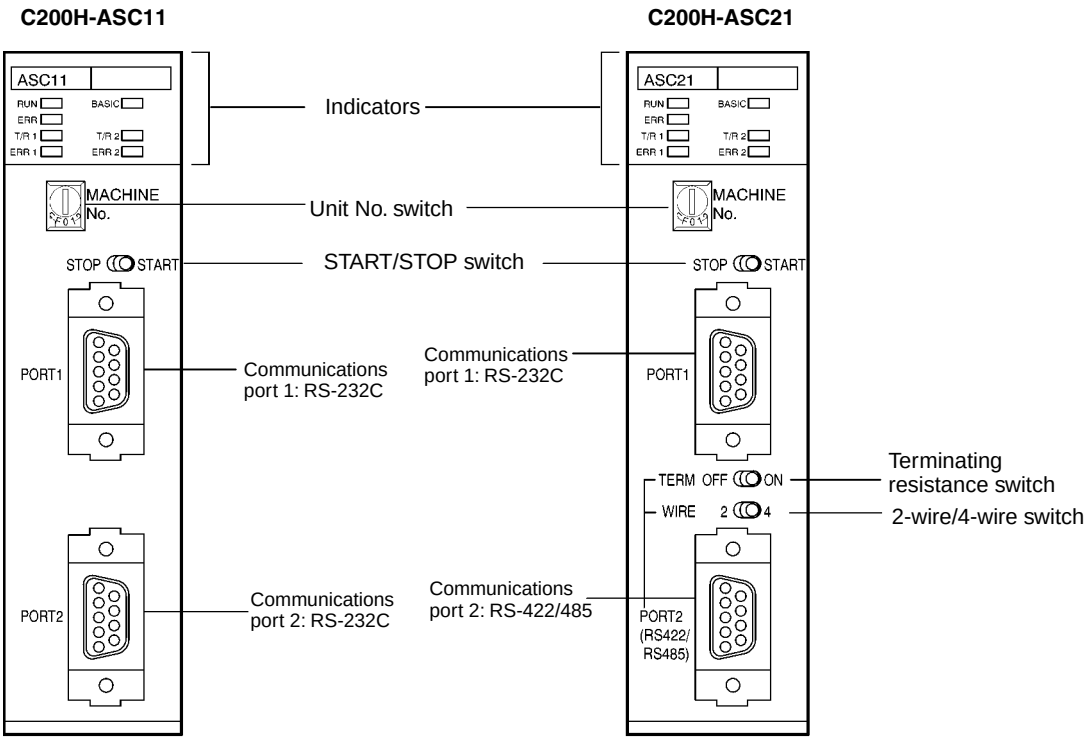
This section describes how to install and connect the ASCII Unit, including mounting to the PC Backplane and connecting to external devices/computers.

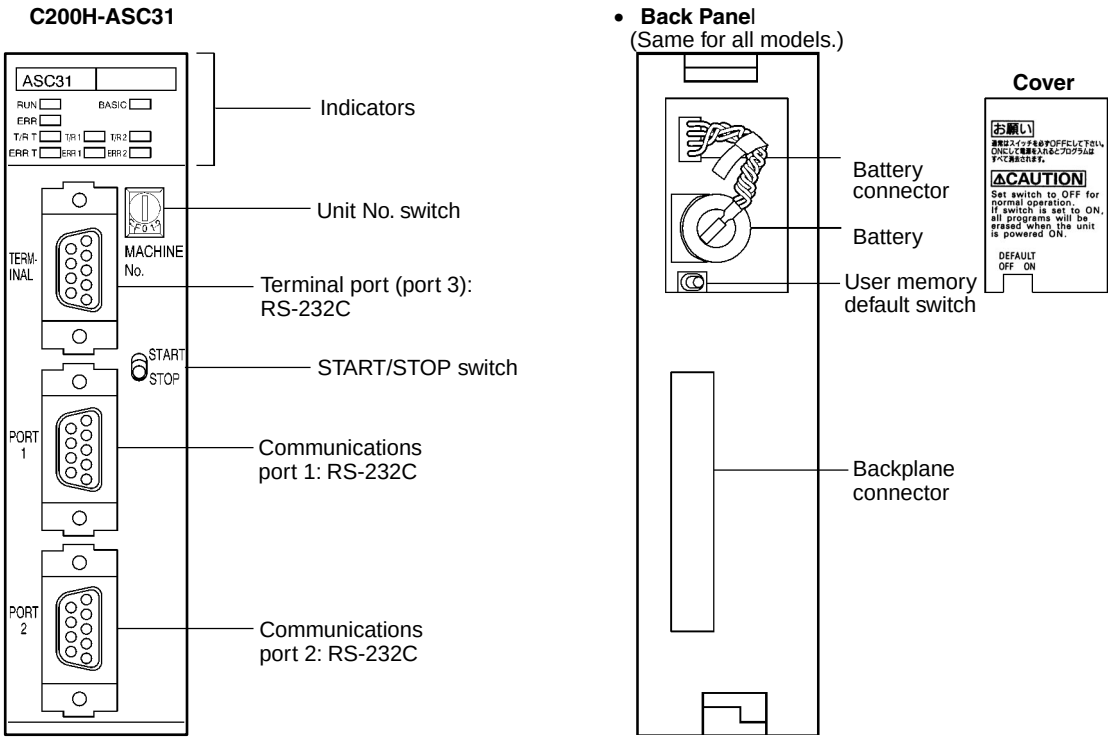
2-1 Nomenclature and Functions 18
2-1-1 Names of Parts 18
2-1-2 Front Panel 19
2-1-3 Rear Panel 21
2-2 Installation 22
2-2-1 Installation Method 22
2-2-2 Connections 24
2-2-3 General-purpose Peripheral Device Connections 26

2-1 Nomenclature and Functions

2-1-1 Names of Parts

- **Front Panel**





Note The above diagram shows the Unit with the cover removed. The caution on the cover will be added from Units shipped after November 1998.

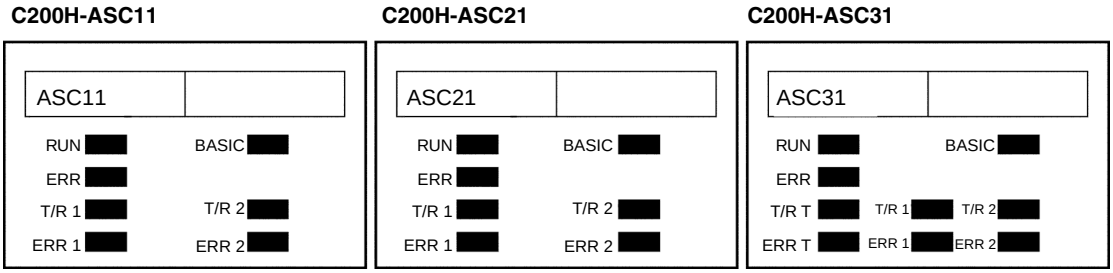
2-1-2 Front Panel

Indicators

Indicator	Color	Name	Display	Meaning	Bit in allocated IR/CIO area words
RUN	Green	Unit operation	Lit	The ASCII Unit is operating normally.	---
			Not lit	An error has occurred. The CPU Unit power is turned OFF, or the MPU in the ASCII Unit is out of control. DM setting is incorrect.	---
BASIC	Green	System mode	Lit	The BASIC program is running.	Bit 7 of n+5
			Slow flashing	INPUT/LINE INPUT/INPUT$ command standby (awaiting input), (L)PRINT command standby (awaiting transmission conditions), or command input standby.	Bit 7 of n+5 (not for INPUT/ LINE INPUT/ INPUT$, and PRINT commands)
			Quick flashing	Transfer standby for the BASIC program. (The flashing cycle will be unstable when transfer of the BASIC program has started.)	---
			Not lit	At power-on waiting for Ctrl+X Key input.	---
ERR	Red	System error/ BASIC error/Battery error	Lit	An error has occurred in the system or the BASIC program.	Bit 2 of n+5
			Flashing	Battery voltage has drop, or battery not connected.	Bit 6 of n+5
T/R 1	Orange	Communications port 1 data transfer	Flashing	Communications port 1 is receiving or transmitting data.	---
T/R 2	Orange	Communications port 2 data transfer	Flashing	Communications port 2 is receiving or transmitting data.	---

Indicator	Color	Name	Display	Meaning	Bit in allocated IR/CIO area words
T/R T	Orange	Terminal port data transfer	Flashing	Terminal port (port 3) is receiving or transmitting data (C200H-ASC31 only).	---
ERR 1	Red	Communications port 1 error	Lit	A reception error (parity error, reception buffer overflow, etc.) has occurred at communications port 1.	Bit 4 of n+5
ERR 2	Red	Communications port 2 error	Lit	A reception error (parity error, reception buffer overflow, etc.) has occurred at communications port 2.	Bit 5 of n+5
ERR T	Red	Terminal port error	Lit	A reception error (parity error, reception buffer overflow, etc.) has occurred at terminal port 1 (C200H-ASC31 only)	Bit 3 of n+5

Note If a PROGRAM MEMORY ERROR (BASIC program memory error) is generated, all error indicators (ERR, ERR 1, ERR 2, ERR T) will flash. (Refer to *9-2 Troubleshooting* for details.)



START/STOP Switch

STOP START

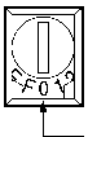
The START/STOP switch starts and stops the BASIC program. If STOP is selected while the program is running, the program will STOP, and if START is selected during a STOP, the program will be restarted from the first line. The effect on operations in relation to initial settings and switch positions are shown in the following table. When START/STOP switch is turned OFF, the program will not run even if the RUN command is sent from the terminal or automatic starting is set in the initial settings.

Operation		Power on or restart		Stopped			Running	
				RUN command from terminal		Switch changed to START	Ctrl+X pressed	Switch changed to STOP
Switch position		**START**	**STOP**	**START**	**STOP**	**STOP to START**	**START**	**START to STOP**
Startup mode (See note.)	**Manual start (00)**	No effect	No effect	RUN	No effect	RUN	Break	Break
	Auto start (5A)	RUN	No effect	RUN	No effect	RUN	Break	Break

Note Either manual or automatic starting is set in bits 00 to 07 of word m + 1 in the DM area words allocated to the Unit.

Unit No. Switch

Set the unit number to between 0 and F. If the same number is set for other Special I/O Units, an I/O Unit Overflow error (fatal) will occur, and the system will not operate.



CPU Unit	Unit No. setting range
C200H-CPU01-E/02-E/03-E C200H-CPU21-E/22-E/23-E/31-E, C200HS-CPU01-E/21-E/31-E/03-E/23-E/33-E, C200HE-CPU11-E/32-E/42-E/11-ZE/32-ZE/42-ZE, C200HG-CPU33-E/43-E/33-ZE/43-ZE, C200HX-CPU34-E/44-E/34-ZE/44-ZE	0 to 9
C200HG-CPU53-E/63-E/53-ZE/63-ZE, C200HX-CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE CS1H-CPU66-E/65-E/64-E/63-E CS1G-CPU45-E/44-E/43-E/42-E	0 to F

2-wire/4-wire Switch (C200H-ASC21 Only)

On the C200H-ASC21, the RS-422A/485 port can be set to either 2-wire or 4-wire communications.

WIRE 2 4

2: 2-wire communications
4: 4-wire communications

Terminating Resistance Switch (C200H-ASC21 Only)

On the C200H-ASC21, the RS-422A/485 port's terminating resistance (220 Ω) can be turned ON or OFF.

TERM OFF ON

ON: Terminating resistance connected.
OFF: Terminating resistance disconnected.

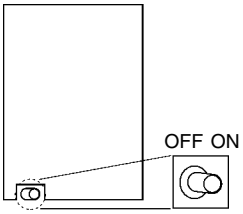
2-1-3 Rear Panel

User Memory Default Switch

When an error occurs in the ASCII Unit as a result of damage to the user memory, this switch can be used to forcibly format (delete all programs from memory) the user memory and return the ASCII Unit to normal operating status.

Note During normal operation, make sure that the user memory switch is turned OFF (set to the left). If the switch is turned ON (set to the right) and the power turned ON, all programs in the ASCII Unit will be deleted. Refer to *9-2 Troubleshooting* for details on processing errors.

ON: All user memory will be formatted (returned to default settings)
OFF: User memory will not be formatted.



Battery

The battery backs up the ASCII Unit's BASIC program, trace memory, and real-time clock settings when power is not being supplied. If the battery voltage drops, or the battery is not connected, the Error Indicator on the front panel will flash.

Part name (replacement part)	Model
Battery Set	C200H-BAT09

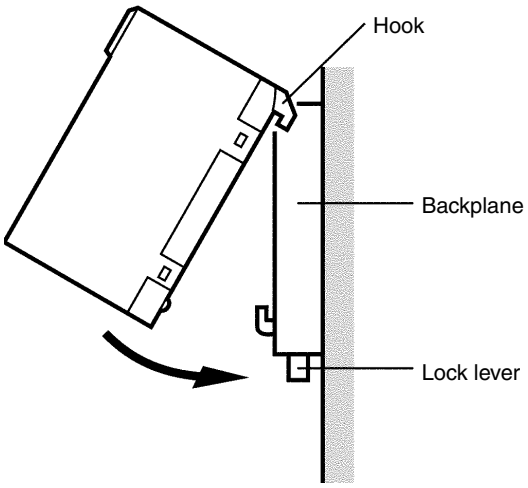
Refer to *9-5 Maintenance* for details on replacing the battery.

2-2 Installation

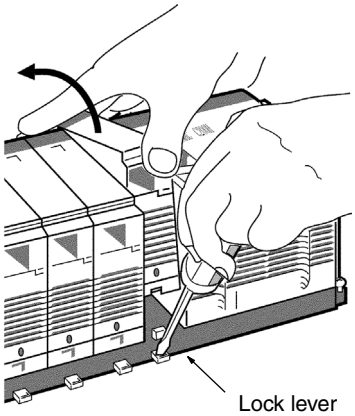
2-2-1 Installation Method

1,2,3...

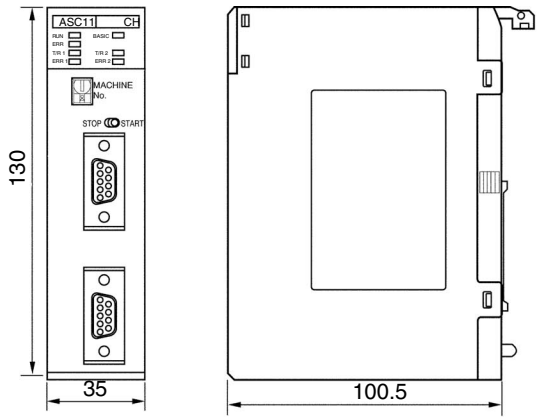
1. Attach the hooks on the upper section of the ASCII Unit onto the Backplane.
2. Insert the ASCII Unit connector into the Backplane connector.



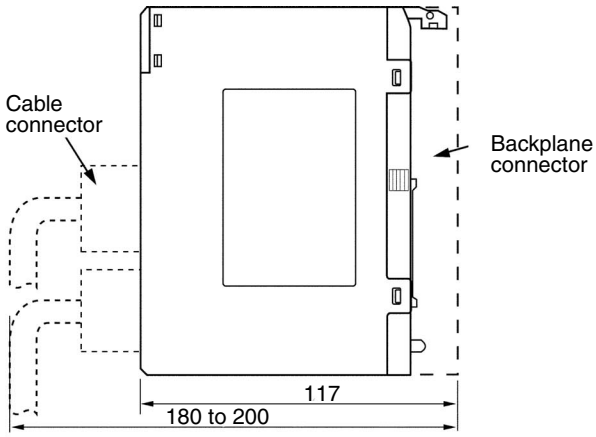
Note When removing the ASCII Unit, lift it out while pressing down on the lock lever with a screwdriver, as shown in the following illustration.



Dimensions



Unit Dimensions when Mounted



Mounting Positions and Number of Units

PC	CPU Unit	Total No. of Units on CPU Rack, Expansion I/O Racks, and SYSMAC BUS Slave Racks	Mounting restrictions
C200HX/HG/HE	C200HE-CPU11-E/32-E/42-E/11-ZE/32-ZE/42-ZE C200HG-CPU33-E/43-E/33-ZE/43-ZE C200HX-CPU34-E/44-E/34-ZE/44-ZE	10	None
	C200HG-CPU53-E/63-E/53-ZE/63-ZE C200HX-CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE	16	
C200HS	C200HS-CPU01-E/01-EC/21-E/21-EC/31-E/03-E/23-E/33-E	10	
CS1	CS1H-CPU66-E/65-E/64-E/63-E CS1G-CPU45-E/44-E/43-E/42-E	16	None
C200H	C200H-CPU01-E/02-E/03-E/11-E/21-E/23-E/31-E	10	Do not mount Unit to rightmost two slots on CPU Rack.

Restrictions on SYSMAC BUS Slave Racks

The following restrictions apply when mounting Units to SYSMAC BUS Slave Racks.

Group	A	B	C	D
Unit	ASCII Unit, High-speed Counter Unit, Position Control Unit (NC111/112/113/213) Analog I/O Unit, ID Sensor Unit, Fuzzy Logic Unit	Multi-point I/O Unit	Temperature Sensor Unit, Voice Unit	Position Control Unit (NC211) Motion Control Unit
Number of Units mountable for each group per Remote I/O Master Unit	4 Units	8 Units	6 Units	2 Units
Number of Units mountable for all groups combined per Remote I/O Master Unit	3A + B + 2C + 6D ≤ 12, and A + B + C + D ≤ 8			

2-2-2 Connections

External Devices

External devices are connected to the RS-232C or RS-422A/485 port. For interface code/timing, refer to *SECTION 4 Data Exchange with General-purpose External Devices*.

RS-232C Connections (C200H-ASC11/21/31)

Using the enclosed connector, connect all pins to match the external device interface.

C200H-ASC11/21 **C200H-ASC31**

Pin	Symbol	Name	Direction	Comments
1	FG	Frame ground	-	
2	SD	Send data	Output	
3	RD	Receive data	Input	
4	RTS	Request to send	Output	
5	CTS	Clear to send	Input	
6	-	Not used	-	Only for terminal port of C200H-ASC31.
	5V	5V output	Output	Other ports
7	DSR	Data set ready	Input	
8	DTR	Data terminal ready	Output	
9	SG	Signal ground	-	
Hood	FG	Frame ground	-	

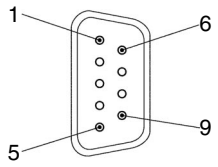
RS-232C Interface Specifications

Item	Specifications	
Electrical characteristics	Conform to EIA RS-232C	
Synchronization	Start-stop	
Baud rate	300/600/1,200/2,400/4,800/9,600/19,200/38,400 bps (38,400 bps cannot be used for the terminal port on the ASC31.)	
Transmission mode	Point-to-point (1:1)	
Communications buffers	Transmit: 512 bytes; receive: 512 bytes	
Flow control	RTS/CTS flow, Xon/Xoff flow control supported	
Connector type	D-sub 9-pin for both ports	
Applicable connector (enclosed)	Plug: XM2A-0901 or equivalent Connector hood: XM2S-0911 or equivalent	
Recommended cable	UL2464 AWG28x5P IFS-RVV-SB (UL product) AWG28x5P IFVV-SB (non-UL product)	Fujikura Ltd.
	UL2464-SB(MA) 5Px28AWG (7/0.127) (UL product) CO-MA-VV-SB 5Px28AWG (7/0.127) (non-UL product)	Hitachi Cable, Ltd.
Cable length	15 m max.	

Note For details on connecting cables, refer to *Appendix F Wiring RS-232C* or *RS-422A/485 Cable Connectors*.

RS-422A/485 Connections (C200H-ASC21 Only)

Using the enclosed connector, connect the pins to match the external device interface.



Pin	Symbol	Name	Direction
1	SDA	Send data A (–)	Output
2	SDB	Send data B (+)	Output
3	NC	Not used	-
4	NC	Not used	-
5	NC	Not used	-
6	RDA	Receive data A (–)	Input
7	NC	Not used	-
8	RDB	Receive data B (+)	Input
9	NC	Not used	-
Hood	FG	Frame ground	-

RS-422A/485 Interface Specifications

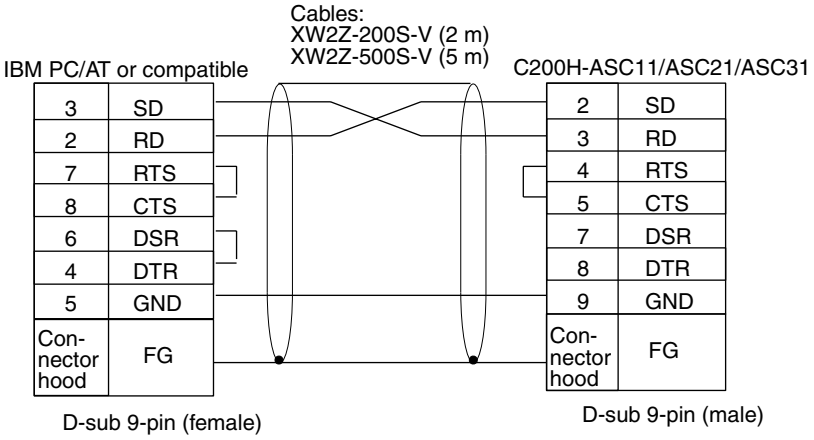
Item	Specifications	
Communications mode	Half duplex (1 ms is required after transmission has been completed until the ASCII Unit is ready to receive.)	
Electrical characteristics	Conform to EIA RS-422A/485	
Synchronization	Start-stop	
Baud rate	300/600/1,200/2,400/4,800/9,600/19,200/38,400 bps	
Transmission mode	Point-to-multipoint (1:N)	
Communications buffer	Transmit: 512 bytes; receive: 512 bytes	
Flow control	None	
Applicable connector (enclosed)	Plug: XM2A-0901 or equivalent Connector hood: XM2S-0911 or equivalent	
Recommended cable	UL2464 AWG28x5P IFS-RVV-SB (UL product) AWG28x5P IFVV-SB (non-UL product)	Fujikura Ltd.
	UL2464-SB(MA) 5Px28AWG (7/0.127) (UL product) CO-MA-VV-SB 5Px28AWG (7/0.127) (non-UL product)	Hitachi Cable, Ltd.
Cable length	500 m max.	

Note

1. For details on connecting cables, see *Appendix F Wiring RS-232C or RS-422A/485 Cable Connectors*.
2. When the 2/4-wire switch is set for 2-wire operation, use either pins 1 and 2, or pins 6 and 8. (Pins 1 and 6 are short-circuited, and pins 2 and 8 are short-circuited.)

Personal Computer Used as Terminal

Check the wiring between the RS-232C connector and the computer, and using the appropriate connectors, prepare the connecting cables using the the pin arrangement shown in the following diagram.



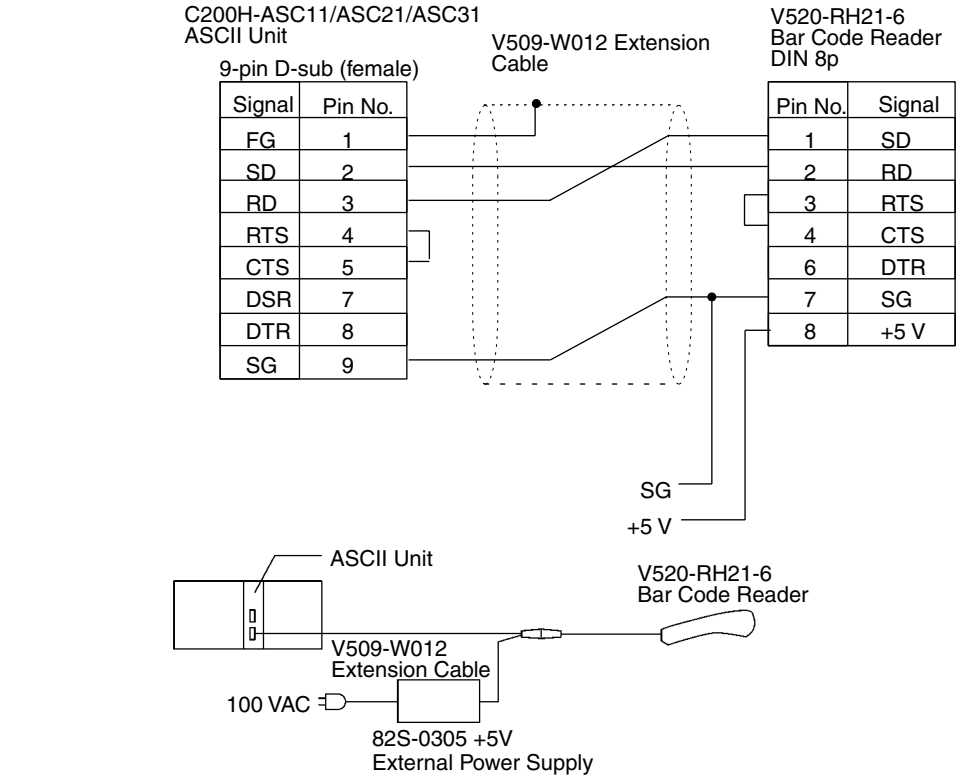
Note

1. DSR-DTR signals are not connected back to each other for the cable used above, so do not specify that the DSR signal is checked when the LOAD command is executed. (If the specification is omitted, the DSR signal will not be checked by default.)
2. CTS-RTS and DSR-DTR are not cross-connected, so if flow control is required, use Xon/Xoff flow control.

2-2-3 General-purpose Peripheral Device Connections

Bar Code Reader Connected with RS-232C

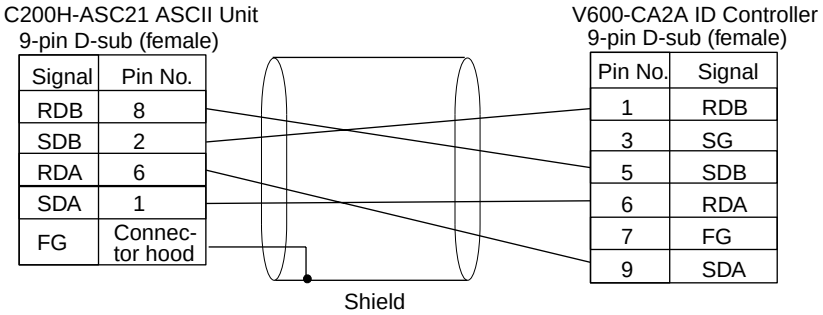
Connecting an OMRON V500 Bar Code Reader

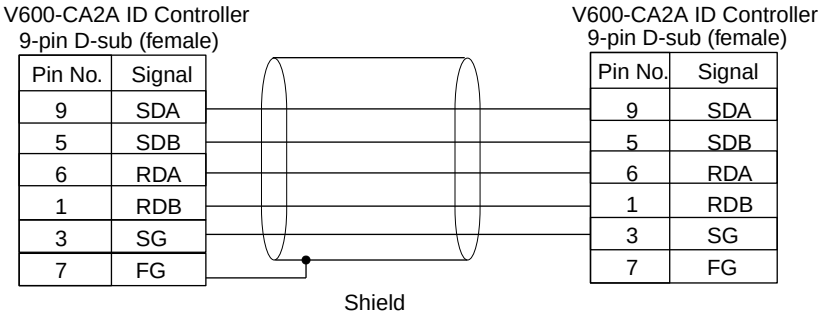


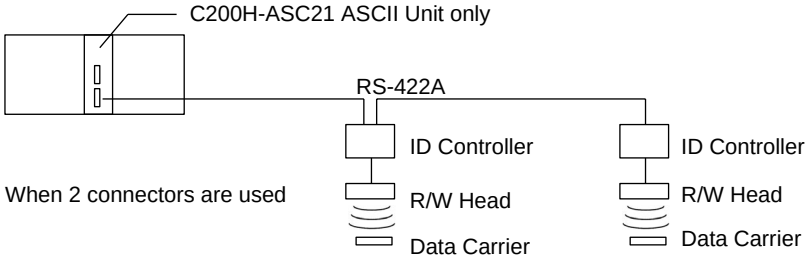
Note In order to prevent malfunctions, when external devices have a FG, ground both the external devices and the ASCII Unit with shielded cables.

ID Controller Connected with RS-422A (4-wire)

Connecting a V600/620 ID Controller



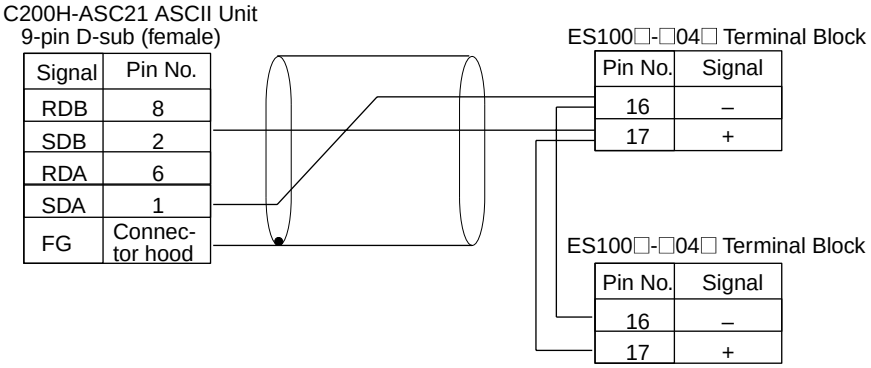




Note
1. Make sure that the shield is grounded at either the ID Controller end or the ASCII Unit end, but not both ends.
2. Make sure to set the 2/4-wire switch on the C200H-ASC21 Unit to 4 wires.

Digital Operator Connected with RS-485 (2-wire)

Connecting an OMRON ES100X Digital Operator



Note

1. Make sure that the shield is grounded at either the Digital Operator end or the ASCII Unit end, but not both ends.
2. Make sure to set the 2/4-wire switch on the C200H-ASC21 Unit to 2 wires.

SECTION 3
IR/CIO and DM Area Allocations

This section describes the methods used to allocate words to the ASCII Unit in the IR/CIO and DM Areas and the applications of the bits and words in these areas.

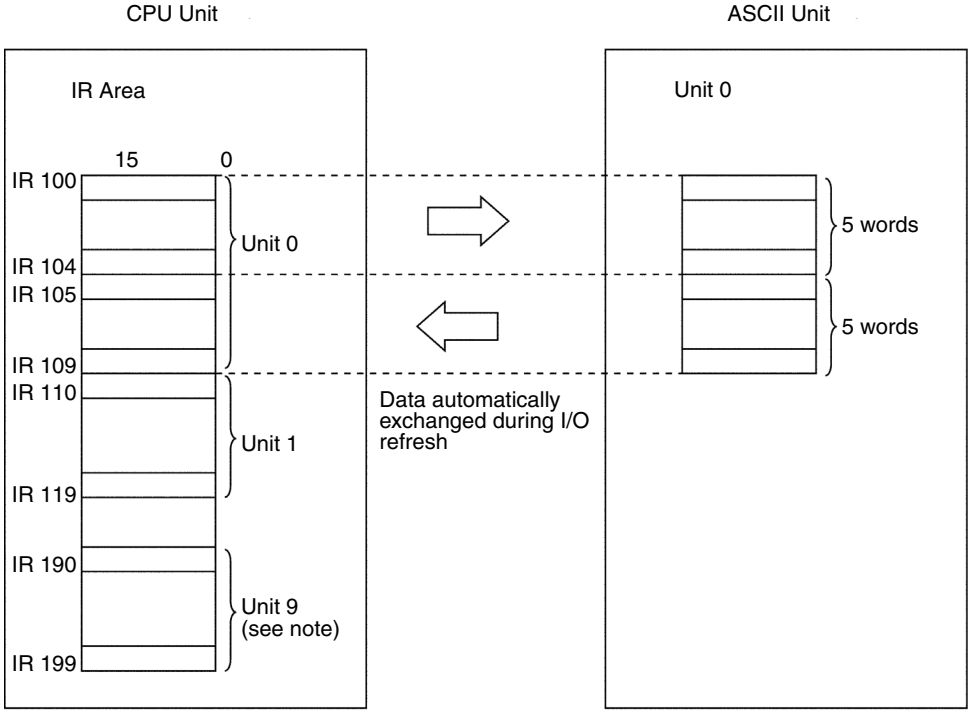
3-1 IR/CIO Area Allocations 30
3-1-1 Overview. 30
3-1-2 Overview of IR/CIO Area Allocations 31
3-2 DM Area Allocations 38
3-2-1 Overview. 38
3-2-2 Overview of DM Allocations 40

3-1 IR/CIO Area Allocations

3-1-1 Overview

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

Data Exchange for the C200HX/HG/HE, C200H, C200HS



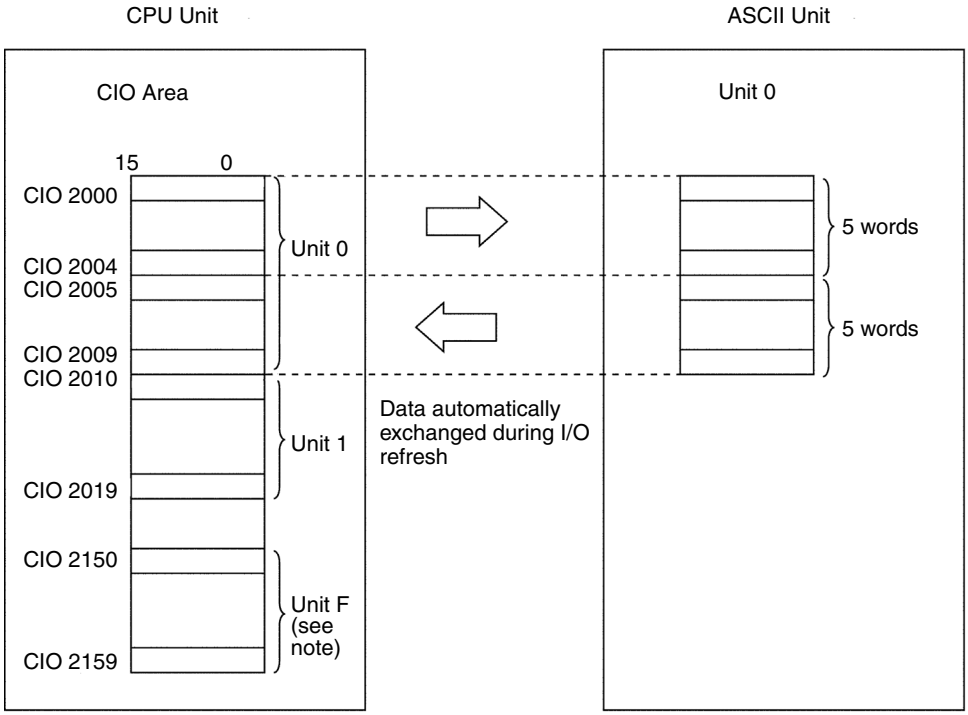
Note For CPU Units C200HG-CPU53-E/63-E/53-ZE/63-ZE and C200HX -CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE, IR 400 to IR 459 are also allocated (maximum of 16 Special I/O Units).

Special I/O Area Allocation for the C200HX/HG/HE, C200H, C200HS

Unit No.	Allocated words	Unit No.	Allocated words	Unit No.	Allocated words	Unit No.	Allocated words
0	IR 100 to IR109	4	IR 140 to IR 149	8	IR 180 to IR 189	C	IR 420 to IR 429 (See note.)
1	IR 110 to IR 119	5	IR 150 to IR 159	9	IR 190 to IR 199	D	IR 430 to IR 439 (See note.)
2	IR 120 to IR 129	6	IR 160 to IR 169	A	IR 400 to IR 409 (See note.)	E	IR 440 to IR 449 (See note.)
3	IR 130 to IR 139	7	IR 170 to IR 179	B	IR 410 to IR 419 (See note.)	F	IR 450 to IR 459 (See note.)

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

Data Exchange for the CS1 Series



Special I/O Area Allocation for the CS1 Series

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

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

The following data is exchanged automatically during the I/O refresh period. The first word allocated to the ASCII Unit in the IR/CIO Area is “n.” The value of “n” can be calculated from the unit number using the following equation.

- C200HX/HG/HE, C200H, C200HS

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

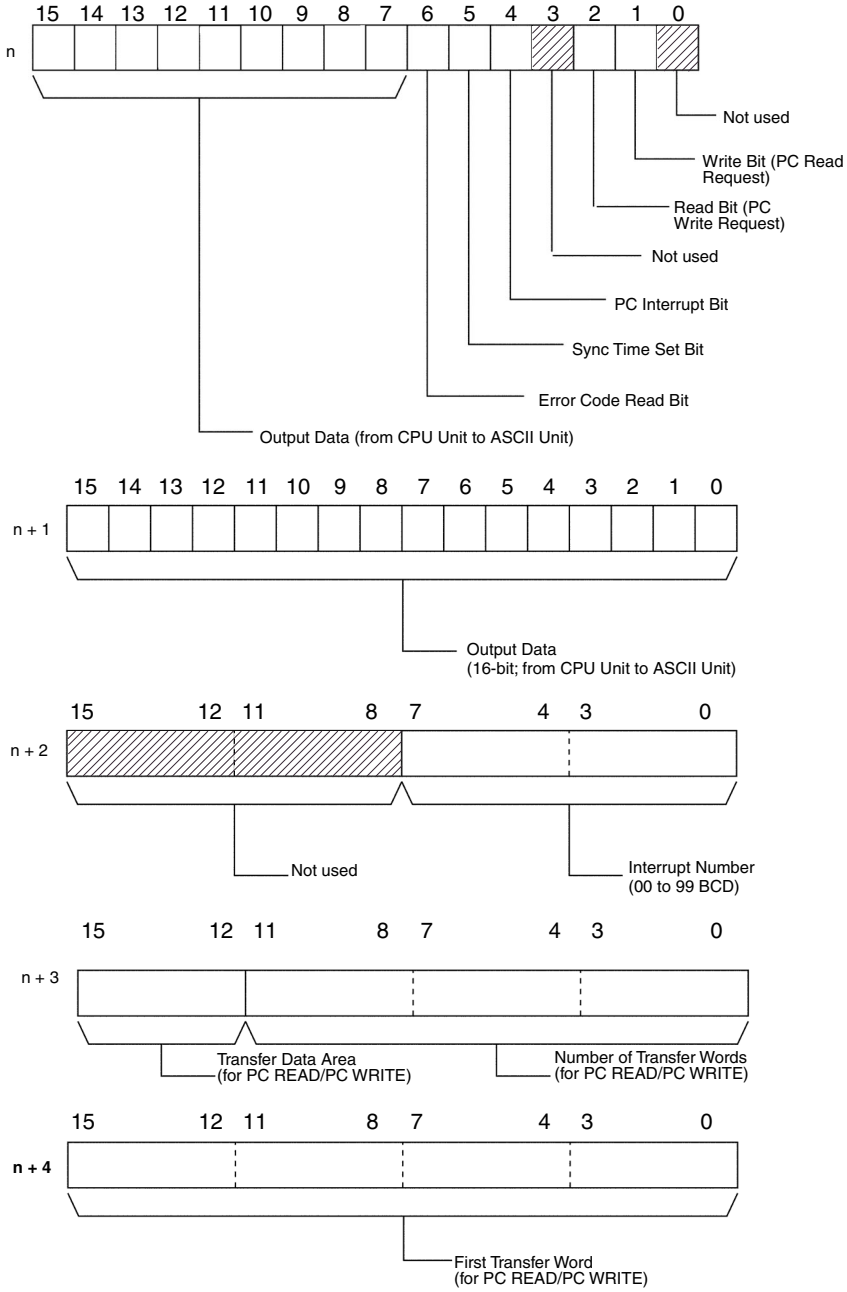
  Unit numbers A to F: n = 400 + 10 x (Unit number – 10)

- CS1 Series

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

Outputs

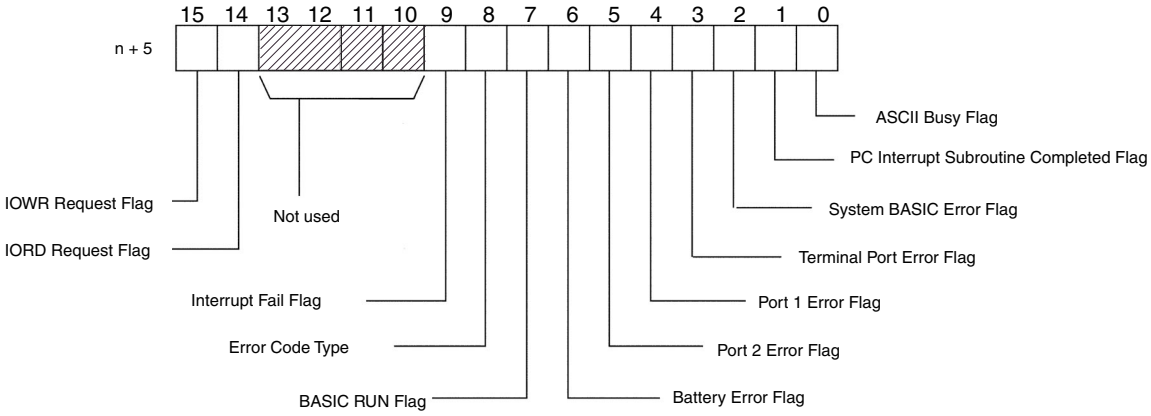
The following words are output from the CPU Unit to the ASCII Unit.

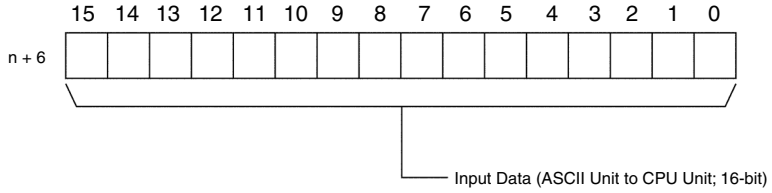


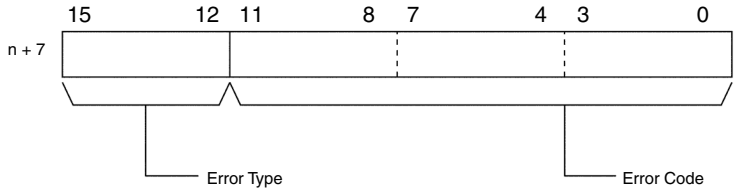
Word	Bit	Name	Function	Use
n	00	Not used	---	---
	01	Write Bit	1: Establishes the execution conditions for PC READ (Data transferred from CPU Unit to ASCII Unit according to the specified First Transfer Word and Number of Transfer Words.) 0: PC READ execution wait.	When this bit is turned ON by the user program in the CPU Unit, data is transferred according to the PC READ command at the ASCII Unit.
	02	Read Bit	1: Establishes the execution conditions for PC WRITE (Data transferred from ASCII Unit to CPU Unit according to the specified First Transfer Word and Number of Transfer Words.) 0: PC WRITE execution wait.	When this bit is turned ON by the user program in the CPU Unit, data is transferred according to the PC WRITE command at the ASCII Unit.
	03	Not used	---	---
	04	PC Interrupt Bit	0 to 1: Sends an interrupt to the ASCII Unit. The interrupt subroutine for the ON PC GOSUB is branched to when this bit changes from 0 to 1.	When this bit is switched from OFF to ON by the user program in the CPU Unit, an interrupt is sent to the ASCII Unit. It is necessary to set the interrupt number in n + 2 bits 0 to 7 when using the PC Interrupt Bit.
	05	Sync Time Set Bit	0 to 1: Sets the ASCII Unit RTC according to the CPU Unit RTC.	The time on the clock in the ASCII Unit is set to the time on the clock in the CPU Unit.
	06	Error Code Read Bit	0 to 1, 1 to 0: Reads the error code to n + 7.	By toggling this bit (0 to 1, 1 to 0) the last error code from the ASCII Unit's internal error log can be read and set in n + 7. The error status in the ASCII Unit will not be cleared when the error code is read using this bit.
	07 to 15	Output Data	Data transferred from the CPU Unit to the ASCII Unit. The ASCII Unit reads this data via the BASIC program's PC GET command.	The CPU Unit writes data here using the MOV instruction in the user program. The ASCII Unit can read this data via the PC GET command.
n + 1	00 to 15	Output Data	Data transferred from the CPU Unit to the ASCII Unit. The ASCII Unit reads this data via the BASIC program's PC GET command.	The CPU Unit writes data here using the MOV instruction in the user program. The ASCII Unit can read this data via the PC GET command.
n + 2	00 to 07	Interrupt Number	00 to 99 BCD (2-digit)	When the ON PC GOSUB command interrupt number and this interrupt number are the same and the PC Interrupt Bit changes from OFF to ON, an interrupt will be sent from the CPU Unit to the ASCII Unit. The interrupt subroutine will be executed in response.
	08 to 15	Not used	---	---

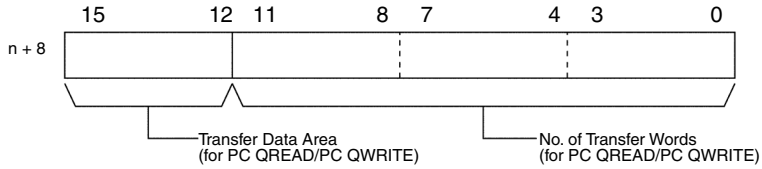
Word	Bit	Name	Function			Use
n + 3	00 to 11	No. of Transfer Words	Number of words transferred when ASCII Unit PC READ/PC WRITE command is executed (1 to 255).			Data is written via the MOV instruction from the user program in the CPU Unit. If the Write Bit is ON, the data will be transmitted by the ASCII Unit's PC READ/PC WRITE command at the next I/O refresh period.
	12 to 15	Transfer Data Area	The memory area for ASCII Unit PC READ/PC WRITE commands is set using the following codes.			
			No.	Data area		
				C200HX/HG/HE, C200H, C200HS	CS1 Series	
			0	DM Area		
			1	IR Area	CIO Area	
			2	LR Area	Data Link Area	
			3	HR Area	H000 to H099 in Holding Area	
			4	AR Area	H100 to H127 in Holding Area	
			5	EM Area bank 0		
			6	TC Area	Timer Area	
n + 4	00 to 15	First Transfer Word (address)	First word transferred from the CPU Unit memory address when the ASCII Unit's PC READ/PC WRITE command is executed.			

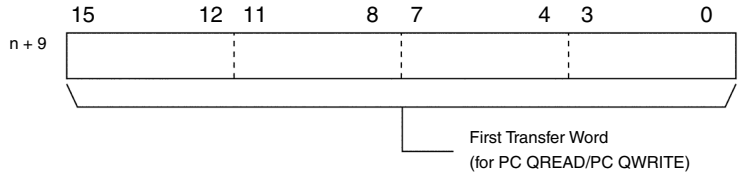
Input The following words are input from the ASCII Unit to the CPU Unit.











Word	Bit	Name	Function	Use
n + 5	00	ASCII Busy Flag	1: The ASCII Unit is exchanging data with the CPU Unit. There is an error in the DM settings. The ROMSAVE command is being executed and data is being transferred to the flash ROM. 0: Transfer is not occurring.	When this flag is ON, the next transfer with the ASCII Unit should not be executed in the user program in the CPU Unit. Use this flag's address in an NC input condition for an OUT instruction controlling the Read Flag and Write Flag outputs to the ASCII Unit or as an NC input condition for the IOWR (#00□□) instruction.
	01	PC Interrupt Subroutine Completed Flag	Turns ON when the interrupt subroutine at the PC is completed.	Until this bit toggles, subroutine results are not yet available. (See Note.)
	02	System BASIC Error Flag	1: System error or BASIC error 0: Normal	---
	03	Terminal Port Error Flag	1: Communications error at the terminal port 0: Normal	These bits are cleared by executing an ERC command.
	04	Port 1 Error Flag	1: Port 1 transmission error (parity error, reception buffer overflow, etc.) 0: Normal	
	05	Port 2 Error Flag	1: Port 2 transmission error (parity error, reception buffer overflow, etc.) 0: Normal	
	06	Battery Error Flag	1: Battery voltage drop or no battery connected 0: Normal	---
	07	BASIC RUN Flag	1: BASIC program running 0: BASIC program stopped	---
	08	Error Code Type	1: New 0: Old	This flag shows whether the error for the error code being read was generated before the last RUN.
	09	PC Interrupt Fail Flag	1: PC Interrupt has failed 0: Normal	This flag turns ON when a PC interrupt fails due to the PC OFF (stop interrupt) condition in the ASCII Unit. The PC Interrupt Bit will turn ON for the next cycle and will turn OFF for the following cycle.
	10 to 13	Not used	---	---
	14	IORD Request Flag	1: IORD (#FD00) execution request by the ASCII Unit to the CPU Unit (When a PC QWRITE command is executed, this flag turns ON automatically, and after the data is transferred by the CPU Unit using the IORD (#FD00) instruction, it automatically turns OFF at the next I/O refresh period.) 0: No request	This flag automatically turns ON when a PC QWRITE command is executed to request execution of an IORD (#FD00) instruction from the CPU Unit.
	15	IOWR Request Flag	1: IOWR (#FD00) execution request by the ASCII Unit to the CPU Unit (When a PC QREAD command is executed, this flag turns ON automatically, and after the data is transferred by the CPU Unit using the IOWR (#FD00) instruction, it automatically turns OFF at the next I/O refresh period.) 0: No request	This flag automatically turns ON when a PC QREAD command is executed to request execution of an IOWR (#FD00) instruction from the CPU Unit.

Word	Bit	Name	Function	Use
n + 6	00 to 15	Input Data	The data (16 bit) transferred from the ASCII Unit to the CPU Unit by a PC PUT command at the ASCII Unit.	The ASCII Unit's PC PUT command writes data to the IR/CIO words allocated to the ASCII Unit. After the next I/O refresh, this 16-bit, 8-bit or 1-bit data is read using the CPU Unit's MOV or LD instruction.
n + 7	00 to 11	Error Code	Error code	Each time word n bit 06 changes status (ON to OFF or OFF to ON) in the CPU Unit, the next newest error code and error type are read from the ASCII Unit's error code table, and stored in the CPU Unit.
	12 to 15	Error Type	B: BASIC error 0: System error 1: Port 1 communications error 2: Port 2 communications error 3: Terminal port communications error (C200H-ASC31 only)	
n + 8	00 to 11	No. of Transfer Words	The number of words transferred when the ASCII Unit's PC QREAD/PC QWRITE command is executed. (1 to 128)	The ASCII Unit writes data for these words when the PC QREAD/PC QWRITE commands are executed, and then sets the IORD/IOWR Request Flag in word n + 5 bit 14/15. When the user program in the CPU Unit detects the ON status of the flag using an LD instruction, it can access the words using the IORD/IOWR (#FD00) instructions. (See note 2.)
	12 to 15	Transfer Data Area	The memory area for the ASCII Unit PC QREAD/PC QWRITE commands.	

No.	Data area: C200HX/HG/HE, C200H, C200HS	Data area: CS1 Series
0	DM Area	
1	IR Area	CIO Area
2	LR Area	Data Link Area
3	HR Area	H000 to H099 in Holding Area
4	AR Area	H100 to H127 in Holding Area
5	EM Area (bank number 0 only)	
6	TC Area	Timer Area

n + 9	00 to 15	First Transfer Word (address)	The first word to be transferred (CPU Unit memory address) for the ASCII Unit's PC QREAD/PC QWRITE commands.	

Note

1. An OR can be taken of DIFU/DIFD instructions to detect changes in the status of the Interrupt Subroutine Completed Flag. This flag can be used when executing an IORD instruction (#00□□) when a PC EPUT command is received during a subroutine.
2. The data set in these words by execution of the PC QREAD/PC QWRITE commands is used for monitoring from the CPU Unit. Actual data transfer is performed according to the operands specified for the IOWR/IORD (#FD00) instructions. Care is required for CS1-series PCs because the addressing range for PC QREAD and PC QWRITE is different from that for IOWR/IORD (#FD00). Dummy addresses can be used when the desired addresses cannot be addressed directly with PC QREAD/PC QWRITE. Refer to *6-3 Details of the Data Exchange Methods* for details on page 75.

3-2 DM Area Allocations

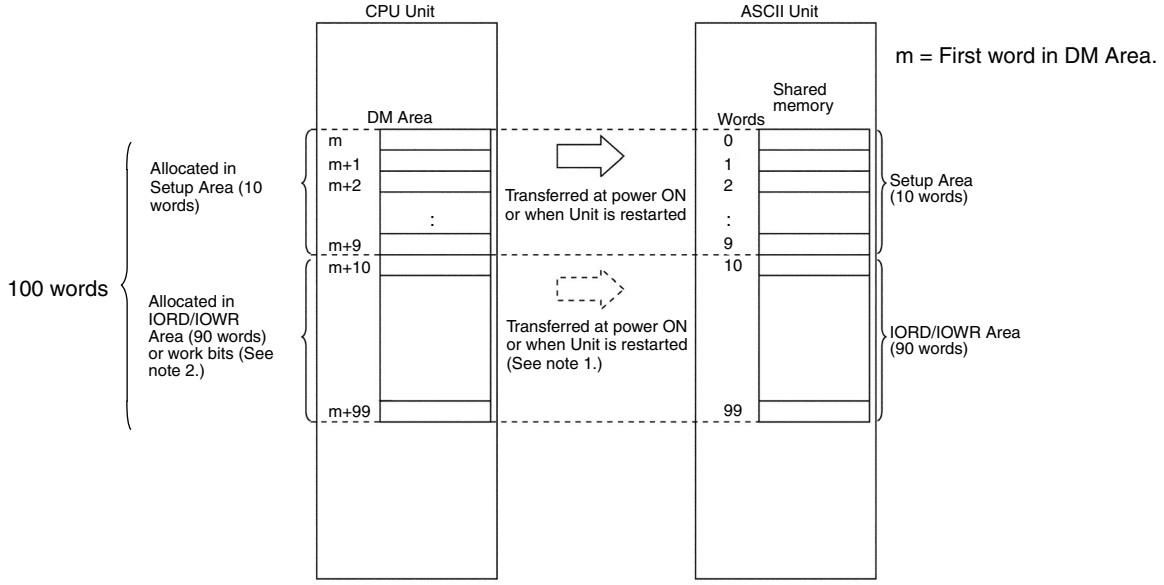
3-2-1 Overview

Each ASCII Unit is also allocated 100 words as shared memory from the CPU Unit's Data Memory Area. The unit number determines which 100 words are allocated. The shared memory (100 words) in the ASCII Unit is divided into two regions as follows:

- Setup Area (10 words): Words 0 to 9
- IORD/IOWR Area (90 words): Words 10 to 99

The ASCII Unit's internal shared memory does not have a battery backup. When the power supply is turned off, it will return to default values. Data from the CPU Unit will be resent every time the power is turned on or the Unit is restarted.

The first word allocated to the ASCII Unit in the DM area and the first word in the shared memory in the ASCII Unit is "m."



Note 1. The IORD/IOWR Area can be set to be transferred or not transferred in bits 08 to 11 of word m+1 in shared memory in the Setup Area.

2. The IORD/IOWR Area (m+10 to m+99) can be used as work bits when not used for the transfer to the ASCII Unit as described in note 1.

DM Area Allocation **C200HX/HG/HE, C200H, C200HS**

Unit No.	Words	Unit No.	Words	Unit No.	Words
0	DM 1000 to DM 1099	6	DM 1600 to DM 1699	A	DM 2000 to DM 2099
1	DM 1100 to DM 1199	7	DM 1700 to DM 1799	B	DM 2100 to DM 2199
2	DM 1200 to DM 1299	8	DM 1800 to DM 1899	C	DM 2200 to DM 2299
3	DM 1300 to DM 1399	9	DM 1900 to DM 1999	D	DM 2300 to DM 2399
4	DM 1400 to DM 1499	---	---	E	DM 2400 to DM 2499
5	DM 1500 to DM 1599	---	---	F	DM 2500 to DM 2599

Note Unit numbers A to F are used for the C200HG-CPU53-E/63-E/53-ZE/63-ZE and C200HX-CPU54-E/64-E/54-ZE/64-ZE/65-ZE/85-ZE only.

CS1-series PCs

Unit No.	Words	Unit No.	Words	Unit No.	Words
0	D20000 to D20099	6	D20600 to D20699	A	D21000 to D21099
1	D20100 to D20199	7	D20700 to D20799	B	D21100 to D21199
2	D20200 to D20299	8	D20800 to D20899	C	D21200 to D21299
3	D20300 to D20399	9	D20900 to D20999	D	D21300 to D21399
4	D20400 to D20499	---	---	E	D21400 to D21499
5	D20500 to D20599	---	---	F	D21500 to D21599

Setup Area (10 words)

The first 10 words (0 to 9) of the allocated 100 words in the DM area are the ASCII Unit's Setup Area. When power is turned ON or the Unit is restarted, it will be possible to transfer the contents of these words from the CPU Unit to the ASCII Unit.

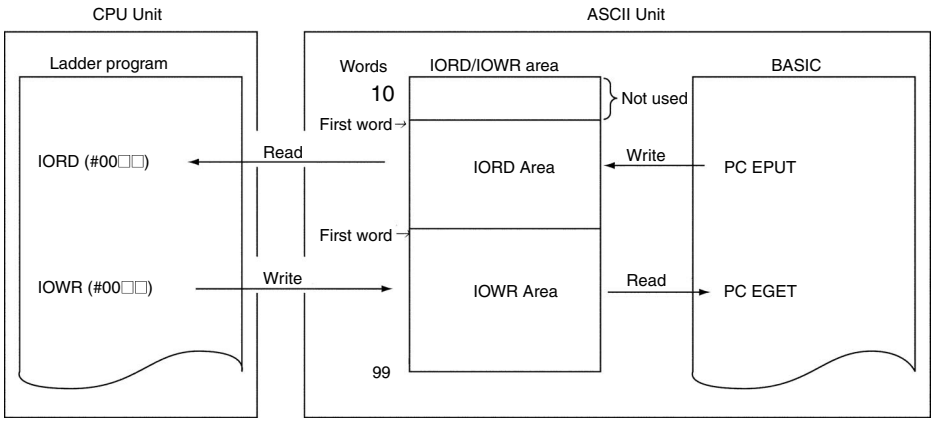
The Setup Area consists of the following parameters. For details, refer to *3-2-2 Overview of DM Allocations*.

- Communications parameters for the ASCII Unit ports.
- Number of the program activated at startup.
- Division of the remaining 90 words into an IOWR Area and IORD Area.
- Transfer ON/OFF setting for the remaining 90 words from the CPU Unit to the ASCII Unit at power ON or when the Unit is restarted.

IORD/IOWR Areas (90 Words)

The 90 words (10 to 99) remaining out of the 100 words are called IORD/IOWR Areas and are used for IOWR/IORD instructions from the CPU Unit. Within the ASCII Unit, these are the areas that can be read/written using the PC EGET/EPUT commands from the ASCII Unit.

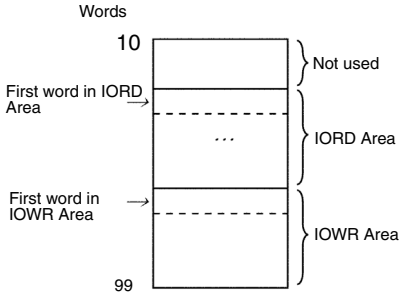
Within these 90 words, you can set which words the CPU Unit can read using the IORD instruction (i.e., the area written by the ASCII Unit via the PC EPUT command called the IORD Area) and which words the CPU Unit can write using the IOWR instruction (i.e., the area read by the ASCII Unit using the PC EGET command called the IOWR Area).

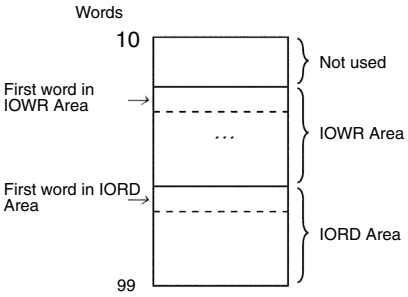


You can also set whether or not the IORD and IOWR Areas in the DM words allocated to the ASCII Unit in the CPU Unit are transferred to the ASCII Unit's IORD/IOWR Area when power is turned ON or the ASCII Unit is restarted. This parameter is set in word m + 1 bits 08 to 11. If not transferred, the DM area words (m+10 to m+99) can be used as word bits for other purposes in programming.

Note Setting the IORD/IOWR Areas within the following ranges.

- Set the first word in the IORD Area, and the first word in the IOWR Area.
- The IORD Area and IOWR Area can be set in any order and the first part of the allocated DM words can be left unused by either area.





3-2-2 Overview of DM Allocations

The first address in the DM words allocated in the CPU Unit is “m.”

C200HX/HG/HE, C200H, C200HS: m = DM 1000 + 100 × Unit No.
CS1-series PCs: m = D20000 + 100 × Unit No.

Shared memory address	DM address	Bit	Name	Function	Settings (hex)	ASCII Unit’s default (hex)
0	m	00 to 07	Program No.	Sets the startup program number.	00: No.1 01: No.1 02: No.2 03: No.3 04: No.4	00
		08 to 15	Automatic transfer from flash ROM	Determines whether the program is read from the flash ROM to the user memory area at power ON or when the Unit is restarted.	00: Do not transfer (user memory mode) 5A: Transfer flash ROM to user memory area	00
01	m + 1	00 to 07	Startup mode	Determines whether or not the program runs automatically at power ON or when the Unit is restarted. When manual start is designated, start using the START/STOP switch on the front panel of the ASCII Unit or the RUN command from the terminal. (See page 20 for details on the START/STOP switch.)	00: Manual start 5A: Automatic start	00
		08 to 11	IORD/IOWR Area transfer	Determines whether or not the CPU Unit’s 90 words (DM m + 10 to m + 99) are sent to the ASCII Unit’s shared memory (IORD/IOWR) area at power on or restart.	0: No transfer 1: Transfer	0
		12 to 15	Not used	---	Set to 0 (OFF)	0

Shared memory address	DM address	Bit	Name	Function	Settings (hex)	ASCII Unit's default (hex)
02	m + 2	00 to 07	Port #1 baud rate	Sets the baud rate for port #1.	00: 9,600 01: 300 02: 600 03: 1,200 04: 2,400 05: 4,800 06: 9,600 07: 19,200 08: 38,400 (bps)	00
		08 to 15	Port #1 DMA transmission (C200H-ASC31 only)	Sets DMA transmission for port #1. (See page 48 for details on DMA.)	00: No DMA sent 5A: DMA sent	00
03	m + 3	00 to 07	Port #2 baud rate	Sets the baud rate for port #2.	00: 9,600 01: 300 02: 600 03: 1,200 04: 2,400 05: 4,800 06: 9,600 07: 19,200 08: 38,400 (bps)	00
		08 to 15	Port #2 DMA transmission	Sets DMA transmission for port #2. (See page 48 for details on DMA.)	00: No DMA sent 5A: DMA sent	00
04	m + 4	00 to 07	Terminal port #3 baud rate (C200H-ASC31 only)	Sets the baud rate of terminal port #3.	00: 9,600 01: 300 02: 600 03: 1,200 04: 2,400 05: 4,800 06: 9,600 07: 19,200 (bps)	00
		08 to 15	Number of transfer words per cycle (C200HX/HG/HE or CS1 only)	Determines the maximum number of transfer words per CPU Unit cycle for PC READ/PC WRITE commands (including @ variants.)	00: 20 words max. 5A: 127 words max. 5A is available only for the C200HX/HG/HE or CS1. (Not available for remote I/O.)	00
05	m + 5	00 to 07	Not used	---	---	00
		08 to 11	Screen size: Width	Determines cursor movement in Edit Mode.	0: 80 characters 1: 40 characters 2: 132 characters	0
		12 to 15	Terminal emulation	Sets the terminal emulation mode for key operation, etc.	0: VT100 mode 1: FIT10 mode	0
06	m + 6	00 to 07	First word in IOWR Area	Sets the first word in the IOWR Area between address 10 and 99.	00: No IOWR Area 10 to 99: IOWR first word	00
		08 to 15	Not used			
07	m + 7	00 to 07	First word in IORD Area	Sets the first word in the IORD Area between address 10 and 99.	00: No IORD Area 10 to 99: IORD first word	00
		08 to 15	Not used			
08	m + 8	00 to 15	Not used			

Shared memory address	DM address	Bit	Name	Function	Settings (hex)	ASCII Unit's default (hex)
09	m + 9	00 to 15	Not used			
10 to 99	m + 10 to m + 99	00 to 15	IORD Area and IOWR Area (set first words at address 6 and 7)			00

Note

1. Do not specify settings other than those listed in the *Settings* column above.
2. When the start mode is set to start automatically, the message "Please wait compiling...," which is normally output to the terminal port when a program is run, will not be output when the power is turned ON/OFF or when the START/STOP switch is used.
3. When there is an error in the DM settings, the ERR indicator will be lit. Since terminals cannot be connected when this indicator is lit, use a Programming Console to confirm the error code, reset the DM settings correctly, and either turn the power OFF then ON again, or restart the ASCII Unit.

SECTION 4
Data Exchange with General-purpose External Devices

This section describes the ASCII Unit commands in conjunction with transmission control signals for opening and closing communications ports and for sending and receiving data between the ASCII Unit and external devices.

4-1 Overview 44
4-2 Opening a Communications Port 44
4-2-1 OPEN# Command 44
4-2-2 OPEN Command Specifications. 45
4-2-3 Control/Monitor Commands for Transmission Control Signals. . . . 46
4-2-4 Device Symbol 46
4-3 Sending Data 47
4-3-1 PRINT# Command 47
4-3-2 DMA Data Transmission 48
4-3-3 Transmission Control Signals 48
4-4 Receiving Data 51
4-4-1 INPUT# Command 51
4-4-2 Transmission Control Signals 51
4-4-3 INPUT$ Function 52
4-5 Closing a Communications Port 53
4-5-1 Transmission Control Signals 53
4-5-2 Transmission Control Signal Timing Charts 53
4-6 Communications Errors 59

4-1 Overview

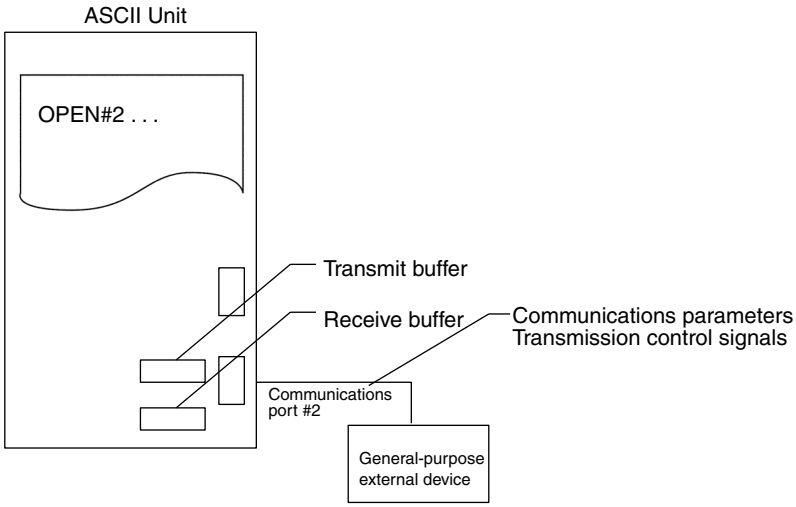
The following operations are covered in this section.

Operation		Command/function
Opening a communications port		OPEN command
Sending data		PRINT command (The LPRINT command executes *device*: LPRT.)
Receiving data	CR terminator	INPUT command (The LINE INPUT command receives everything in one character variable).
	Specified character length	INPUT$ function
Closing a communications port		CLOSE command
Communications error processing		ON ERROR command, ERC command, ERR command, and RESUME command

4-2 Opening a Communications Port

4-2-1 OPEN# Command

A communications port is opened using the OPEN# command. The OPEN# command specifies the device (peripheral device name), communications parameters for the communications port, and the use of transmission control signals.



Example

OPEN #2, "COMU: 9600, 8, N, 2, CTS_XX, RTS_XX, DSR_XX, XN_XX"

- CTS_XX, RTS_XX, DSR_XX, XN_XX: Operation of transmission control signals
- 9600, 8, N, 2: Communications parameters for communications port
- COMU: Device symbol (peripheral device name)

4-2-2 OPEN Command Specifications

The following details are specified in the OPEN command.

Item		Settings	Detail
Device symbol (peripheral device name)		SCRN, LPRT, EKPRT, NKPRT, KYBD, COMU, TERM	• Use of the transmission/receive buffer. • Control of the transmission control signals.
Baud rate		300/600/1,200/2,400/4,800/9,600/19,200/38,400 bps (default: 9,600 bps) (port 3 ≤ 19,200 bps)	
Data length		7 or 8 bits (default: 8 bits)	
Parity		None, odd, or even (default: none)	
Stop bits		1 or 2 bits (default: 2 bits).	
RS-232C port transmission controls	CTS	CTS_ON	Executes RTS/CTS flow control for the receive buffer of the external device. The ASCII Unit monitors the CTS signal during execution of the PRINT command. When the CTS signal is OFF, data transmission is suspended. When the CTS signal is ON, data transmission continues.
		CTS_OFF	RTS/CTS flow control is not executed. The ASCII Unit does not monitor the CTS signal during execution of the PRINT command.
	RTS	RTS_ON	Turns ON the RTS signal from the opening of the communications port until it closes. When receiving data normally from the external device into the receive buffer, RTS must be set to RTS_ON.
		RTS_OFF	Turns ON the RTS signal only during the execution of I/O commands (PRINT and INPUT), turning it OFF at all other times. BASIC's RTS control is disabled.
RS-232C port transmission controls	DSR	DSR_ON	When output commands (such as PRINT or LPRINT) are being executed, the ASCII Unit checks the DSR signal, and sends data if the DSR signal is ON. If the DSR signal is OFF, the ASCII Unit waits to transmit until the DSR signal turns ON. The BASIC indicator will flash slowly while the Unit is waiting to transmit.
		DSR_OFF	The ASCII Unit does not check the DSR signal when output commands are being executed.
RS-422A/485 port transmission controls		CTS, RTS, DSR not used (invalid)	---
Xon/Xoff (Not available for port 2 of ASC21.)		XN_ON	Controls Xon/Xoff flow to both the ASCII Unit's receive buffer and the external device's receive buffer. • ASCII Unit's Receive Buffer When the receive buffer of the ASCII Unit reaches 3/4th full (384 bytes), it sends an Xoff signal, requesting the external device to suspend data transmission. Once the Unit's receive buffer drops to 1/4th full (128 bytes), it sends an Xon signal, requesting the external device to resume data transmission. • External Device's Receive Buffer When the ASCII Unit receives an Xoff signal from the external device during execution of a PRINT command, it suspends data transmission. When it receives an Xon signal from the external device, it resumes data transmission.
		XN_OFF	Xon/Xoff flow control disabled.

4-2-3 Control/Monitor Commands for Transmission Control Signals

When control of the transmission control signals is not specified in the OPEN command (i.e., when they are set to OFF), they can be controlled and monitored using the following commands.

Action	Command	Example	Conditions
Turn RTS signal ON/OFF	RTS	RTS #1_SET	Only possible when RTS_ON is specified in the OPEN command.
Turn DTR signal ON/OFF	DTR	DTR #1_SET	Possible when the port is open.
Monitor the DSR signal	DSR	IF DSR(1) THEN ...	None
Monitor the CTS signal	CTS	IF CTS(1) THEN ...	None

4-2-4 Device Symbol

The following details are specified by the device symbol (peripheral device name).

- I/O specifications (use of the transmission/receive buffer)
- Control of transmission control signals
- Output code for the PRINT command

Note For details on the output codes for the PRINT command, refer to *4-3-1 PRINT# Command*.

Device symbol	Peripheral device	I/O (send/receive)		Transmission control signals					
				Communications port open				I/O command execution (PRINT or INPUT)	
				RTS		DTR		RTS	DTR
		Reception by ASCII Unit (using receive buffer)	Transmission by ASCII Unit (using send buffer)	RTS_OFF in OPEN command	RTS_ON in OPEN command	Port 1/ terminal port	Port 2	RTS_ON and RTS_OFF in OPEN command	
COMU	General-purpose communication device	Yes	Yes	OFF	ON	ON	OFF	ON	No change
TERM	Terminal	Yes	Yes	OFF	ON	ON	OFF	ON	No change
SCRN	Display	None	Yes	OFF	ON	OFF	OFF	ON	No change
LPRT (See note.)	Printer	None	Yes	OFF	ON	OFF	OFF	ON	No change
NKPRT (See note.)	Printer	None	Yes	OFF	ON	OFF	OFF	ON	No change
EKPRT (See note.)	Printer	None	Yes	OFF	ON	OFF	OFF	ON	No change
KYBD	Keyboard	Yes	None	OFF	ON	ON	OFF	ON	No change

Note The differences between LPRT, EKPRT and NKPRT are shown in the following table.

Device symbol	External device	Details
LPRT	When not outputting Chinese-derived characters (Kanji)	No conversion from Shift JIS code to JIS code.
NKPRT	When outputting Chinese-derived characters, the printer control code is “PC-PR” printer	Shift JIS code is converted to JIS code and KI (Kanji In): [ESC]K (1B4B Hex) KO (Kanji Out): [ESC]H (1B48 Hex) are added.
EKPRT	When outputting Chinese-derived characters, the printer control code is “ESC/P” printer	Shift JIS code is converted to JIS code and KI (Kanji In): [FS]&(1C26 Hex) KO (Kanji Out): [FS]. (1C2E Hex) are added.

Note When the Super ESC/P printer control code is used, both NKPRT and EKPRT are supported.

PC-PR: Printer control code used by NEC’s PC-PR xx Series.

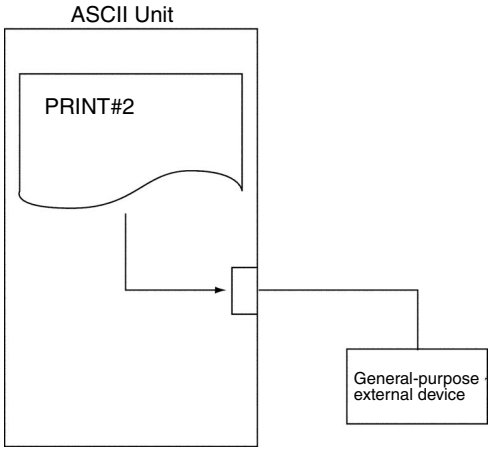
ESC/P: Printer control code supported by EPSON. Used in most printers.

Super ESC/P: Printer control code that automatically switches between ESC/P and PC-PR.

4-3 Sending Data

4-3-1 PRINT# Command

The PRINT# command sends data through the designated communication port. For example, it can send command frames originally stored in variables, print variables to the printer, and display data from variables on the screen.



Note The following apply when the device symbol is TERM, LPRT, or SCRN:

- If delimited with commas, output is divided into 9-character fields.
- If delimited with semi-colons or spaces, output is continuous.
- If there is no semi-colon at the end, then a new line is inserted after the transmission has ended.
- In a numerical format, one space is placed at the start and finish. (The first space is for the minus sign.)

4-3-2 DMA Data Transmission

It is possible to transmit data using DMA (Direct Memory Access) depending on the model of ASCII Unit and the communications port used.

Model	C200H-ASC11		C200H-ASC21		C200H-ASC31		
Port	Port #1	Port #2	Port #1	Port #2	Port #1	Port #2	Port #3
DMA transmission (See note.)	NO	YES	NO	YES	YES	YES	NO

Note At ports that are capable of DMA transmission, DMA transmission is enabled or disabled in the DM Setup Area according to the following table.

Bits	Name	Setting (hex)
m + 2 bits 08 to 15	Port #1 DMA transmission (C200H-ASC31 only)	00: DMA disabled (default) 5A: DMA enabled
m + 3 bits 08 to 15	Port #2 DMA transmission (all models)	00: DMA disabled (default) 5A: DMA enabled

For communications ports that do not transmit DMA data, the data is sent byte by byte from memory to the communications port by the PRINT command. This means that until the transmission is completed, the BASIC program is stopped. For ports that do transmit DMA data, the PRINT command simply provides the trigger, and the DMA controller processes the data transmission. This allows the BASIC program to proceed to execute the next line, speeding up processing. Therefore, use DMA transmission to speed up processing.

If DMA data transmission is enabled and the OPEN command is executed, the RTS/CTS flow control and Xon/Xoff flow control settings will be enabled. If CTS or Xon/Xoff flow control is enabled, DMA will be disabled. When flow control settings are enabled and DMA data transmission is used, the transmission must be completed using PRINT command.

The amount of DMA data that can be sent at one time is 255 bytes max.

To transmit data that is more than 255 bytes, write a program as follows:

Example
```
A$ = ...........: 200 bytes
B$ = ...........: 100 bytes
PRINT A$;B$
```

This example will transmits 300 bytes of data.

Note DMA data transmission is a method of sending data directly to a communications port without using the MPU. Because this does not involve the MPU, the ASCII Unit is able to continue executing the BASIC program as data is being transmitted. Transmission of data is carried out by a dedicated circuit (the DMA controller).

4-3-3 Transmission Control Signals

If the RTS signal is OFF when the PRINT# command is executed, RTS is turned ON, and the processing will proceed as follows:

For OPEN Command with CTS_ON, DSR_ON

If CTS and DSR are both ON, data will be transmitted. If DSR is OFF when transmission starts, the BASIC indicator will flash slowly and the Unit will wait to transmit data. If CTS turns OFF during data transmission, data transmission will stop and wait, and the BASIC indicator will flash slowly.

For OPEN Command with CTS_ON, DSR_OFF

If CTS is ON, data will be transmitted. If CTS is OFF, data will not be transmitted and the Unit will wait for CTS to turn ON; the BASIC indicator will flash slowly. If CTS turns OFF during data transmission, data transmission will stop and wait.

For OPEN Command with CTS_OFF, DSR_ON

If DSR is ON at the start of transmission, data will be transmitted. If DSR is OFF, data will not be transmitted, the BASIC indicator will flash slowly, and the Unit will wait for DSR to turn ON.

For OPEN Command with CTS_OFF, DSR_OFF

Data is transmitted irrespective of CTS and DSR.

Output Status by Device Symbol in the OPEN Command

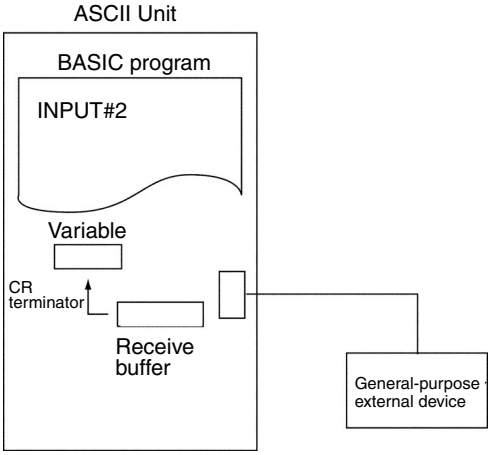
Hex	Code	Meaning	Communications port 1			Communications port 2 (port 1 for ASC31)			
			TERM/SCRN	LPRT/ NKPRT/ EKPRT	COMU	TERM	SCRN	LPRT/ NKPRT/ EKPRT	COMU
00	NUL	Space	Output	Output	Output	Output	Not sup-ported	Output	Output
01	SOH	Start of heading							
02	STX	Start text							
03	ETX	End text							
04	EOT	End of transmission							
05	ENQ	Enquiry							
06	ACK	Acknowledgment							
07	BEL	Bell							
08	BS	Backspace							
09	HT	Horizontal tab							
0A	LF	Return (line field)							
0B	VT	Vertical tab							
0C	FF	Form forward							
0D	CR	Carriage return							
0E	SO	Shift out							
0F	SI	Shift in							
10	DLE	Device line expansion							
11	XON	Device control 1 / Xon							
12	DC2	Device control 2							
13	XOFF	Device control 3 / Xoff							
14	DC4	Device control 4							
15	NAK	Negative acknowledgment							
16	SYN	Sync signal							
17	ETB	End transmission block							
18	CAN	Cancel							
19	EM	End medium							
1A	SUB	Substitute character							
1B	ESC	Expansion							
1C	IS4	File isolation character							
1D	IS3	Group isolation character							
1E	IS2	Record isolation character							
1F	IS1	Unit isolation character							

4-4 Receiving Data

Data is received using the INPUT# command or the INPUT$ (<numeral expression>[,#<port expression>])) function.

4-4-1 INPUT# Command

Use the INPUT# command to store data received from the receive buffer in variables using the CR code as a terminator. This allows reception of CR-delimited communications responses and input data from the keyboard. The BASIC indicator on the front panel will flash slowly until the CR code is entered. If data is not received within a certain time, INPUT# can be quit by using this command after the WAIT command. See *5-4 Time Processing*.



Example INPUT #2, A$
Data from communications port #2's receive buffer up until the CR code terminator is stored in character variable A$.

Example INPUT #2, A$, B$
Data from communications port #2's receive buffer delimited by commas is stored in character variables A$ and B$.

Note CR or LF is not received as data for the INPUT # command.

4-4-2 Transmission Control Signals

If the RTS signal is ON, received data will be stored in the receive buffer independent of the INPUT command. When the INPUT command is executed, the data in the receive buffer will be stored in each variable, using the CR code as the terminator.

RTS will turn ON if the INPUT command is executed when the RTS signal is OFF. If data is received, it will be stored in the receive buffer, and then stored in each variable using the CR code as the terminator. (CTS and DSR are not checked.)

If the receive buffer overflows, the allocated I/O area ports (1, 2, and terminal port) will turn ON error flags and indicators.

Note
1. Use Xon/Xoff flow control to ensure that the ASCII Unit receive buffer does not overflow.
2. Choose one of the following flow controls according to the external devices to ensure that the receive buffers of the external devices do not overflow.

- Use CTS_ON with the OPEN command and monitor the CTS signal for RTS/CTS flow control.
- Use XN_ON with the OPEN command and perform Xon/Xoff flow control.

3. When an ON COM interrupt is used, the baud rate of all ports will be as follows:

   Port 1 + port 2 ≤ 38.4 Kbps

   If the baud rate is set to exceed this condition, a PORT CONFIGURATION ERROR (error code 0055) will be generated when compiling.

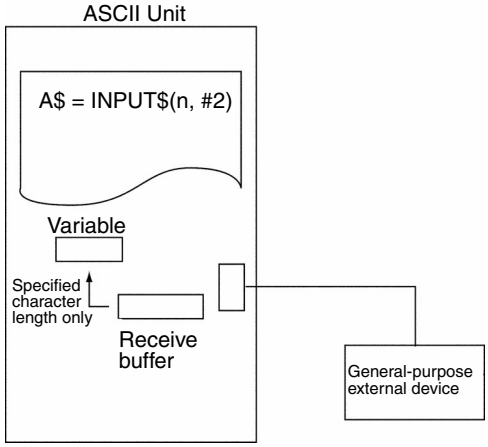
4-4-3 INPUT$ Function

The INPUT$(*character_length*, *port*) function is used to store a fixed number of characters from the receive buffer in one string variable.

This function enables read communications data with terminators other than CR, specified numbers of bar code characters, a specified number of bytes from the receive buffer, etc. The INPUT$(*character_length*, *port*) function can be used in conjunction with the ON COM command (communications interrupt) or ON KEY command (key interrupt), so that when a communications interrupt or key interrupt occurs, it is possible to store a specified number of characters from the receive buffer to a variable.

Until the specified number of characters is received, the BASIC indicator on the front panel of the Unit will flash slowly, signifying that it is awaiting input.

If the specified number of characters is not received within a certain time, INPUT$ can be quit by using this function after the WAIT command. See *5-4 Time Processing*.



Example

A$ = INPUT$(10, #2)
Ten characters of data is stored from communications port #2 to A$.

Example

Reception of Communications Response Frame

When a terminator other than CR is used, the communication data is received using the INPUT$ function.

A$ = INPUT$(LOC(2), #2)
All data in the communications receive buffer is stored in character variable A$.

Note 1. If 256 bytes or more of data is stored in the receive buffer, or the receive buffer if empty, and the data is retrieved using A$ = INPUT$(LOC(2), #2), an ILLEGAL FUNCTION CALL ERROR will be generated. Therefore, use the following program to retrieve the data.

```
I% = 0
A% = LOC(2)
WHILE (A% <> 0)
        IF (A%>255) THEN A% = 255
        A$(I%) = INPUT$ (A%, #2)
        I% = I%+1
        A% = LOC(2)
WEND
```

4-5 Closing a Communications Port

Communications ports are closed using the CLOSE command.

When a transmission device has been specified using the OPEN command and the CLOSE command is executed, the data remaining in the transmit buffer is sent before the port is closed.

When a reception device has been specified using the OPEN command and the CLOSE command is executed, the data remaining in the receive buffer is cleared.

All communications ports are also closed by the END, CLEAR, and NEW commands. They are not closed by the STOP command.

4-5-1 Transmission Control Signals

Model	C200H-ASC11		C200H-ASC21		C200H-ASC31		
Port	Port #1	Port #2	Port #1	Port #2	Port #1	Port #2	Port #3
RTS	OFF			---	OFF		
DTR	ON	No change	ON	---	No change		ON

4-5-2 Transmission Control Signal Timing Charts

After turning ON the power or restarting the Unit, and until each port is opened using the OPEN# command, the device symbol (peripheral device name) for each port is as follows:

Model	Port	Device symbol (peripheral device name)
C200H-ASC11	Port #1	TERM
C200H-ASC21	Port #2	LPRT
C200H-ASC31	Port #1	LPRT
	Port #2	LPRT
	Port #3	TERM

Transmission Control Signal Status

The statuses of the transmission control signals (RTS, DTR, DSR, and CTS) at each point of time are changed as shown in the following table according to the device symbol (peripheral device name) and transmission control signal operation specified in the OPEN# command.

RTS Signal

Device symbol (peripheral device name)	Transmission control signal operation setting	Executing RUN	Ports opened by OPEN# command	Receiving data with INPUT# command or INPUT$ function	Sending data with PRINT# command	Ports closed
Port #1: TERM, COMU, or KYBD (Port #3 for ASC31)	RTS_ON	ON to OFF	OFF to ON	ON	ON	ON to OFF
	RTS_OFF	ON to OFF	OFF	Switches from OFF to ON, then back OFF again after receiving data.	Switches from OFF to ON. (For CTS_ON and DSR_ON, checks CTS and DSR, and if they are both ON, turns RTS OFF after sending data)	OFF
Port #1: KYBD (Port #3 for ASC31)	RTS_ON	ON to OFF	OFF to ON	ON	---	ON to OFF
	RTS_OFF	ON to OFF	OFF	Switches from OFF to ON, then back OFF again after receiving data.	---	OFF
Port #2: COMU (Ports #1 and #2 for ASC31)	RTS_ON	OFF	OFF to ON	ON	ON	ON to OFF
	RTS_OFF	OFF	OFF	Switches from OFF to ON, then back OFF again after receiving data.	Switches from OFF to ON. (For CTS_ON and DSR_ON, checks CTS and DSR, and if they are both ON, turns RTS OFF after sending data)	OFF
Port #2: KYBD (Ports #1 and #2 for ASC31)	RTS_ON	ON to OFF	OFF to ON	ON	---	ON to OFF
	RTS_OFF	ON to OFF	OFF	Switches from OFF to ON, then back OFF again after receiving data.	---	OFF
Port #1: SCRN and LPRT (Port #3 for ASC31)	RTS_ON	ON to OFF	OFF to ON	---	ON	ON to OFF
	RTS_OFF	ON to OFF	OFF	---	Switches from OFF to ON. (For CTS_ON and DSR_ON, checks CTS and DSR, and if they are both ON, turns RTS OFF after sending data)	OFF
Port #2: SCRN and LPRT (Ports #1 and #2 for ASC31)	RTS_ON	OFF	OFF to ON	---	ON	ON to OFF
	RTS_OFF	OFF	OFF	---	Switches from OFF to ON. (For CTS_ON and DSR_ON, checks CTS and DSR, and if they are both ON, turns RTS OFF after sending data)	OFF

DTR Signal

Device symbol (peripheral device name)	Executing RUN	Ports opened by OPEN# command	Receiving data with INPUT# command or INPUT$ function	Sending data with PRINT# command	Ports closed
Port #1: TERM and COMU (Port #3 for ASC31)	ON	ON	ON	ON	ON
Port #2: COMU (Ports #1 and #2 for ASC31)	OFF	OFF to ON	ON	ON	ON to OFF
Port #1: KYBD (Port #3 for ASC31)	ON	ON	ON	---	ON
Port #2: KYBD (Ports #1 and #2 for ASC31)	OFF	OFF to ON	ON	---	ON to OFF
Port #1: SCRN and LPRT (Port #3 for ASC31)	ON	ON to OFF	---	OFF	OFF to ON
Port #2: SCRN and LPRT (Ports #1 and #2 for ASC31)	OFF	OFF	---	OFF	OFF

CTS and DSR Signals

CS is monitored constantly if CTS_ON is set. DSR is only checked when printing starts.

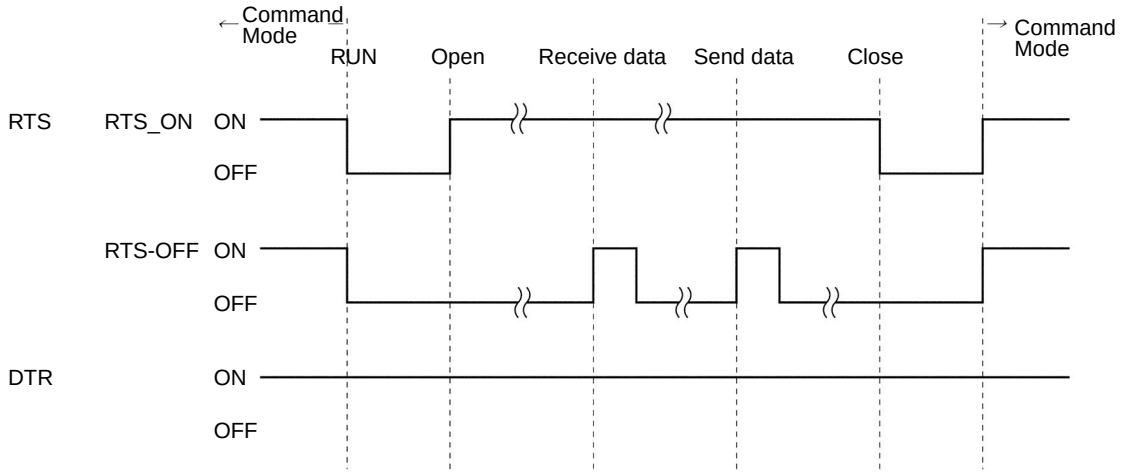
Device symbol (peripheral device name)	Transmission control signal operation setting	Executing RUN	Ports opened by OPEN# command	Receiving data with INPUT# command or INPUT$ function	Sending data with PRINT# command	Ports closed
Port #1: TERM, COMU, SCRN, and LPRT (Port #3 for ASC31) Port #2: COMU, SCRN, and LPRT (Ports #1 and #2 for ASC31)	CTS_ON	No check	No check	No check	If CTS is OFF, data transmission is stopped. (See note 1.)	No check
	CTS_OFF	No check	No check	No check	No check	No check
	DSR_ON	No check	No check	No check	If DSR is ON, data is transmitted. (See note 2.)	No check
	DSR_OFF	No check	No check	No check	No check	No check

Note

1. The CTS signal is checked when data is being transmitted by the PRINT command if CTS_ON is used. If CTS is OFF, data transmission is stopped. If CTS is ON, transmission will resume (RTS/CTS flow control performed at the receive buffer of the peripheral devices).
2. The DSR signal is checked when the PRINT command is started if DSR_ON is used. If DSR is OFF, the Unit will wait to send, and if DSR is ON, data will be transmitted.

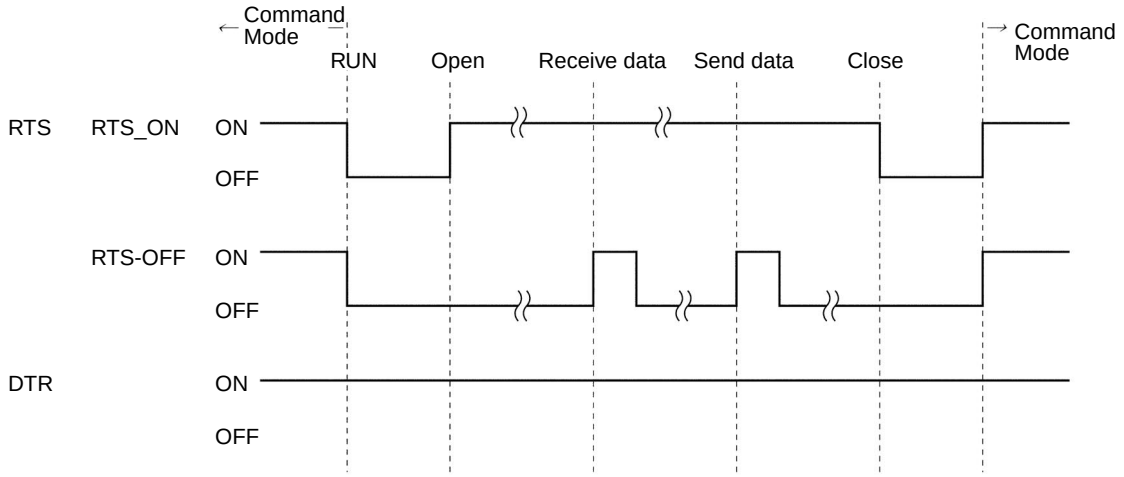
Timing Charts

Device Symbol: TERM
Port #1/Port #3 (ASC31): TERM



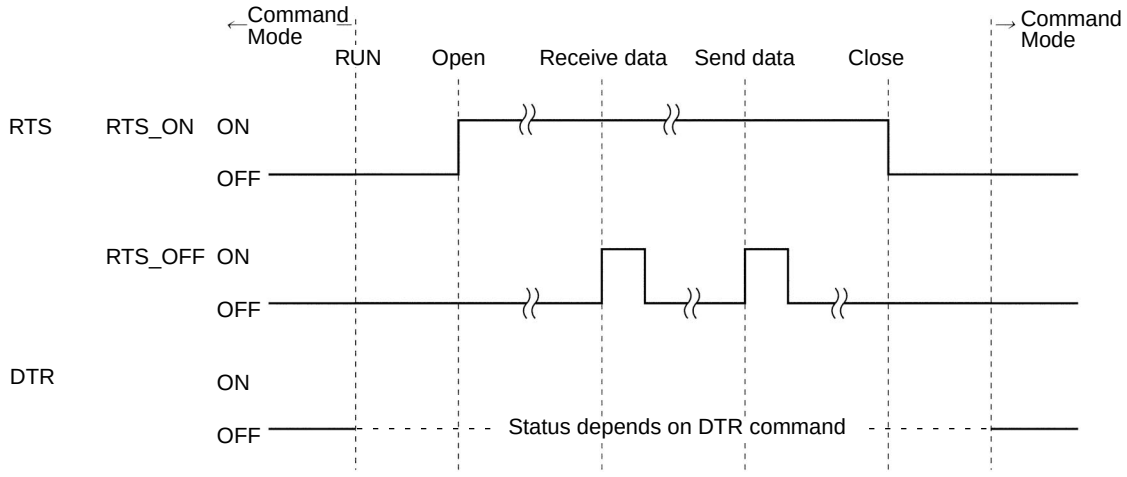
CTS	CTS_ON	No check	No check	No check	Check (during send)	No check	No check
	CTS_OFF	No check	No check	No check	No check	No check	No check
DSR	DSR_ON	No check	No check	No check	Check	No check	No check
	DSR_OFF	No check	No check	No check	No check	No check	No check

Device Symbol: COMU
Port #1: COMU (ASC11, ASC21)



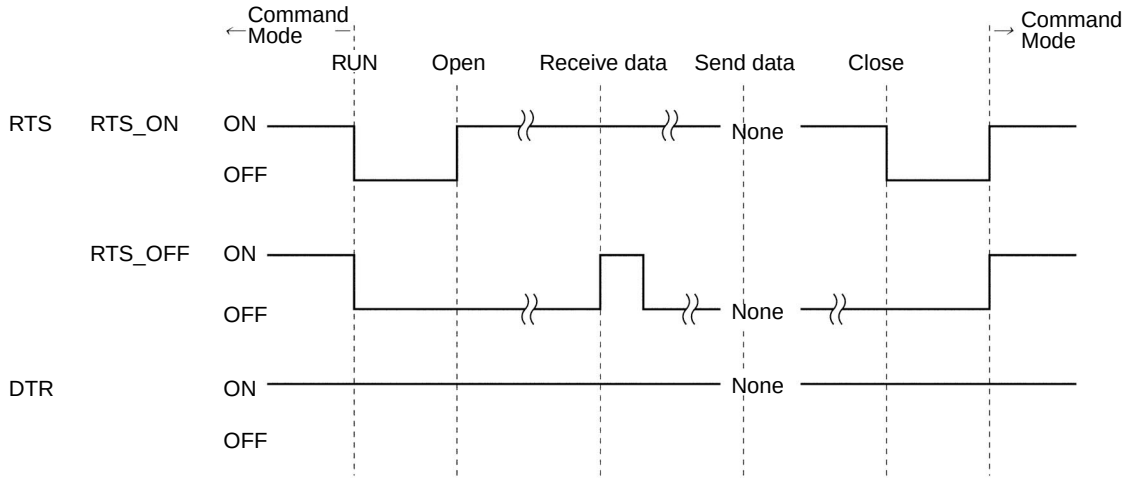
CTS	CTS_ON	No check	No check	No check	Check (during send)	No check	No check
	CTS_OFF	No check	No check	No check	No check	No check	No check
DSR	DSR_ON	No check	No check	No check	Check	No check	No check
	DSR_OFF	No check	No check	No check	No check	No check	No check

Port #2: COMU or
Port #1: COMU for ASC31



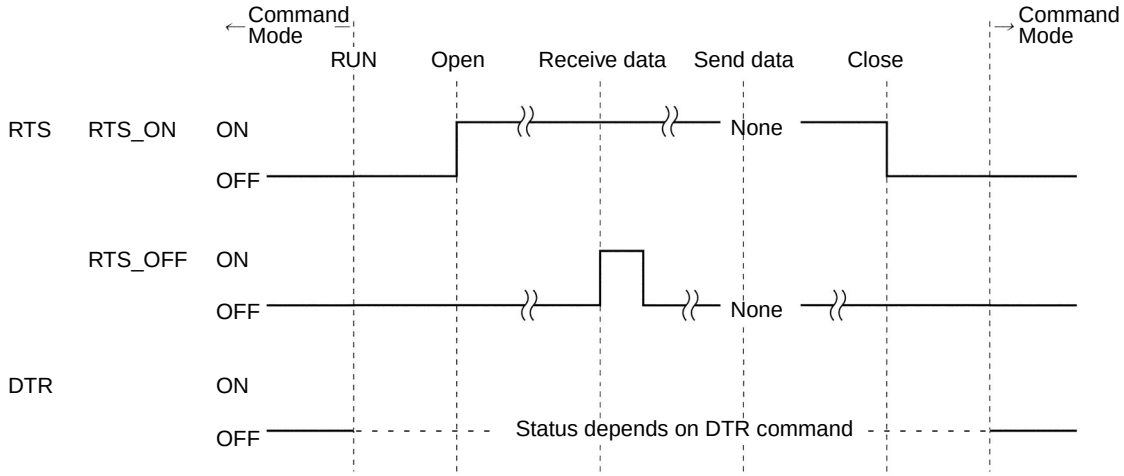
CTS	CTS_ON	No check	No check	No check	Check (during send)	No check	No check
	CTS_OFF	No check	No check	No check	No check	No check	No check
DSR	CTS_ON	No check	No check	No check	Check	No check	No check
	CTS_OFF	No check	No check	No check	No check	No check	No check

Device Symbol: KYBD
Port #1: KYBD (ASC11 or ASC21)



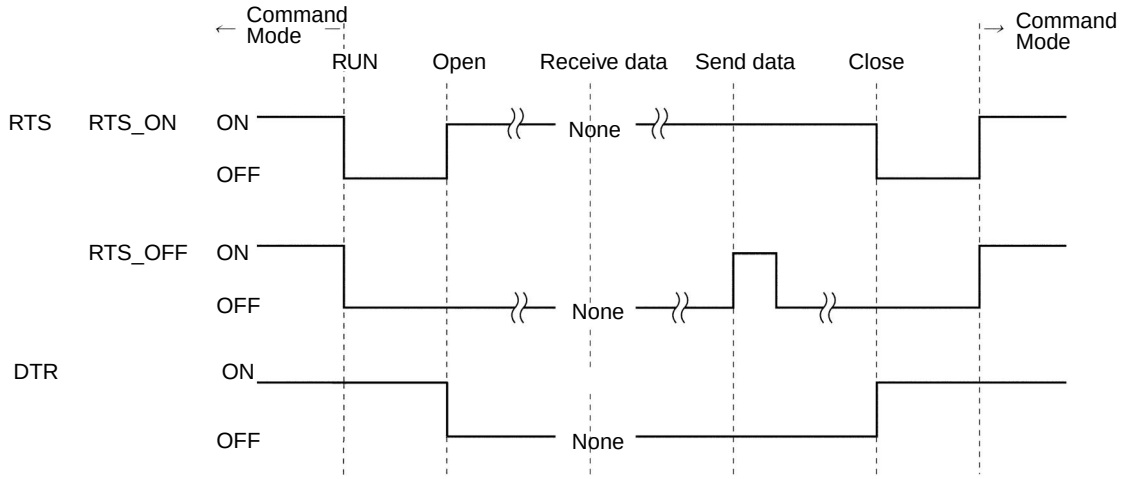
CTS	CTS_ON	No check	No check	No check		No check	No check
	CTS_OFF	No check	No check	No check		No check	No check
DSR	DSR_ON	No check	No check	No check		No check	No check
	DSR_OFF	No check	No check	No check		No check	No check

Port #2: KYBD



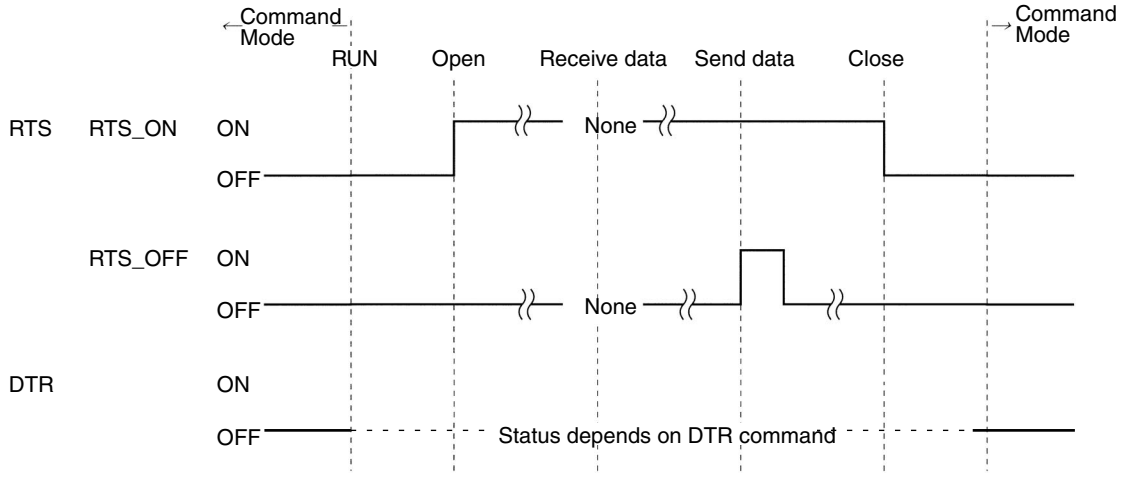
CTS	CTS_ON	No check	No check	No check		No check	No check
	CTS_OFF	No check	No check	No check		No check	No check
DSR	DSR_ON	No check	No check	No check		No check	No check
	DSR_OFF	No check	No check	No check		No check	No check

Device Symbol: SCRN or LPRT
Port #1: SCRN or LPRT



CTS	CTS_ON	No check	No check		Check (during send)	No check	No check
	CTS_OFF	No check	No check		No check	No check	No check
DSR	DSR_ON	No check	No check		Check	No check	No check
	DSR_OFF	No check	No check		No check	No check	No check

Port #2: SCRN or LPRT



CTS	CTS_ON		No check	No check	Check (during send)	No check	No check
	CTS_OFF		No check	No check	No check	No check	No check
DSR	DSR_ON		No check	No check	Check	No check	No check
	DSR_OFF		No check	No check	No check	No check	No check

4-6 Communications Errors

When a communications error occurs while a BASIC program is being executed, the ERR1 or ERR2 indicator on the ASCII Unit will be lit and at the same time the CPU Unit's port 1/2 error flag (bit 04/05 of word n+5) will be turned ON (be set to 1). Execution of the BASIC program will continue. If the ON ERROR command has been used in the BASIC program, the program will be diverted to the specified interrupt subroutine when a communications error occurs.

If another error occurs during the execution of the error interrupt subroutine however, the BASIC program will stop executing (regardless of whether the error affects the error subroutine). If the following situation is likely, program execution may be stopped, so be careful when using the ON ERROR command. Alternatively, do not use the ON ERROR command in a situation like the one below, where execution of the BASIC program should be continued even if a communications error occurs.

- Other errors occur during an interrupt subroutine due to communications errors.

Note ON ERROR Command

The ON ERROR command is used for processing with other errors, so it will be necessary to use the IF-type statements for processing. When error processing is complete, use the RESUME command to return to the origin of the diversion and continue processing. Error status can be cleared after error processing is complete by using the ERC command.

The ERR function can be used to determine the type of error that occurred. Since communications errors can be detected from the BASIC program, retry processing is easily performed from the BASIC program.

ON ERROR GOTO *EINT

Main program

*EINT

Error processing

SECTION 5
Processing

This section provides examples of processing character strings, bits, receive buffers, looping, and interrupts.

5-1 Processing Character Strings 62
5-2 Processing Bits 62
5-3 Processing Receive Buffers. 62
5-4 Time Processing 63
5-5 Interrupt Functions 64
5-6 Loop Processing 67

5-1 Processing Character Strings

Process	Programming
Retrieving n characters from beginning of A$	B$ = MID$(A$, 1, n) or B$ = LEFT$(A$, n)
Retrieving n characters from character number m	B$ = MID$(A$, m, n)
Retrieving n characters from end of A$	B$ = RIGHT$(A$, n)
Searching for character string from character number m in A$ and getting position of first character	B = INSTR(m, A$, *character_string*)
Getting number of characters in A$	B = LEN(A$)
Converting character string A$ to numeric value	B = VAL(A$)
Converting numeric value B to character string A$	A$ = STR$(B)

Example

The following code will find the position of "STX" in response data R$, retrieve the characters from there to "ETX" and store them in A$.

```
100   STX$ = CHR$(&H02)        Generate STX.
110   ETX$ = CHR$(&H03)        Generate ETX.
120   A = INSTR(1, R$, STX$)   Find the position of STX$ in R$.
130   B = INSTR(1, R$, ETX$)   Find the position of ETX$ in R$.
140   A$ = MID$(R$, A, B–A+1)  Retrieve the character string from STX$ to
                               ETX$ in R$ and store it in A$.
```

Example

If the two characters after the fourth character from the start of response data A$ are 00, The following line will jump to line 1000 as a correct response.

```
150   IF MID$(A$,4,2)="00" THEN 1000
```

5-2 Processing Bits

Process	Programming
Turning ON/OFF 1 bit of a variable.	BITON A%, 3 (Turns ON bit 3 of variable A%.)
	BITOFF A%, 3 (Turns OFF bit 3 of variable A%.)

Note

1. One bit will be set without changing the other bits.
2. When transferring bit data to the PC, turn ON/OFF the required bit with these commands and then use the PC PUT command.

5-3 Processing Receive Buffers

Process	Programming
Obtaining the number of characters in the receive buffer	A = LOC(1) (Stores the number of characters held in the communications port #1 receive buffer in A.)
Executing a process if the number of characters in a receive buffer is not 0 (i.e.,if it is not empty)	100 A = LOC(1) 110 IF A<>0 THEN 200 200 Process A

5-4 Time Processing

Process		Program
Storing a time in T$ (HH:MM:SS)		T$ = TIME$
Storing hours in H$ (HH)		H$ = LEFT$(TIME$, 2) or H$ = MID$(TIME$, 1, 2)
Storing minutes and seconds in MS$ (MM:SS)		MS$ = MID$(TIME$, 4, 5)
Executing process A at fixed intervals	Using TIME$ interrupt, interrupt subroutine is executed every minute (HH:MM:00) when the seconds equal zero.	100 ON TIME$ "**:**:00" GOSUB *A 110 TIME$ ON 500 *A Process A
	Using TIMER interrupt, interrupt subroutine executes process A after every sixty seconds.	100 ON TIMER 600 GOSUB 500 110 TIMER ON 500 Process A
Standby processing	If response is not received within three seconds after COMMAND (C$) transmission, process A is executed.	100 PRINT #1, C$ 110 WAIT "3.0", 200 120 A$ = INPUT$(1, #1) 200 Process A
	If data is not completely transferred from CPU Unit within ten seconds after execution of PC READ, process A is executed.	100 WAIT "10.0", 200 110 PC READ . . . 200 Process A

Note If a BASIC interrupt occurs during the wait state caused by the WAIT command, the wait monitor timer will be suspended throughout the time of the interrupt.

5-5 Interrupt Functions

Interrupt	Details	Program	Enable/disable/ stop commands (See note 2.)
Communications interrupts	The interrupt subroutine is executed when data is received in the receive buffer. All data received is stored in the receive buffer	ON COM n GOSUB n: 1 or 2	COM n (ON/OFF/STOP) n: 1 or 2
	The interrupt subroutine is executed when a specified character is received in the receive buffer. (Data beyond the specified character is also received in the buffer.) (See note 1.)	ON COM n GOSUB, CODE = XX n: 1 or 2	
	The interrupt subroutine is executed when a specified number of bytes is received in the receive buffer. (Data beyond the specified number of bytes is also received in the buffer.) (See note 1.)	ON COM n GOSUB, BYTE = XX n: 1 or 2	
	The interrupt subroutine is executed when a specified final character is received in the receive buffer after a specified start character. (The string of characters between the start character and final character remains in the buffer. Data after the specified final character is also received in the buffer.) (See note 1.)	ON COM n GOSUB, HEAD = XX, TERM = YY n: 1 or 2	
	The interrupt subroutine is executed when a specified number of characters is received in the receive buffer after a specified start character. (The character string from the specified start character to the specified number of characters remains in the receive buffer, and the data after the specified number of characters is also received in the receive buffer.) (See note 1.)	ON COM n GOSUB, HEAD = XX, BYTE = YY n: 1 or 2	
Key interrupts	The interrupt subroutine is executed when a certain key signal is input. (See note 3.)	ON KEY(n) GOSUB n: 0 to 9	KEY n (ON/OFF/STOP) n: 0 to 9
PC interrupts	Specified PC interrupt number interrupt subroutines are executed by interrupts from the IOWR(#CC00) instruction or IR/CIO words allocated to the ASCII Unit in the PC. (See note 4.) I/O Allocations: PC interrupts are triggered by the IR/CIO words allocated to the ASCII Unit: IR/CIO n bit 04. PC interrupt number is set in IR/CIO n + 2 bits 00 to 07. IOWR(#CC00): Triggers a PC interrupt by sending data.	ON PC GOSUB n: 01 to 99	PC n (ON/OFF/STOP) n: 01 to 99
Timer interrupts	The interrupt subroutine is executed repeatedly each time the timer times out (interval time). (Once executed, the subroutine is repeated until TIME OFF is specified.)	ON TIMER 300 GOSUB A subroutine branch to the GOSUB line number will occur 30 seconds after this command is executed; the subroutine branch is then repeated each 30 seconds. Timer: 0 to 864000 (0.1-s unit)	TIMER (ON/OFF/STOP)
On alarm interrupts	The subroutine interrupt is executed when the timer times out (one-shot timer). (No repetition after the initial execution.)	ON ALARM 300 GOSUB A subroutine branch to the GOSUB line number will occur once only, 30 seconds after this command is executed. Timer: 0 to 864000 (0.1-s unit)	ALARM (ON/OFF/STOP)
Scheduled interrupts	The interrupt subroutine is executed at a specified time.	ON TIME$ = "□:□:□" GOSUB	TIME$ (ON/OFF/STOP)
Error interrupts	The interrupt subroutine is executed when an error occurs.	ON ERROR GOTO	

Note

1. When an interrupt is generated upon receiving a specified number of characters or data up to a specified character, all data is received in the receive buffer. If the necessary number of characters is retrieved by the INPUT$ function, it is possible to use first-in/first-out (FIFO) processing to move that group of characters to the front of the receive buffer.
2. One of the ON, OFF, or STOP commands must be specified for the interrupt as shown in the column on relevant commands. The STOP command prevents branch processing even if the interrupt has occurred; it simply records the interrupt entry, and branching occurs with the subsequent ON command.
3. Interrupts set for GOTO statements will wait until the current BASIC command has been executed.
4. IOWR#CC00 interrupts are not accepted while the ASCII Busy Flag is ON.

Interrupt Priority

If more than one interrupt is generated at the same time, the interrupt subroutines will be executed in the following order of priority.

Interrupt Priority Order Table

Order of priority	Interrupt
1	ON COM 1
2	ON COM 2
3	ON KEY 0
:	:
12	ON KEY 9
13	ON PC 1
:	:
111	ON PC 99
112	ON ALARM
113	ON TIMER
114	ON TIME$

If further interrupts are generated during the execution of an interrupt subroutine, they will be processed after execution of the current interrupt subroutine has been completed, according to the interrupt priority order table shown above. ON ERROR interrupts, however, will be executed immediately, even during execution of other interrupt subroutines.

Example

If ON COM 1 and ON ALARM interrupts are generated while an ON PC1 interrupt subroutine is being executed, the ON COM1 interrupt subroutine will be executed after the ON PC1 interrupt subroutine has been completed, as ON COM 1 has priority over ON ALARM. After the ONCOM 1 interrupt subroutine has finished executing, the ON ALARM interrupt subroutine will be executed.

If a PC interrupt subroutine is generated while the ON COM 1 interrupt subroutine is being executed, the PC interrupt subroutine will take priority over the ON ALARM interrupt.

If more than one interrupt is generated simultaneously or if another interrupt is generated while a interrupt subroutine is being executed, one interrupt will be recorded if this status is either ON or STOP.

Application Example

```
200   ON COM 1 GOSUB 1000
210   COM 1 ON
```

Communications interrupt enabled

```
1000  Interrupt subroutine
      RETURN
```

The ON COM command is often used in conjunction with the INPUT$ function as shown below. (STX% = &H02, ETX% = &H03, Receive data = 0<LOC(2)<255)

Example 1

```
100  ON COM 2 GOSUB 500
110  COM 2 ON
120  GOTO 120
500  A$ = INPUT$(LOC(2),#2)
510  RETURN
```

If data is received, all the data in the receive buffer is stored in A$.

Example 2

```
100  ON COM 2 GOSUB 500, CODE = ETX%
110  COM 2 ON
120  GOTO 120
500  A$ = INPUT$(LOC(2), #2)
510  RETURN
```

If ETX% is received, all data in the receive buffer is stored in A$.

Example 3

```
100  ON COM 2 GOSUB 500, BYTE = 20
110  COM 2 ON
120  GOTO 120
500  A$ = INPUT$(20, #2)
510  RETURN
```

If 20 characters are received, 20 characters from the receive buffer are stored in A$.

Example 4

```
100  ON COM 2 GOSUB 500, HEAD = 02, TERM = 03
110  COM 2 ON
120  GOTO 120
500  A$ = INPUT$(LOC(2), #2)
510  RETURN
```

If ETX% is received from STX%, all data in the receive buffer is stored in A$.

Example 5

```
100  ON COM 2 GOSUB 500, HEAD=STX%, BYTE = 20
110  COM 2 ON
120  GOTO 120
500  A$ = INPUT$(LOC (2), #2)
510  A$ = RIGHT$(A$, 20)
520  RETURN
```

If 20 characters are received from STX%, 20 characters from the receive buffer are stored in A$.

Caution is required, because the the next data may already have been placed in the receive buffer.

5-6 Loop Processing

Process	Programming
Specified number of repetitions: If no response is received within 3 s of transmission of the command (C$), the command is resent. If there is no response after 3 repetitions, error processing will commence.	10 FOR A = 1 TO 3 20 PRINT #2, C$ 30 WAIT "3.0", 50 40 A$ = INPUT$(1, #2): GOTO 1000 50 NEXT A 60 Error processing
Repetition under set conditions: A C$ command is executed and the command/response reception are repeated as long as the 2 characters following the 4th character are 20 or less.	10 WHILE A <= 20 20 PRINT #2, C$ 30 B$ = INPUT$(LOC(2), #2) 40 A$ = MID$(B$,4,2) 50 A =VAL(A$) 100 WEND

SECTION 6
Data Exchange with the CPU Unit

This section describes procedures for exchanging data between the ASCII Unit and the CPU Unit, the transfer sequence, instruction/command timing, and the use of various interrupts.

6-1 Overview of Data Exchanges 70
6-1-1 Data Exchange Methods 70
6-1-2 Sending Interrupts 72
6-2 Selecting the Data Exchange Method 74
6-3 Details of the Data Exchange Methods 75
6-3-1 Data Exchange Method No. 1 75
6-3-2 Data Exchange Method No. 2 78
6-3-3 Data Exchange Method No. 3 79
6-3-4 Data Exchange Method No. 4 80
6-3-5 Data Exchange Method No. 5 82
6-3-6 Data Exchange Method No. 6 85
6-3-7 Data Exchange Method No. 7 86
6-4 Data Exchange Time Charts 87
6-4-1 PC READ/PC WRITE Timing 87
6-4-2 PC READ@/PC WRITE@ Timing 87
6-4-3 PC PUT/PC GET Timing 88
6-4-4 IOWR (#00□□)/IORD (#00□□) Timing 88
6-4-5 PC EPUT/PC EGET Timing 88
6-4-6 IOWR (#FD00)/IORD (#FD00) and PC QREAD/PC QWRITE Timing 89
6-4-7 Combined IOWR (#00□□), IOWR (#CC00), ON PC GOSUB, and PC EGET Timing 89
6-4-8 Combined IOWR (#CC00), IORD (#00□□), ON PC GOSUB, and PC EPUT Timing 90
6-4-9 Sending Interrupts from CPU Unit to ASCII Unit: PC Interrupts . . 90
6-5 IOWR/IORD Instruction Specifications 95
6-5-1 IOWR Instruction (SPECIAL I/O UNIT WRITE) 95
6-5-2 IORD Instruction (SPECIAL I/O UNIT READ) 102
6-5-3 Using the IOWR and IORD Instructions 106
6-6 Restrictions for CS1-series PCs 108
6-6-1 Using the Sync Time Set Bit for CS1-series PCs 109

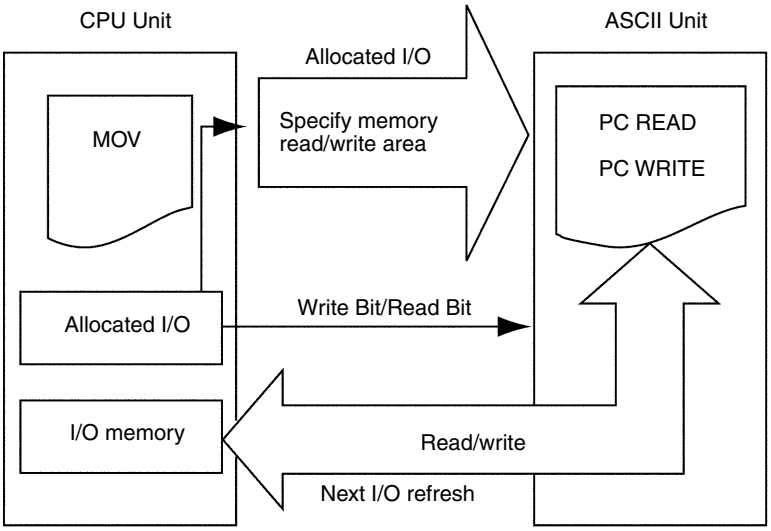
6-1 Overview of Data Exchanges

6-1-1 Data Exchange Methods

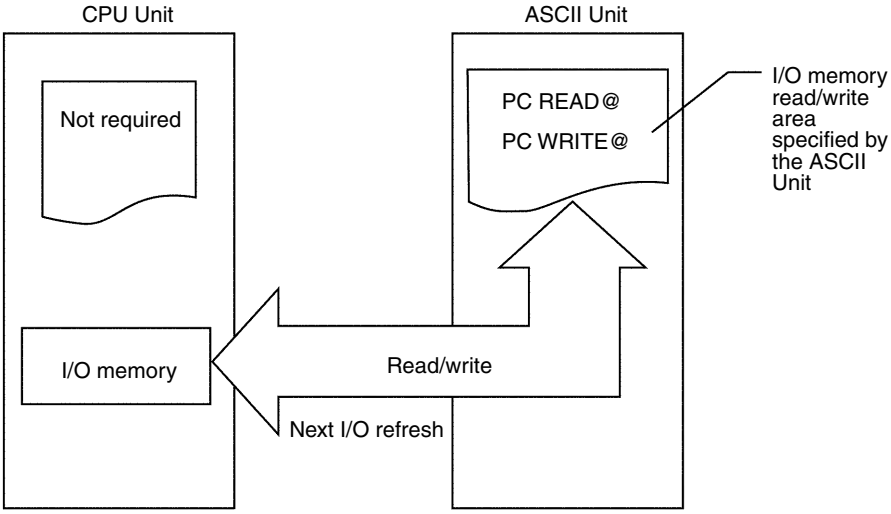
The ASCII Unit can exchange data with the CPU Unit in the following five ways.

1,2,3...

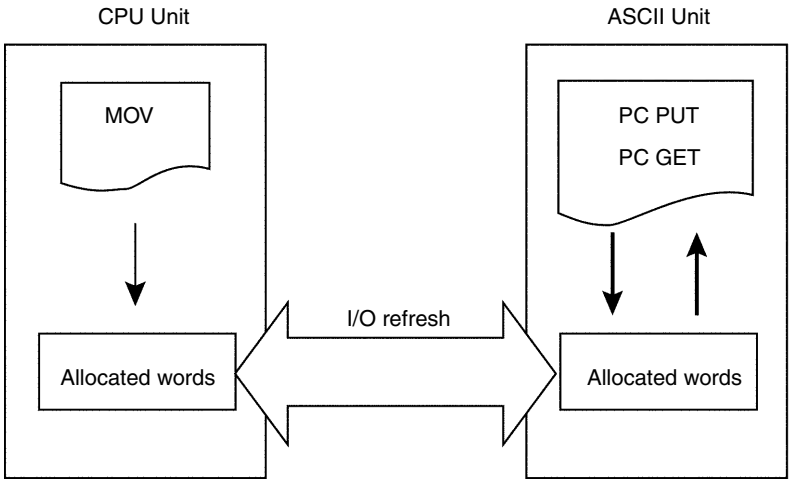
1. The CPU Unit designates its own read/write area in memory. If a PC READ/WRITE command is then specified from the CPU Unit, the ASCII Unit uses the PC READ/WRITE command to access the memory area indicated by the CPU Unit.



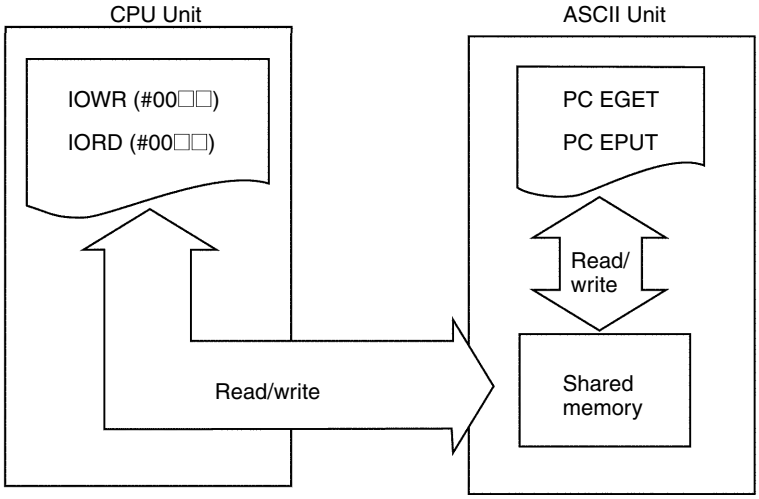
2. The ASCII Unit independently accesses the CPU Unit I/O memory. The CPU Unit's user program is not required at all.



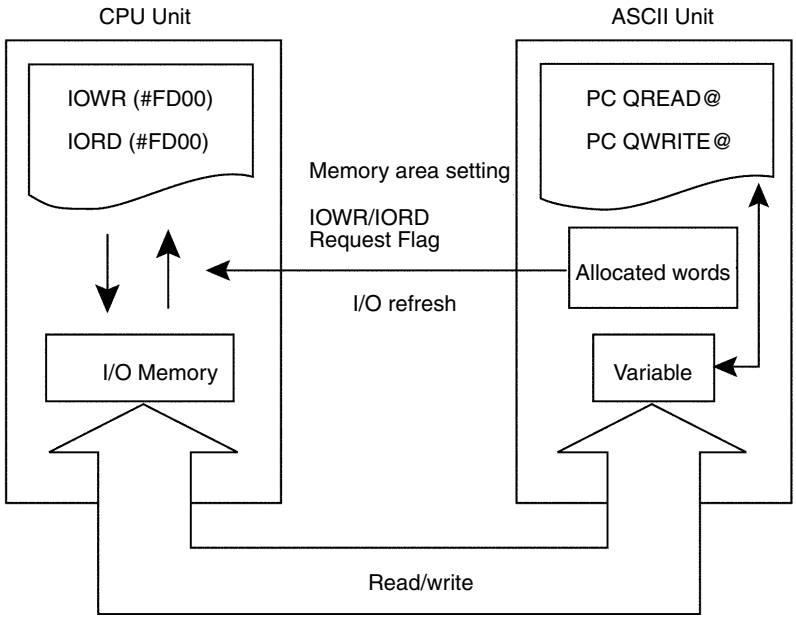
3. The CPU Unit and the ASCII Unit both access their own allocated words in memory. (The ASCII Unit accesses its own data in 16-bit units via the PC PUT/GET command.) The words allocated in the CPU Unit and memory words in the ASCII Unit are exchanged during the I/O refresh period.



4. The ASCII Unit accesses its own shared memory via a PC EGET/EPUT command. Asynchronously, the CPU Unit reads/writes the ASCII Unit's shared memory via an IORD (#00□□)/IOWR (#00□□) instruction. The IORD (#00□□)/IOWR (#00□□) instructions are supported only by the C200HX/HG/HE and CS1-series PCs.



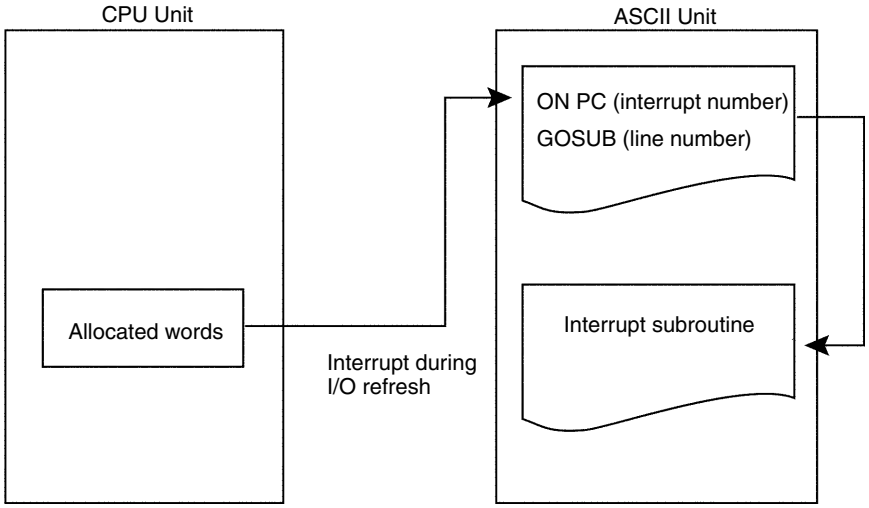
5. The ASCII Unit sets an IOWR Request Flag for the CPU Unit. When an IOWR (#FD00) instruction is executed, the CPU Unit writes data. Alternately, the ASCII Unit sets an IORD Request Flag for the CPU Unit. When an IORD (#FD00) instruction is executed, the CPU Unit reads data (supported only by the C200HX/HG/HE and CS1-series PCs).



6-1-2 Sending Interrupts

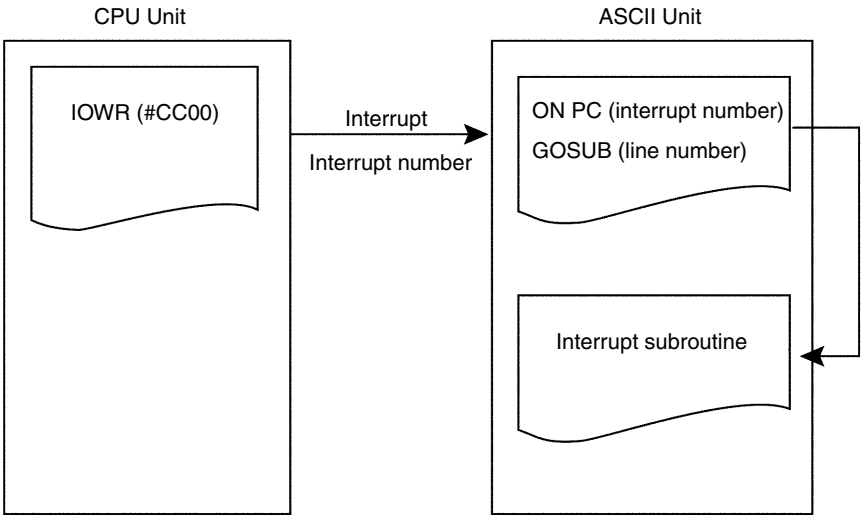
There are two methods for sending interrupts from the CPU Unit to the ASCII Unit. It is not possible to send interrupts to the CPU Unit from the ASCII Unit.

1,2,3... 1. Interrupts are sent during the I/O refresh period. An interrupt number is set in the allocated IR/CIO area word, and when the PC Interrupt Bit goes ON, the ON PC command branches to the specified line number.



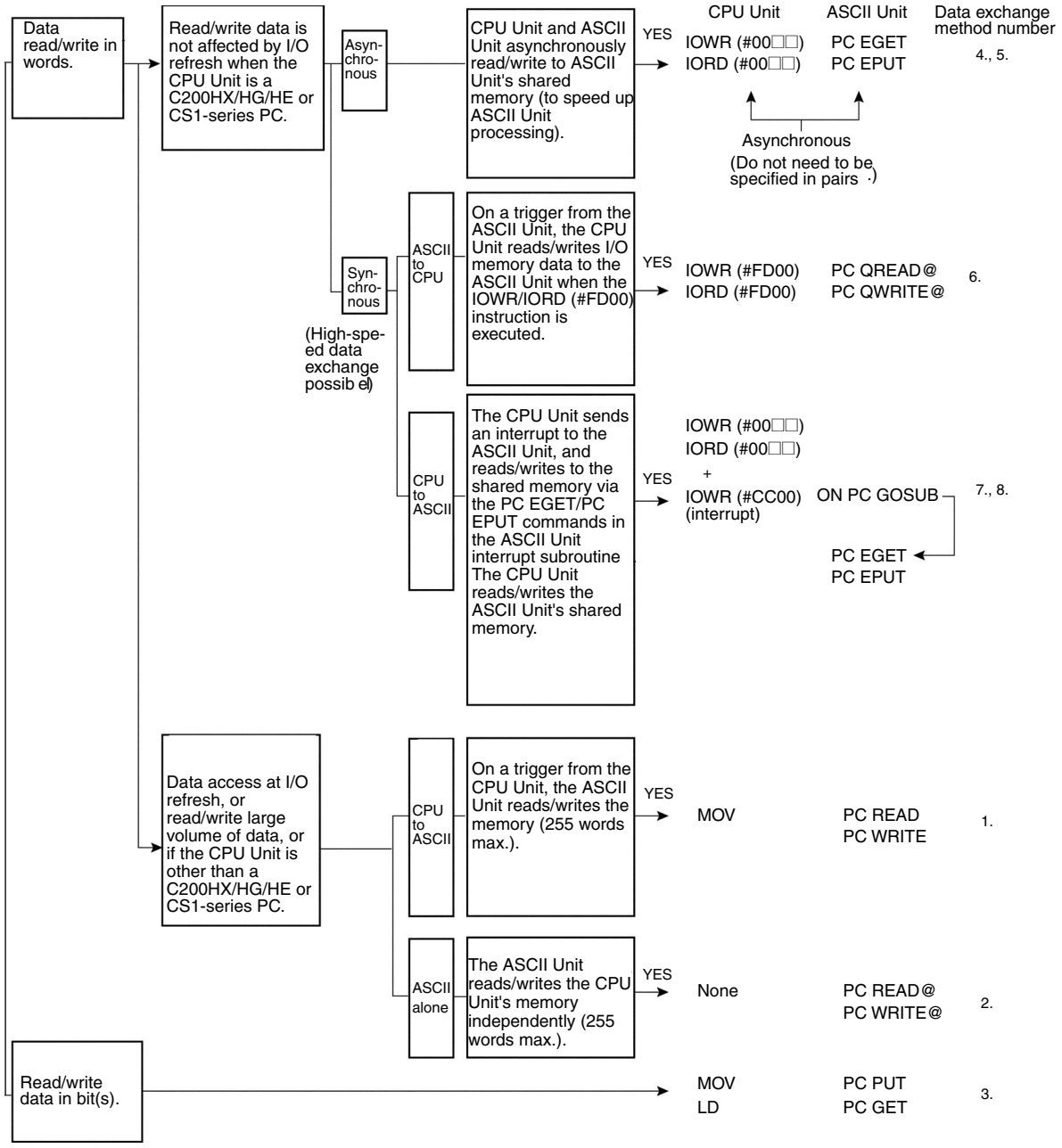
Note PC interrupts are disabled while the ASCII Busy Flag is ON.

2. An interrupt is sent at the execution of the IOWR (#CC00) instruction. The IOWR (#CC00) instruction in the CPU Unit's user program sends an interrupt, the interrupt number is sent to the ASCII Unit, and the program branches to the line number specified in the ON PC command. If the IOWR (#00□□) instruction has previously set input data in the shared memory, it is possible to access the shared memory via the PC EGET/EPUT command in the interrupt subroutine. This is supported only by the C200HX/HG/HE and CS1-series PCs.



Note PC interrupts are disabled while the ASCII Busy Flag is ON.

6-2 Selecting the Data Exchange Method



Comparison Chart of Data Exchange Methods

The differences between the data exchange methods are shown in the table below.

Data exchange method	CPU Unit instructions	ASCII Unit commands	Accessing Unit	Access trigger	Transfer timing	ASCII program	Max. no. of transfer words	
							Per command	Per cycle
1	MOV etc. (specify memory read/write area for execution of PC WRITE/PC READ commands)	PC WRITE PC READ (Reads/writes the CPU Unit's I/O memory)	ASCII Unit	CPU Unit	At I/O refresh	Pauses at execution of PC WRITE/ PC READ until completion of transfer.	PC WRITE/ PC READ: 255 words	Using a C200HX/HG/ HE or CS1-series PC, and when the number of bytes transferred per cycle set in the Setup Area (Bits 08 to 15 of DM m + 4) is 5A: 127 words 00: 20 words
2	Programming not required	PC WRITE@ PC READ@ (Reads/writes CPU Unit memory)	ASCII Unit	ASCII Unit	At I/O refresh	Pauses at execution of PC WRITE@/ PC READ@, until completion of transfer.	PC WRITE@/ PC READ@: 255 words	
3	MOV, LD, etc.	PC GET PC PUT (Reads/writes allocated IR/CIO area words)	Either	Either	At I/O refresh	Execution does not stop.	PC PUT: 1 word PC GET: 2 words	PC PUT: 1 word PC GET: 2 words
4, 5	IOWR (#00□□) IORD (#00□□) (C200HX/HG/ HE or CS1-series PC only)	PC EGET PC EPUT (Reads/writes to ASCII Unit's shared memory)	CPU Unit	Either	At IOWR/IORD (#00□□) execution	Execution does not stop.	PC EGET/ PC EPUT: 90 words IOWR/IORD (#00 □□): 90 words	No restrictions
6	IOWR (#FD00) IORD (#FD00) (C200HX/HG/ HE or CS1-series PC only)	PC QREAD@ PC QWRITE@	CPU Unit	ASCII Unit	At IOWR/IORD (#FD00) execution	Pauses at execution of PC QREAD/ PC QWRITE until IOWR/ IORD (#FD00) has been executed.	PC QREAD/ PC QWRITE: 128 words IOWR, IORD (#FD00): 128 words	No restrictions
7, 8	IOWR (#00□□) IORD (#00□□) + IOWR (#CC00) (C200HX/HG/ HE or CS1-series PC only)	ON PC GOSUB PC EGET/ EPUT	CPU Unit	CPU Unit	At IOWR/IORD execution	Execution does not stop.		No restrictions

Note There are some restrictions in the memory area addresses that can be used for methods 1 and 2. Refer to *6-6 Restrictions for CS1-series PCs* for details. Use methods 3 to 8 whenever possible.

6-3 Details of the Data Exchange Methods

6-3-1 Data Exchange Method No. 1

The ASCII Unit reads/writes the CPU Unit memory during the I/O refresh period on a trigger from the CPU Unit.

This method allows the exchange of comparatively large volumes of data with a maximum of 255 words, although it will cause the ASCII Unit to be on standby for some time. It is possible with any CPU Unit.

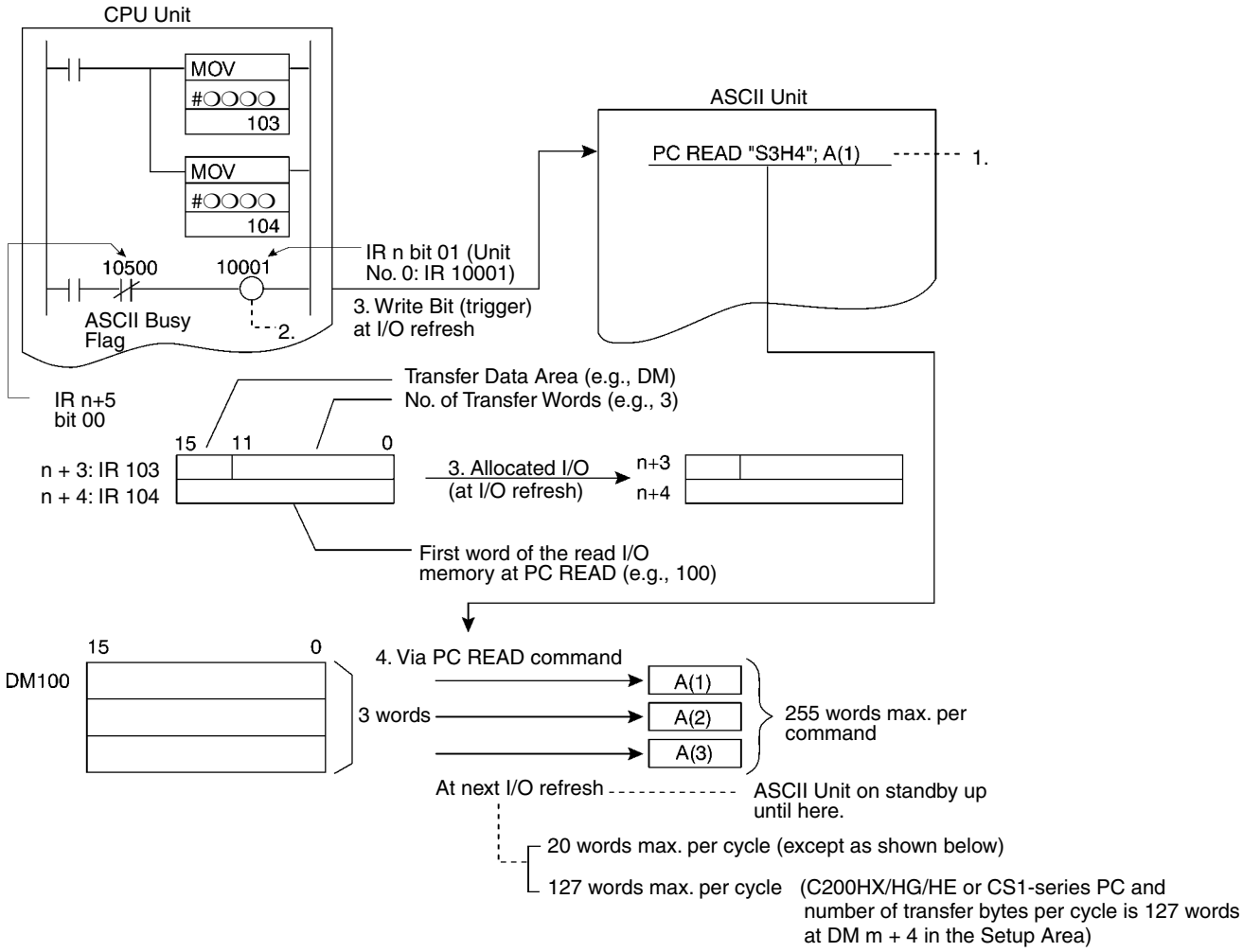
PC READ

When the ASCII Unit receives an ON Write Bit from the CPU Unit, it reads the memory words specified in the allocated IR/CIO words, and stores the contents in the specified variable.

1,2,3...

1. Request using the PC READ command.
2. Write Bit turned ON using the OUT instruction in the ladder diagram.
3. During the I/O refresh period, the Write Bit (IR/CIO n bit 01) from the CPU Unit to the ASCII Unit turns ON.
4. At the next I/O refresh period, the ASCII Unit reads data and the ASCII Busy Flag is turned ON by the system.
5. When the ASCII Busy Flag changes to ON, the Write Bit is turned OFF in the ladder diagram.
6. Once the PC READ command's data transfer has been completed, the ASCII Busy Flag is turned OFF by the system.

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



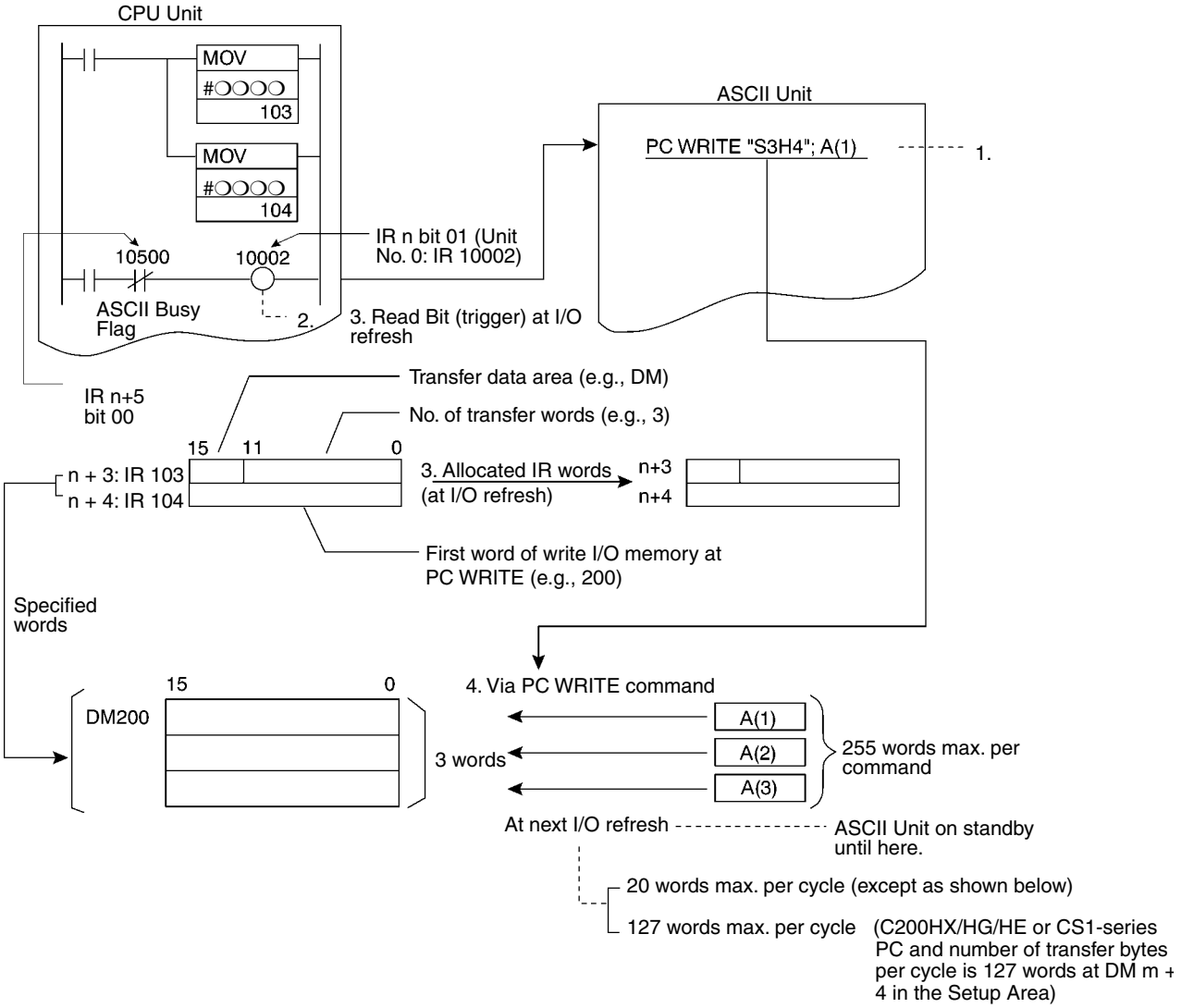
PC WRITE

When the ASCII Unit receives an ON Read Bit from the CPU Unit, it writes the contents of the specified variable to the I/O memory area specified in the allocated I/O.

1,2,3...

1. Request by PC WRITE command.
2. Read Bit turned ON via the OUT command in the ladder diagram
3. During the I/O refresh period, the Read Bit (n bit 02) from the CPU Unit to the ASCII Unit turns ON.
4. At the next I/O refresh period, the ASCII Unit writes data (ASCII Busy Flag goes ON).
5. When the ASCII Busy Flag changes to ON, the Read Bit turned OFF in the ladder diagram.
6. Once the PC WRITE command data transfer has been completed, the ASCII Busy Flag is turned OFF by the system.

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



6-3-2 Data Exchange Method No. 2

The ASCII Unit independently reads/writes the CPU Unit memory during the I/O refresh period.

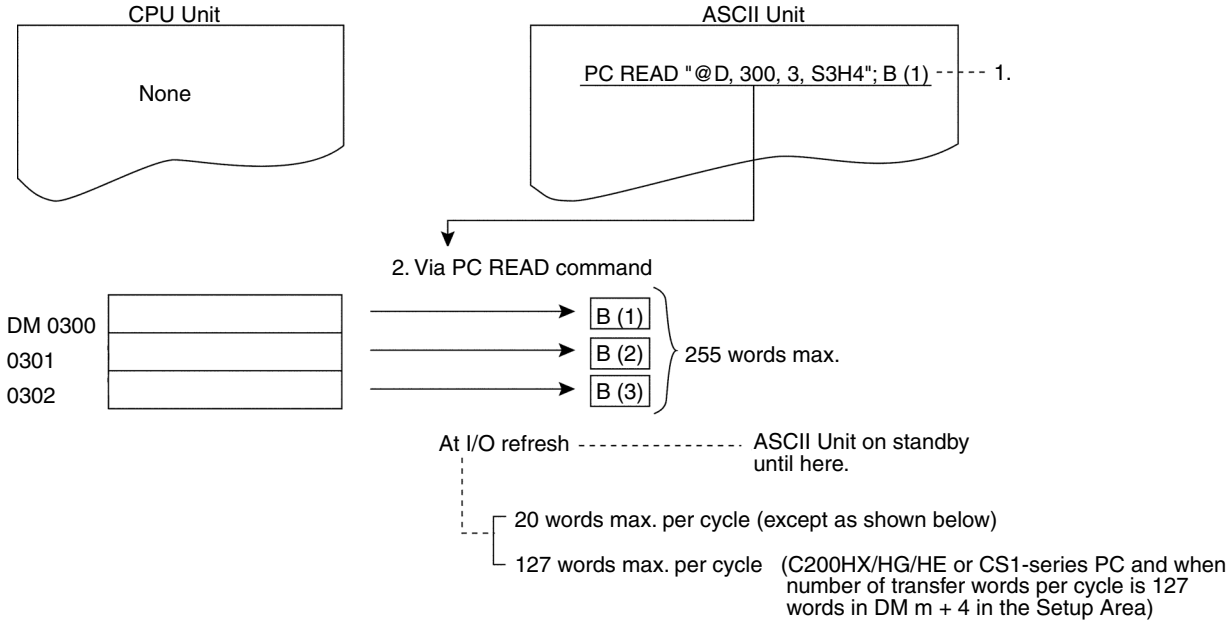
PC READ@

The ASCII Unit reads the CPU Unit memory as specified in the PC READ@ command.

1,2,3...

1. PC READ@ command executed.
2. The ASCII Unit reads data during the I/O refresh period (the ASCII Busy Flag goes ON).
3. When the PC READ@ command data transfer is completed, the ASCII Busy Flag goes OFF.

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



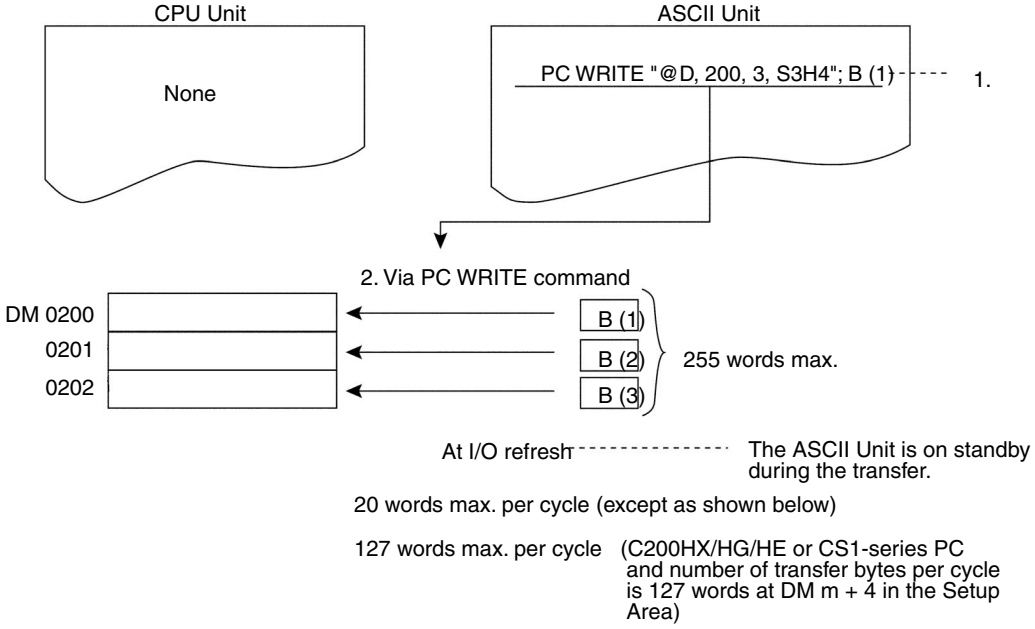
PC WRITE@

The ASCII Unit writes to the CPU Unit's internal I/O memory, as specified in the PC WRITE@ command.

1,2,3...

1. PC WRITE@ command executed.
2. The ASCII Unit writes data during the I/O refresh period (the ASCII Busy Flag goes ON).
3. When the PC WRITE@ command data transfer is completed, the ASCII Busy Flag goes OFF.

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



6-3-3 Data Exchange Method No. 3

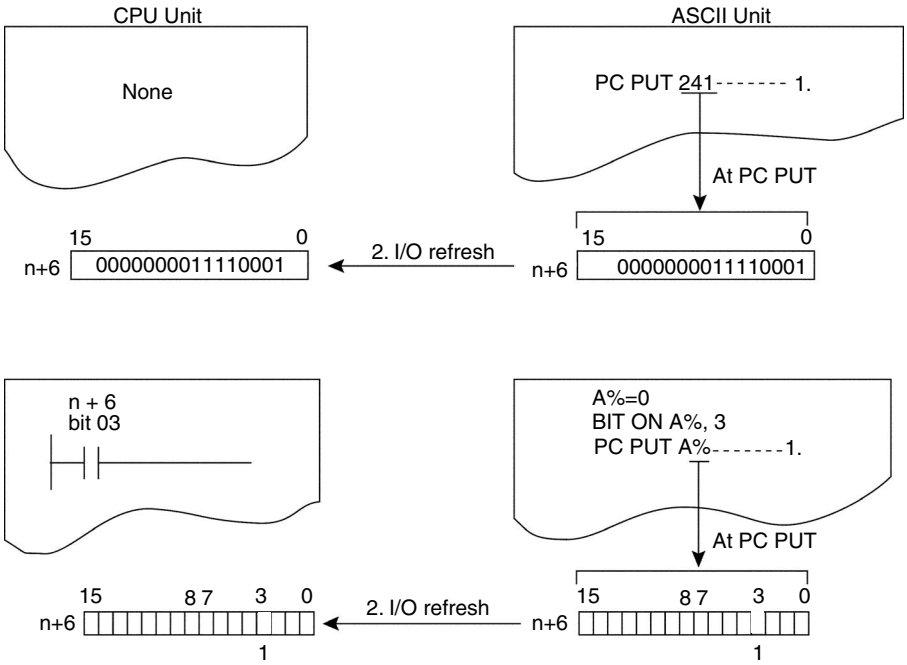
Single-bit or multibit data is read/written during the I/O refresh period from either the ASCII Unit or the CPU Unit.

PC PUT

The ASCII Unit converts decimal data to hexadecimal data and stores it at allocated word IR/CIO n + 6. It writes decimal values from –32768 to 32767.

1,2,3...

1. PC PUT command executed.
2. The ASCII Unit writes data during the I/O refresh period.

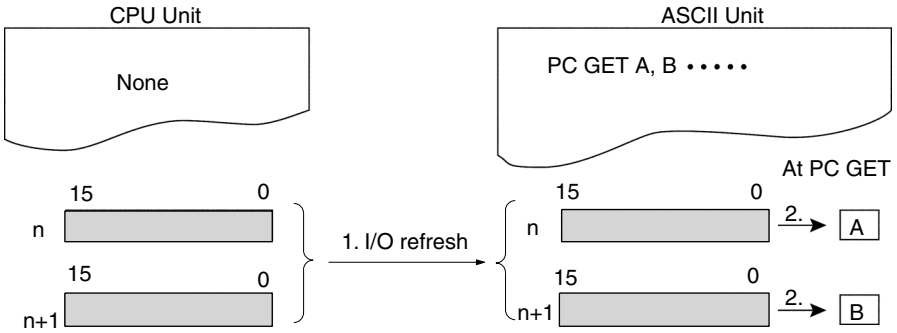


PC GET

The ASCII Unit converts the allocated word IR/CIO n bits 0 to 15 from hexadecimal data to decimal data, and stores them in the first variable. It also converts the word IR/CIO n + 1 bits 0 to 15 from hexadecimal to decimal data and stores them in the second variable.

1,2,3...

1. Data set during I/O refresh period.
2. At the execution of the PC GET command, the ASCII Unit reads the specified data.



6-3-4 Data Exchange Method No. 4

The ASCII Unit and CPU Unit both asynchronously read/write to the ASCII Unit's shared memory.

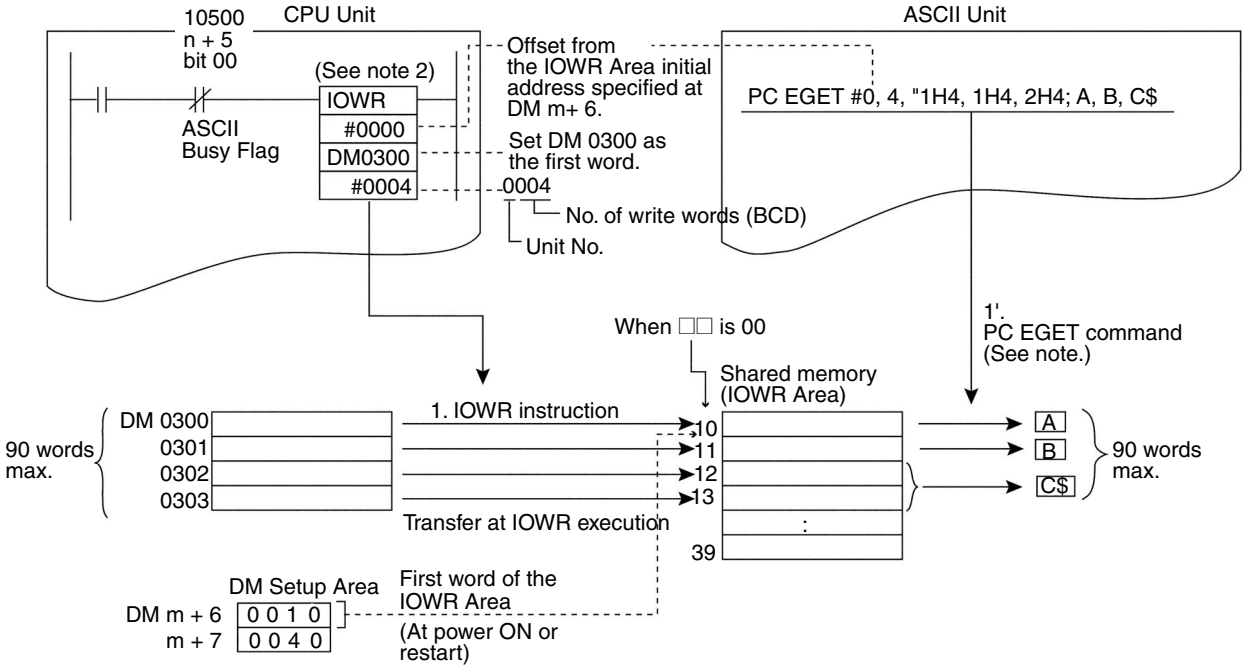
PC EGET

Using the PC EGET command, the ASCII Unit reads the IOWR Area of its shared memory area to a specified variable. Asynchronously, the CPU Unit writes specified memory area data to the ASCII Unit's shared memory areas using the IOWR instruction.

1,2,3...

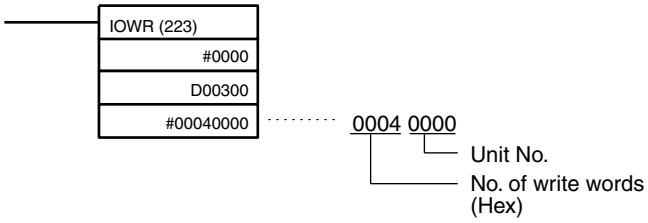
1. Using the IOWR (#00□□) instruction, write data to the ASCII Unit's shared memory area.
2. Using the PC EGET command, read data from the shared memory area.

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



Note 1. The PC EGET/EPUT commands can synchronize with the PC as they are executed by interrupts from the PC. PC interrupts are possible via IOWR instruction (control code #CC00) and allocated IR/CIO area words (word IR/CIO n bit 04 and word IR/CIO n + 2 bits 00 to 07).

2. The IOWR instruction is structured as shown below for the CS1-series PCs. The number of write words must be specified as a hexadecimal value.



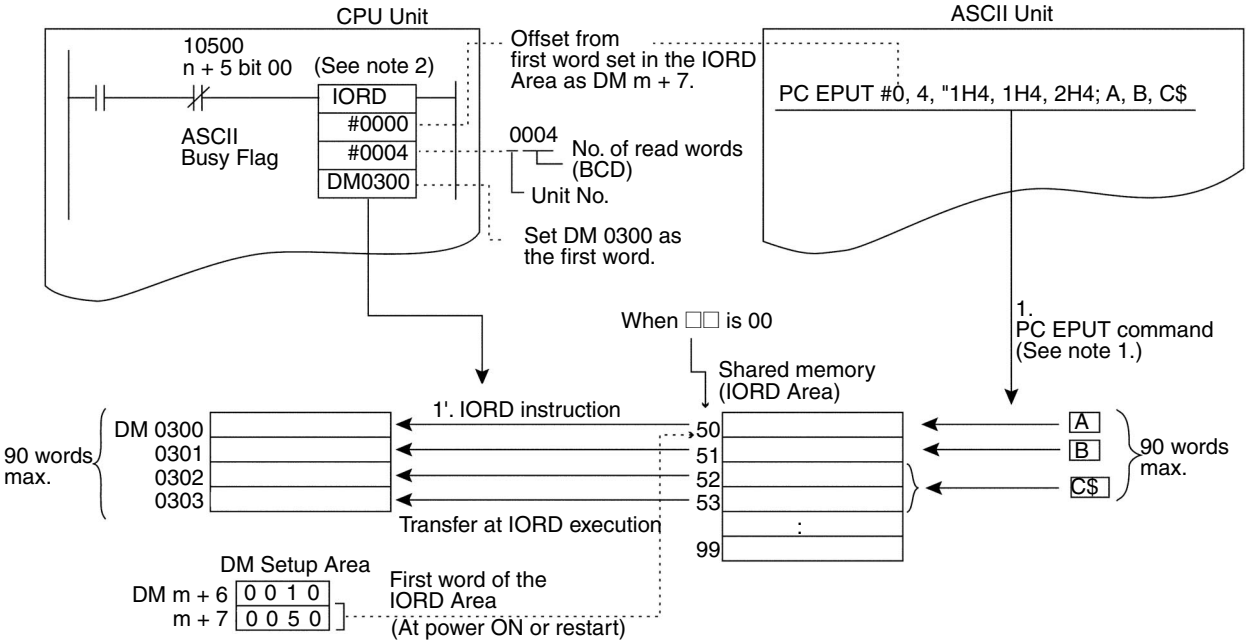
PC EPUT

The ASCII Unit writes the contents of specified variables to its shared memory IORD Area using the PC EPUT command. Asynchronously, the CPU Unit reads data from the ASCII Unit's shared memory and writes it to the specified I/O memory area using the IORD instruction.

1,2,3... 1. Write data to the shared memory area using the PC EPUT command.

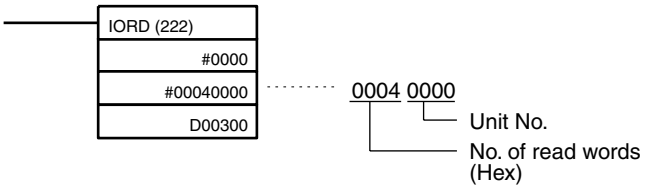
2. Read data from the ASCII Unit's shared memory area when the IORD (#00□□) instruction is executed in the ladder diagram.

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



Note 1. The PC EGET/EPUT commands can synchronize with the PC as they are executed by interrupts from the PC. PC interrupts are possible via IOWR instruction (control code #CC00) and allocated IR/CIO area words (word IR/CIO n bit 04 and word IR/CIO n + 2 bits 00 to 07).

2. The IORD instruction is structured as shown below for the CS1-series PCs. The number of read words must be specified as a hexadecimal value.



6-3-5 Data Exchange Method No. 5

On a trigger from the ASCII Unit, the CPU Unit reads/writes memory data to the ASCII Unit by executing an IOWR/IORD (#FD00) instruction.

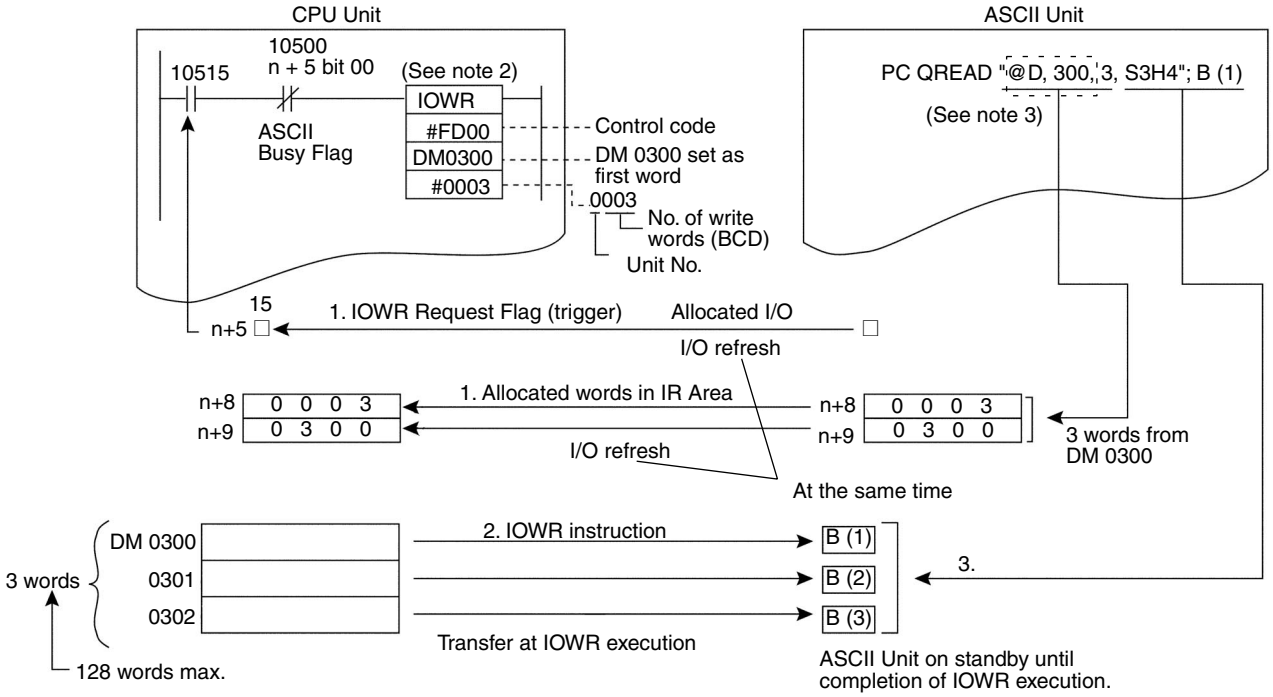
PC QREAD (Combined with IOWR (#FD00))

The ASCII Unit sets an IOWR Request Flag in the CPU Unit in bit 15 of IR/CIO n + 5. When the CPU Unit detects the ON status of the flag, it will execute the IOWR (#FD00) instruction. Based on IR/CIO n + 8 and n + 9, it writes the specified words to the ASCII Unit.

1,2,3...

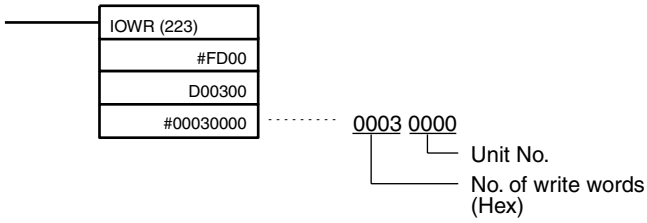
1. At I/O refresh, the ASCII Unit sends an IOWR Request Flag to the CPU Unit by the system.
2. Data is written to the ASCII Unit when the IOWR (#FD00) instruction is executed in the ladder diagram.
3. As a result of the PC QREAD command, the received data is read to a specified variable.

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



Note 1. IORD and IOWR instructions cannot be used for ASCII Units on Slave Racks.

2. The IOWR instruction is structured as shown below for the CS1-series PCs. The number of write words must be specified as a hexadecimal value.



3. The data set in words n+8 and n+9 by PC QREAD is used only as monitor data by the CPU Unit. The data that is actually transferred is specified in an operand of the IOWR instruction. This aspect of operation differs from that of the PC READ@ command, which reads data from the operand specified for it. For the CS1 Series, the range that can be addressed for PC QREAD differs from the range for IOWR, making it necessary to use dummy addresses when executing PC QREAD if it cannot address the required words directly. (The CPU Unit will not monitor the words when dummy addresses are used.) In the above example, the address of the first word, D07000 cannot be specified for PC QREAD even though it can be specified for IOWR. Here, a dummy address, like @D,0, can be used for PC QREAD.

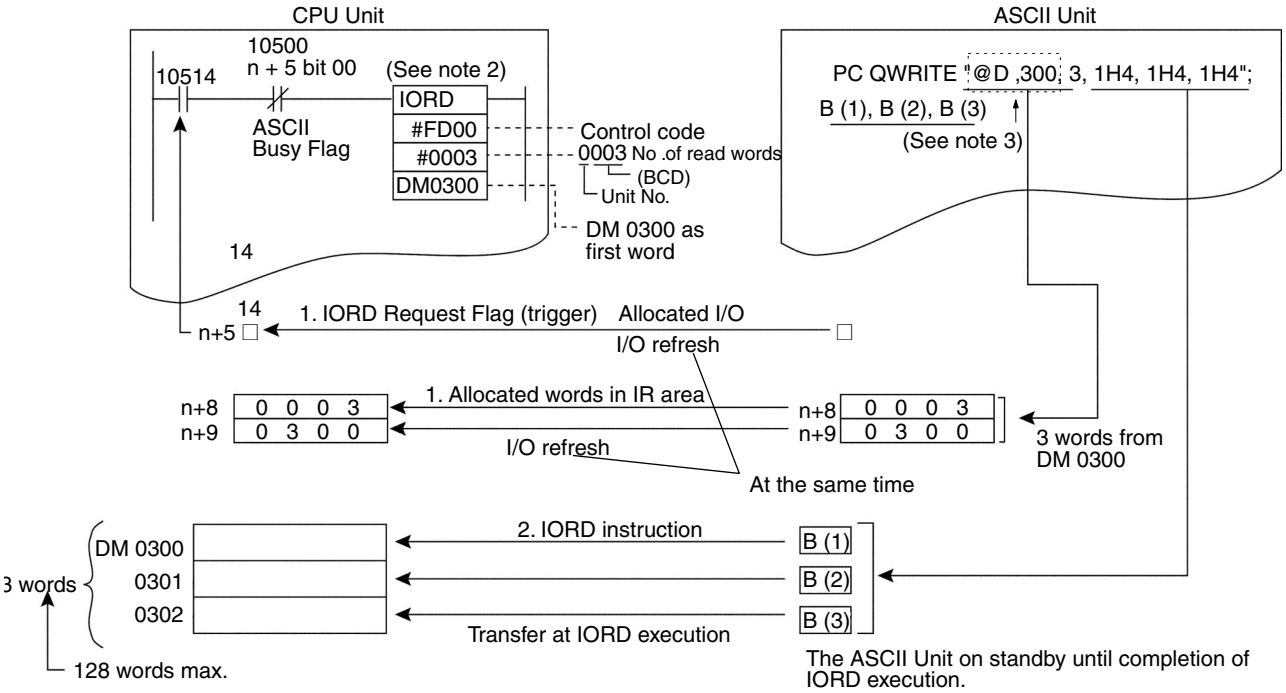
PC QWRITE (Combined with IORD (#FD00))

The ASCII Unit sends an IORD Request Flag to the CPU Unit by bit 14 of IR/CIO n + 5. When the CPU Unit detects the ON status of the flag, it will execute the IORD instruction. Based on IR/CIO n + 8 and n + 9, it reads the words from the ASCII Unit.

1,2,3...

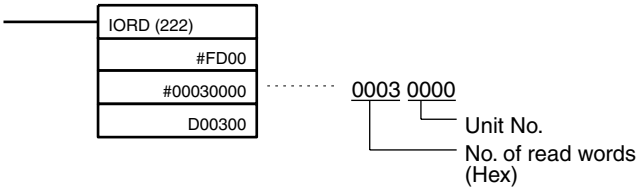
1. At I/O refresh, the ASCII Unit sends an IORD Request Flag to the CPU Unit by the system.
2. Data is prepared for transmission from the specified variable by the PC QWRITE command.
3. Data is read from the ASCII Unit when the IORD (#FD00) instruction is executed in the ladder diagram.

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



Note 1. IORD and IOWR instructions cannot be used for ASCII Units on Slave Racks.

2. The IORD instruction is structured as shown below for the CS1-series PCs. The number of read words must be specified as a hexadecimal value.



3. The data set in words n+8 and n+9 by PC QWRITE is used only as monitor data by the CPU Unit. The data that is actually transferred is specified in an operand of the IORD instruction. This aspect of operation differs from that of the PC WRITE@ command, which reads data from the operand specified for it. For the CS1 Series, the range that can be addressed for PC QWRITE differs from the range for IORD, making it necessary to use dummy addresses when executing PC QWRITE if it cannot address the required words directly. (The CPU Unit will not monitor the words when dummy addresses are used.) In the above example, the address of the first word, D07000 cannot be specified for PC QWRITE even though it can be specified for IORD. Here, a dummy address, like @D,0, can be used for PC QWRITE.

6-3-6 Data Exchange Method No. 6

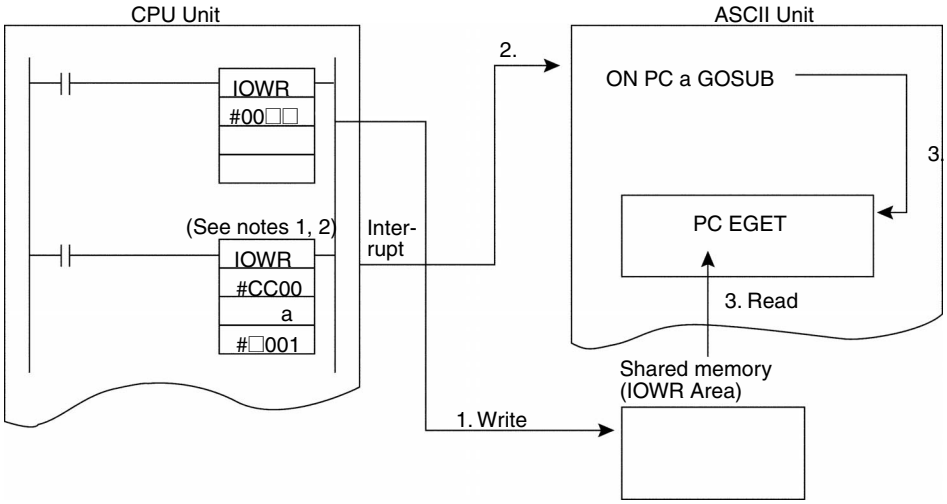
After the CPU Unit has written to the ASCII Unit's shared memory area, it sends an interrupt to the ASCII Unit, and during the interrupt subroutine, the ASCII Unit reads from the shared memory.

Note See *6-4-9 Sending Interrupts from CPU Unit to ASCII Unit: PC Interrupts* for details on PC interrupts.

1,2,3...

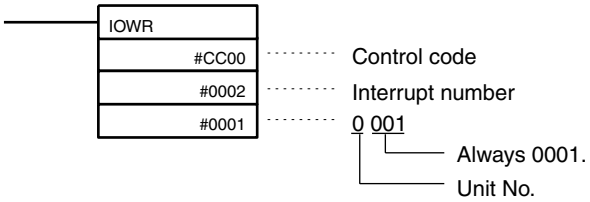
1. The CPU Unit writes to the ASCII Unit shared memory area (IOWR Area) when the IOWR instruction (#00□□) is executed in the ladder diagram.
2. The CPU Unit sends an interrupt to the ASCII Unit when the IOWR instruction (#CC00) is executed in the ladder diagram.
3. Once the interrupt is received by the ASCII Unit, data is read in the ASCII Unit via the PC EGET command.

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

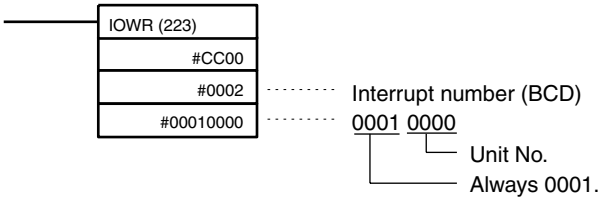


Note

1. The following IOWR instruction would be used if the ASCII Unit was unit number 0 and the interrupt number was 2.



2. The structure of the IOWR instruction for the CS1-series PCs is shown below.



6-3-7 Data Exchange Method No. 7

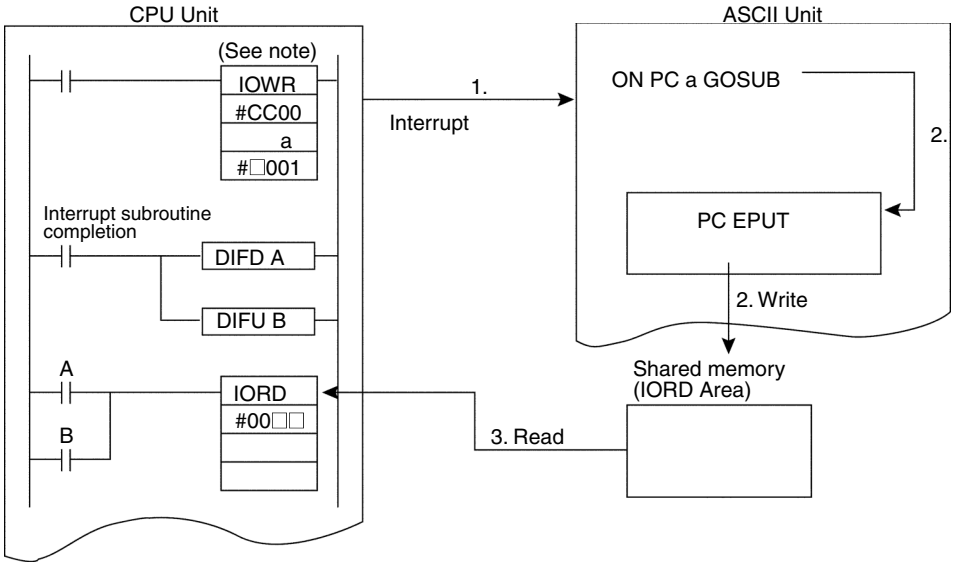
The CPU Unit sends an interrupt to the ASCII Unit, and the ASCII Unit writes data to the shared memory. After the completion of the interrupt subroutine, the CPU Unit reads data from the shared memory.

Note See *6-4-9 Sending Interrupts from CPU Unit to ASCII Unit: PC Interrupts* for details on PC interrupts.

1,2,3...

1. The CPU Unit sends an interrupt to the ASCII Unit when the IOWR instruction (#CC00) is executed in the ladder diagram.
2. The ASCII Unit receives the interrupt and writes data to the shared memory using a PC EPUT command.
3. When a change is detected in the Interrupt Subroutine Completed Flag (1 to 0 or 0 to 1), the shared memory is read when the IORD instruction (#00□□) is executed in the ladder diagram.

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

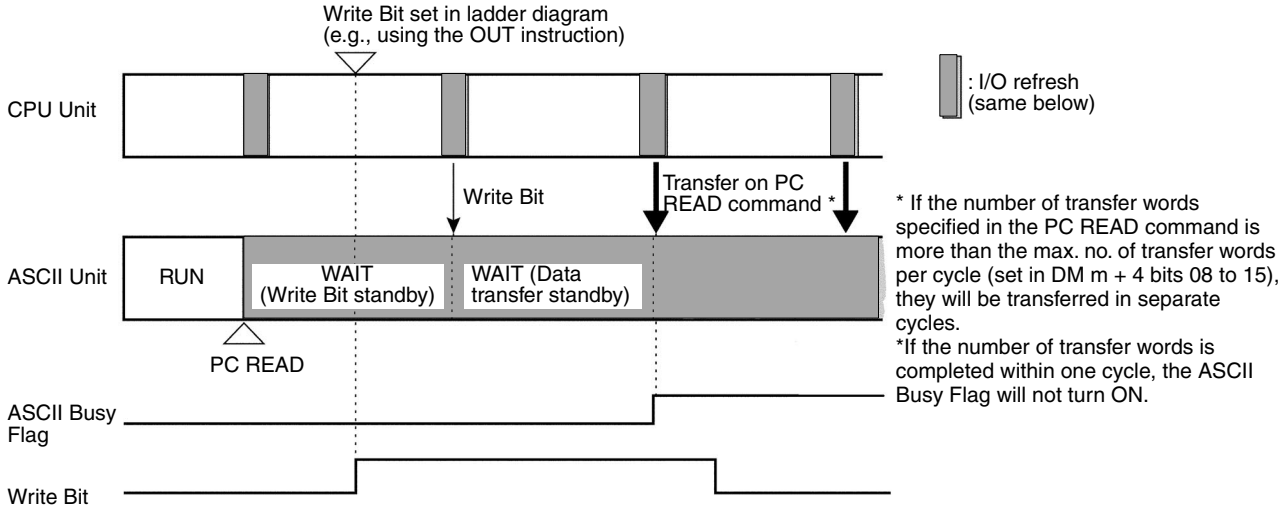


Note Refer to the note for method 6 on the previous page.

6-4 Data Exchange Time Charts

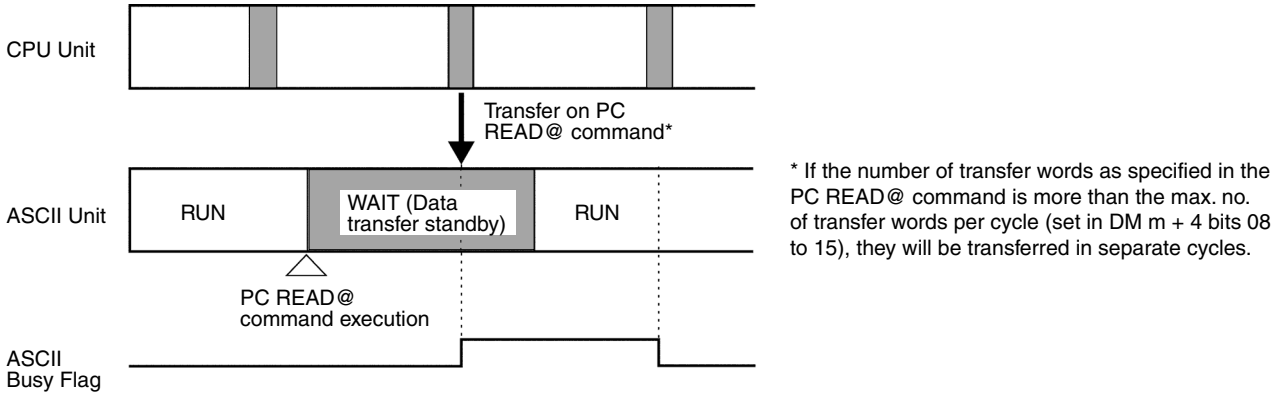
6-4-1 PC READ/PC WRITE Timing

After a trigger is sent from the CPU, data is transferred at the next I/O refresh period. After receiving a PC WRITE or PC READ command, the ASCII Unit puts the BASIC program on standby until the data transfer is complete.



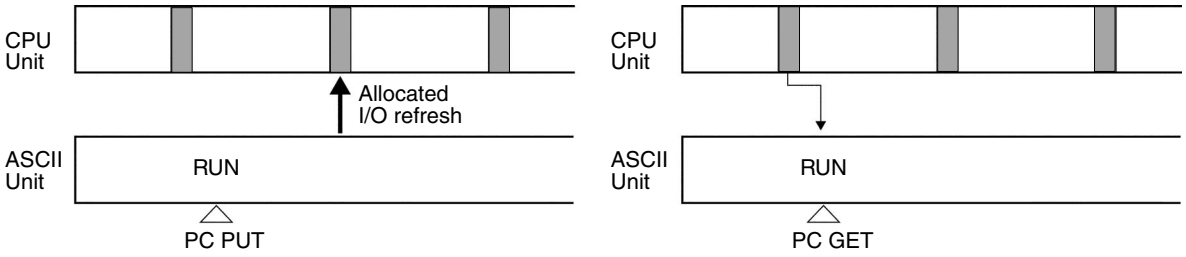
6-4-2 PC READ@/PC WRITE@ Timing

Data transfer occurs during the I/O refresh period immediately after the execution of the PC READ@/PC WRITE@ command. The ASCII Unit puts the BASIC program on standby until the completion of the PC READ@/PC WRITE@ command.



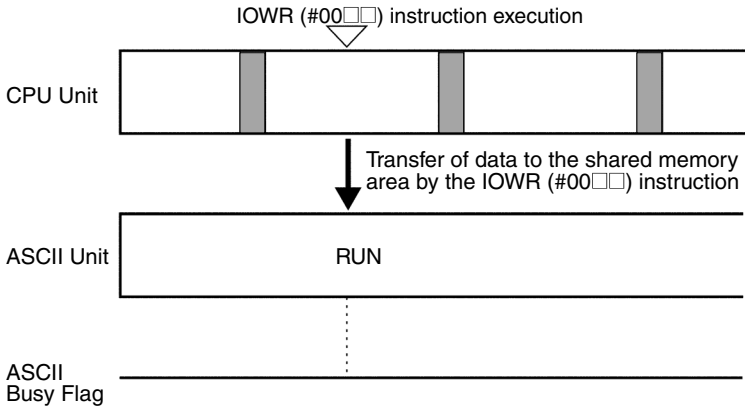
6-4-3 PC PUT/PC GET Timing

Data is written during the I/O refresh period immediately after the execution of the PC PUT command. The ASCII Unit BASIC program continues execution. Data is read at the I/O refresh immediately after the execution of the PC GET command.



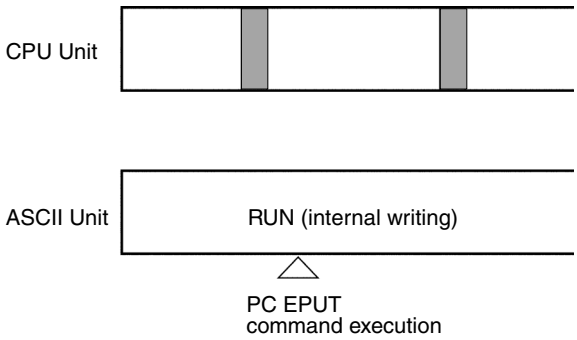
6-4-4 IOWR (#00□□)/IORD (#00□□) Timing

Data transfer occurs immediately after the IOWR (#00□□)/IORD (#00□□) instruction is executed. The ASCII Unit BASIC program continues execution.



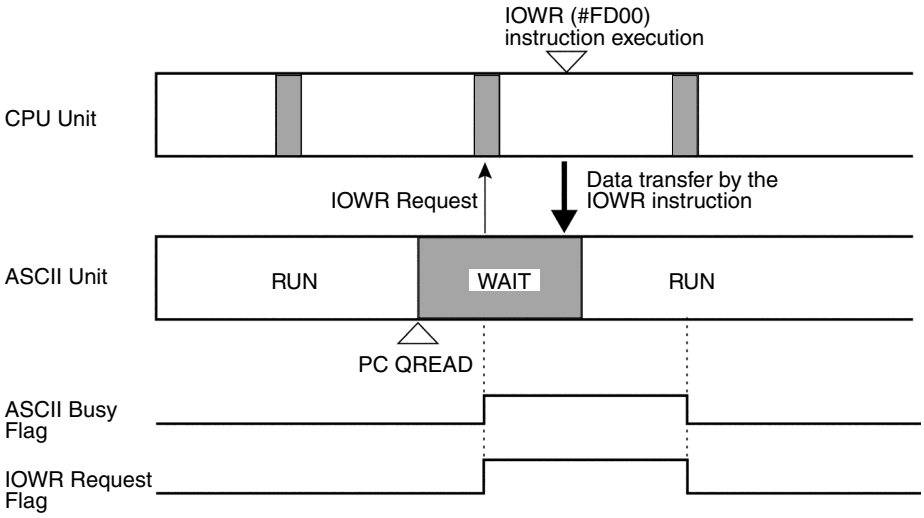
6-4-5 PC EPUT/PC EGET Timing

The internal shared memory is read/written to immediately when the execution of the PC EPUT/PC EGET command. The ASCII Unit's BASIC program continues execution.



6-4-6 IOWR (#FD00)/IORD (#FD00) and PC QREAD/PC QWRITE Timing

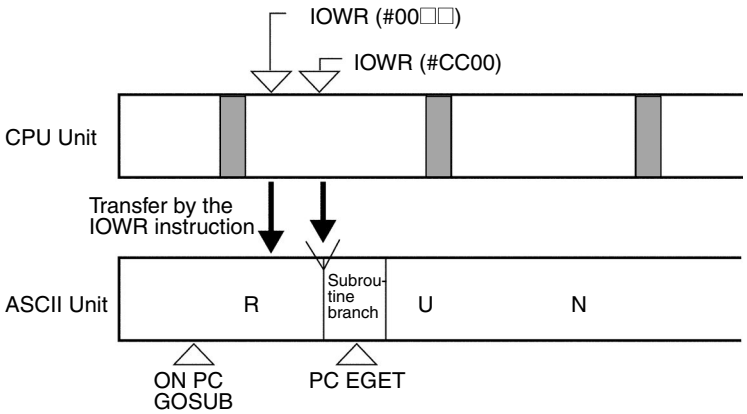
Data transfer occurs as soon as a trigger arrives from the ASCII Unit and the IOWR (#FD00)/IORD (#FD00) command is executed. The ASCII Unit is on standby from the execution of the PC QREAD/PC QWRITE command until the execution of the IOWR (#FD00)/IORD (#FD00) command.



Note The ASCII Busy Flag turns ON for PC QREAD only.
It does not turn ON for PC QWRITE.

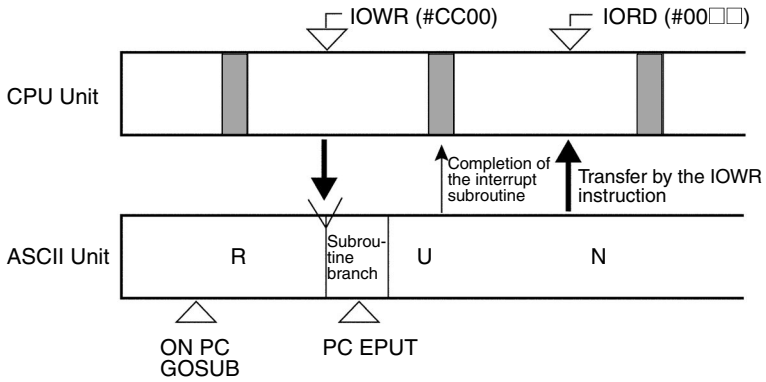
6-4-7 Combined IOWR (#00□□), IOWR (#CC00), ON PC GOSUB, and PC EGET Timing

Data is transferred immediately by the IOWR (#00□□) instruction. The IOWR (#CC00) instruction sends an interrupt to the ASCII Unit. The ASCII Unit receives data via the PC EGET command in an interrupt subroutine. Data transfer in the CPU Unit is completed within 1 cycle.



6-4-8 Combined IOWR (#CC00), IORD (#00□□), ON PC GOSUB, and PC EPUT Timing

The IOWR (#CC00) instruction sends an interrupt to the ASCII Unit. The interrupt subroutine is executed immediately andwrites to the shared memory using the PC EPUT command. At the next I/O refresh period, the completion of the interrupt subroutine is transferred to the CPU Unit, and at the next cycle data is transferred by the IORD (#00□□) instruction.

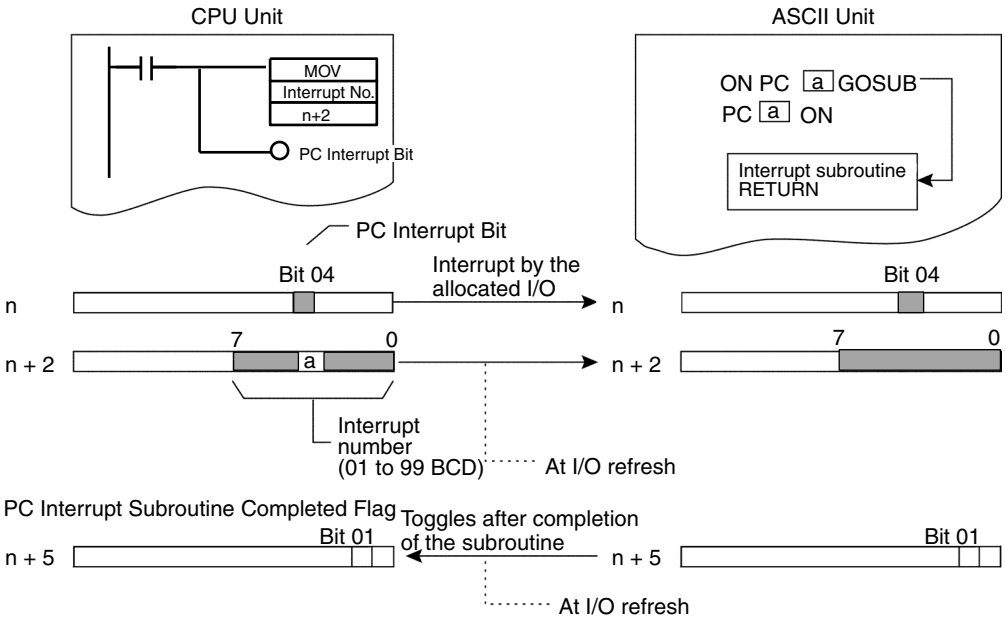


6-4-9 Sending Interrupts from CPU Unit to ASCII Unit: PC Interrupts

There are two methods for sending PC Interrupts as described below.

Using PC Interrupt Bit

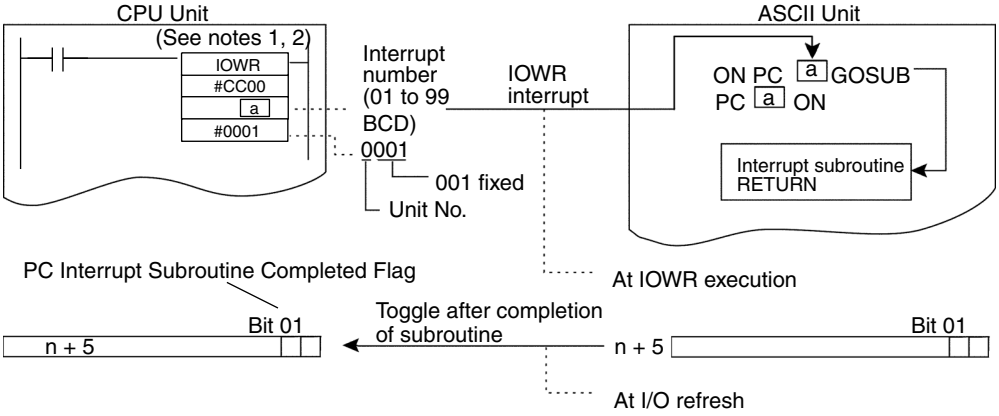
The CPU Unit sets an interrupt number in the ASCII Unit in the allocated words in the IR/CIO area. When the PC Interrupt Bit turns ON, an interrupt is sent. The ASCII Unit executes the interrupt subroutine if the interrupt number is the same as the ON PC command interrupt number. When the interrupt subroutine has been completed, the Interrupt Subroutine Completed Flag is toggled (OFF to ON or ON to OFF).



Using IOWR Interrupt: Control Code: #CC00 (C200NX/HG/HE and CS-1seried PCs only)

The CPU Unit indicates an interrupt number to the ASCII Unit by using the IOWR (#CC00) instruction, and sends an interrupt. The ASCII Unit executes the interrupt subroutine which matches the interrupt number in an ON PC command. When the interrupt subroutine is completed, the Interrupt Subroutine Completed Flag is toggled (OFF to ON or ON to OFF). This method is supported only by the C200HX/HG/HE and CS1-series PCs.

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



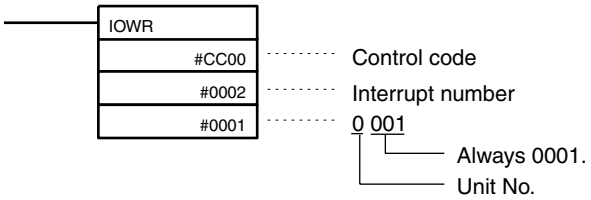
When using the PC Interrupt Bit described on the previous page, the interrupt is sent as soon as the IOWR instruction is executed, in contrast to interrupting using the PC Interrupt Bit (described on previous page), by which the interrupt is sent during the I/O refresh period. For this reason, high-speed interrupts to the ASCII Unit occur in the same cycle, not in the next cycle.

By combining the IOWR/IORD (#00□□) instruction at the CPU Unit with the PC EGET/PC EPUT commands in the ASCII Unit's interrupt subroutine, reads/writes between the CPU Unit and the ASCII Unit can occur within a single cycle.

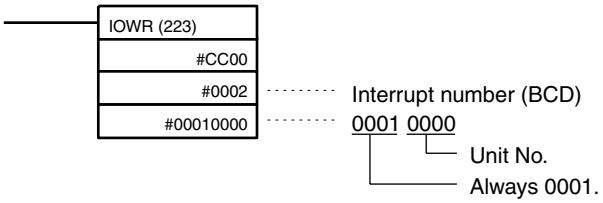
The following is an example of a program.

Note

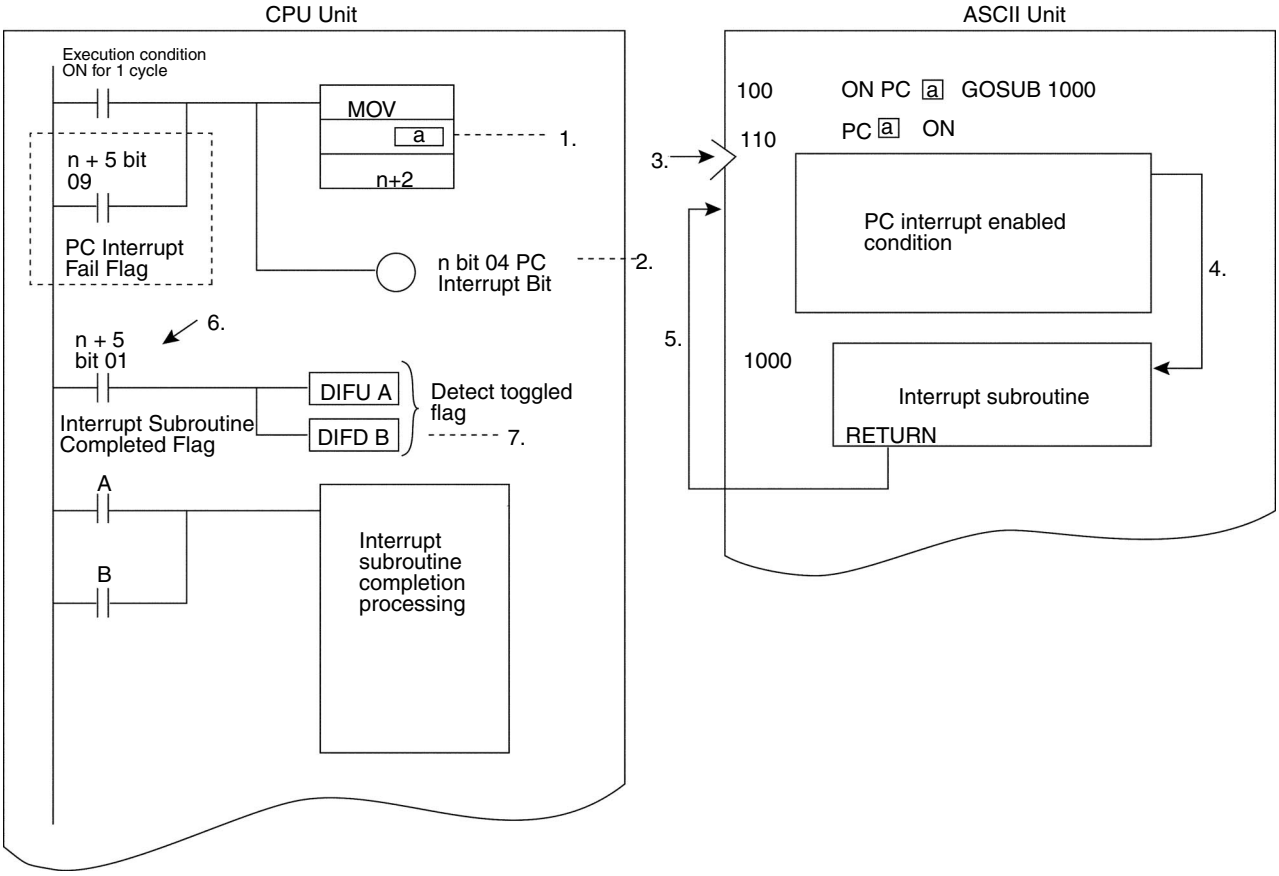
1. The following IOWR instruction would be used if the ASCII Unit was unit number 0 and the interrupt number was 2.



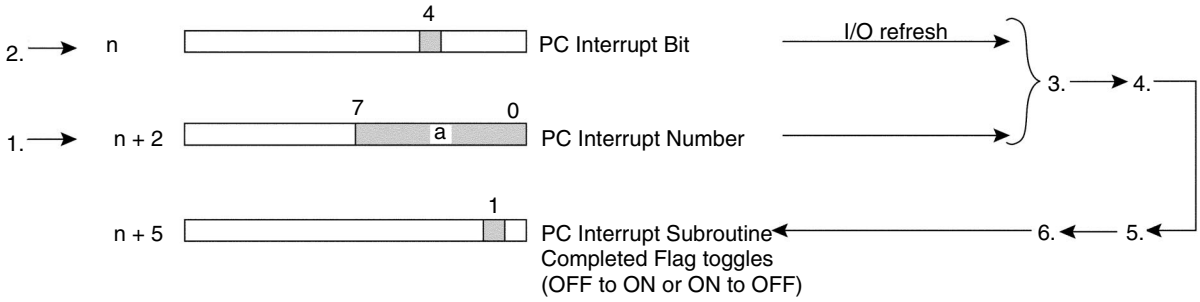
2. The structure of the IOWR instruction for the CS1-series PCs is shown below.



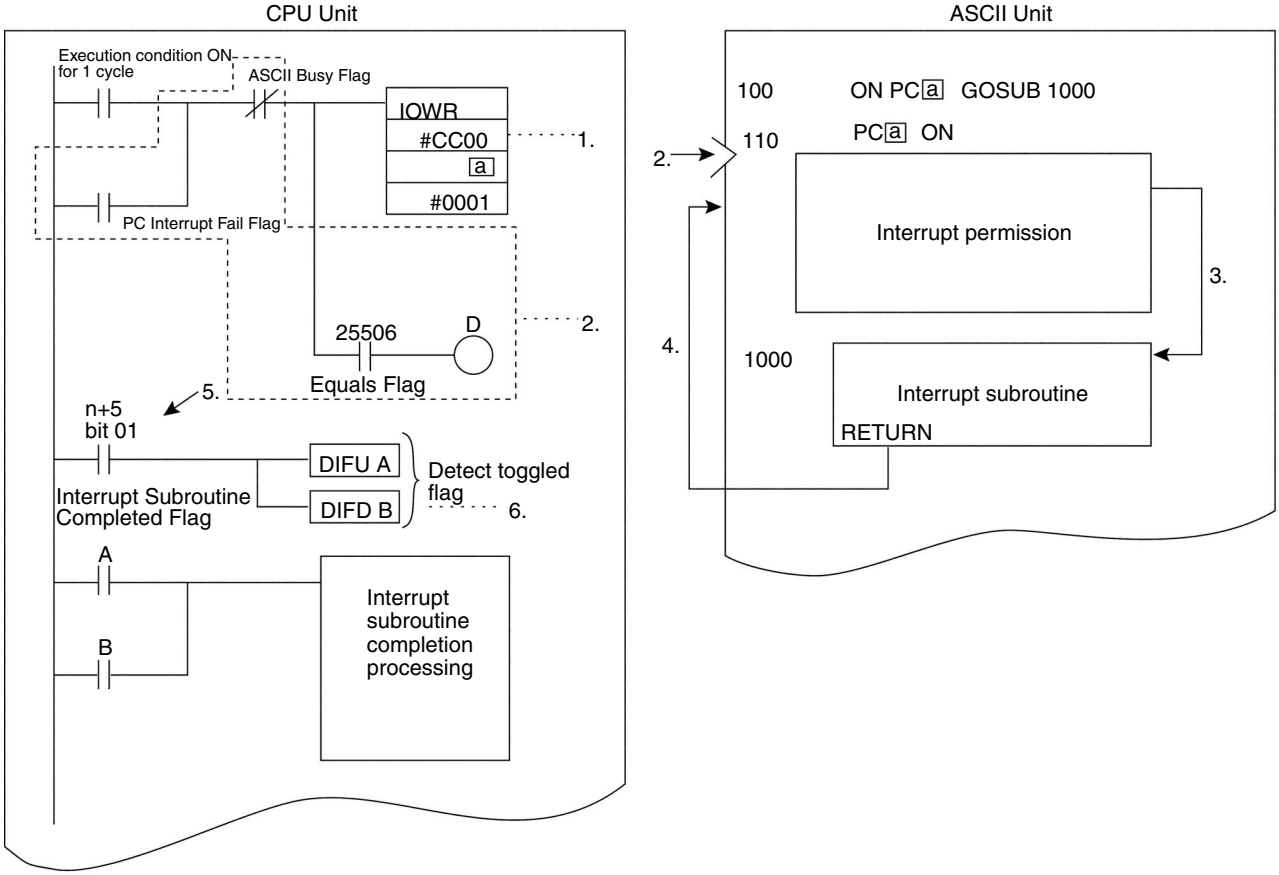
Interrupt by PC Interrupt Bit



Note Areas within dotted lines denote retry processing when PC interrupt has failed due to conditions such as a PC OFF (stop interrupt) condition in the ASCII Unit. If an interrupt is not accepted due to PC OFF condition when attempting to execute the interrupt by turning ON the PC Interrupt Bit (word n bit 04), the Interrupt Fail Flag (word n + 2 bit 09) will be set to ON (1) for one scan in the next cycle.



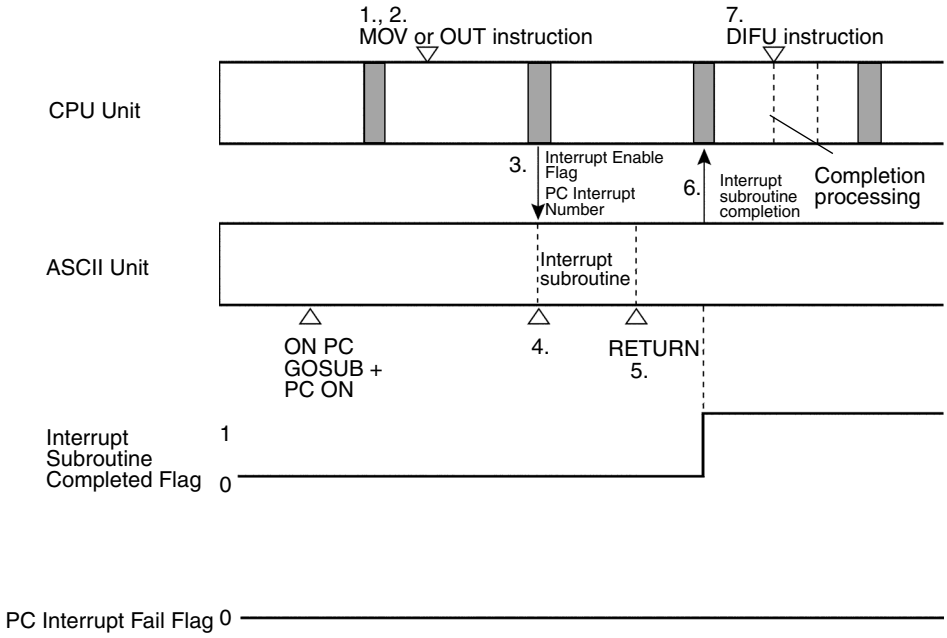
IOWR (#CC00) Interrupt: C200HX/HG/HE or CS1-series PCs Only



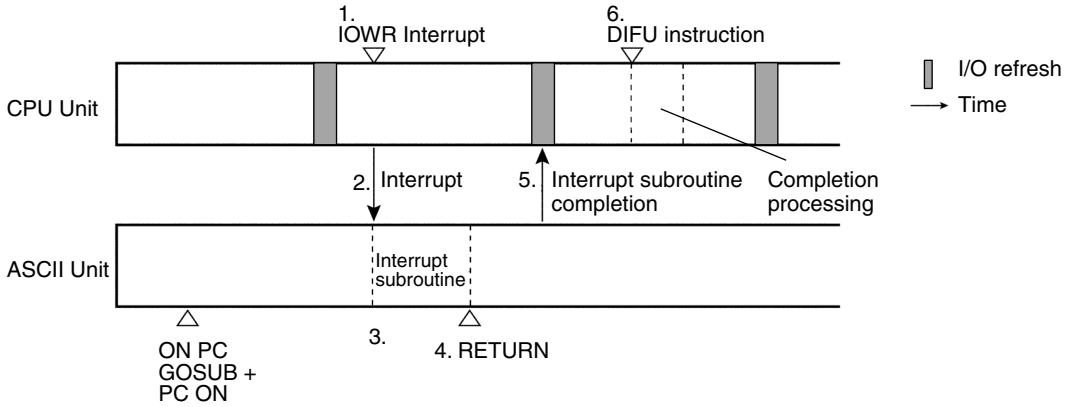
Note Areas within dotted boxes denote retry processing when PC interrupt has failed due to conditions such as a PC OFF (stop interrupt) condition in the ASCII Unit. If an IOWR (#CC00) interrupt is sent but not accepted, the PC Interrupt Fail Flag will be set to ON (1) for one scan in the next cycle. The operation of the PC Interrupt Fail Flag when an interrupt is not accepted is explained next.

Timing Charts

Interrupt Execution for PC Interrupt Bit

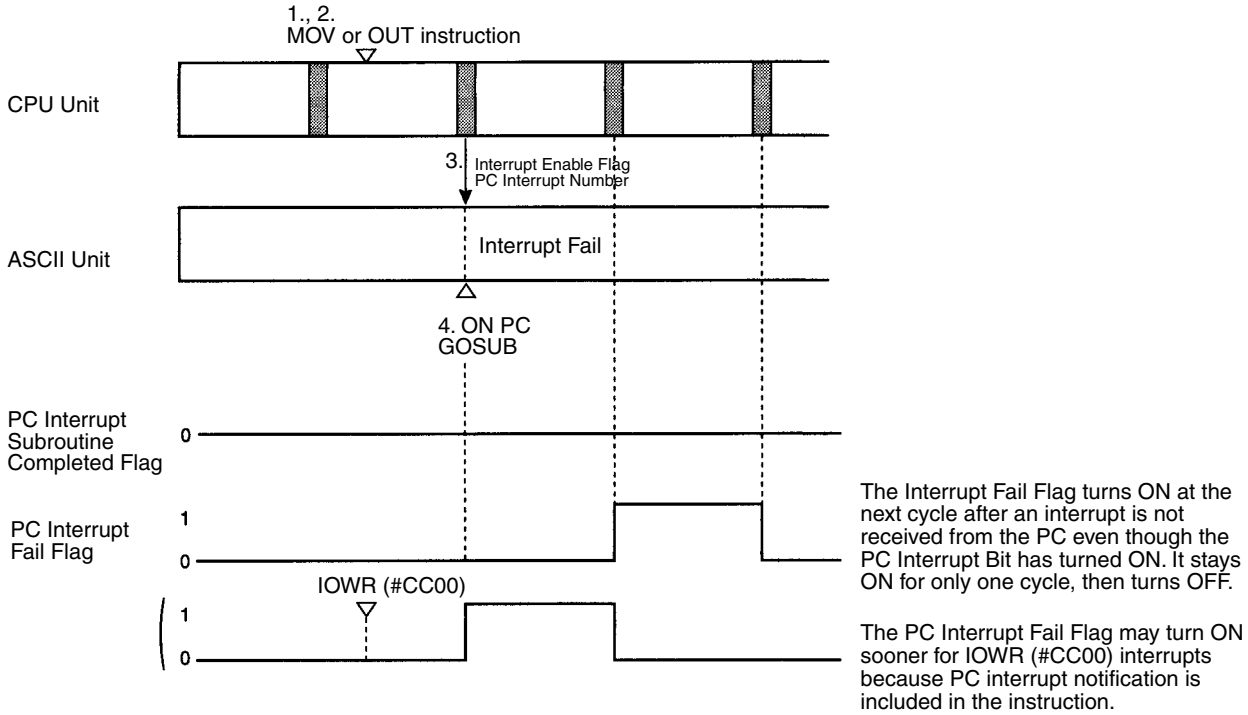


Interrupt Execution for IOWR (#CC00) Interrupts

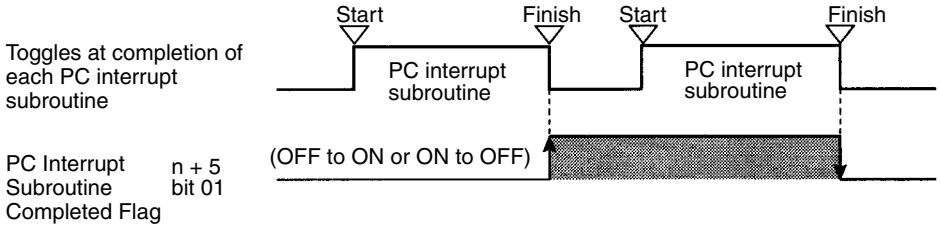


As shown above, interrupts using the IOWR (#CC00) instruction can be processed in one cycle less than interrupts using the PC Interrupt Bit.

Interrupt Failure (Same for PC Interrupt Bit Interrupts and IOWR (#CC00) Interrupts)



PC Interrupt Subroutine Completed Flag Changes

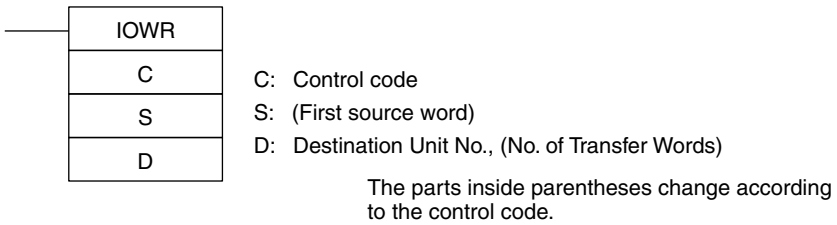


Note The time chart shown here indicates the flag operations which are different from the PC flag operations.

6-5 IOWR/IORD Instruction Specifications

The IOWR/IORD instructions are available only with the C200HX/HG/HE and CS1-series PCs. They cannot be used for ASCII Units on Slave Racks.

6-5-1 IOWR Instruction (SPECIAL I/O UNIT WRITE)



The IOWR instruction can be used with the ASCII Unit in the following four ways.

Writing to the ASCII Unit's Variables

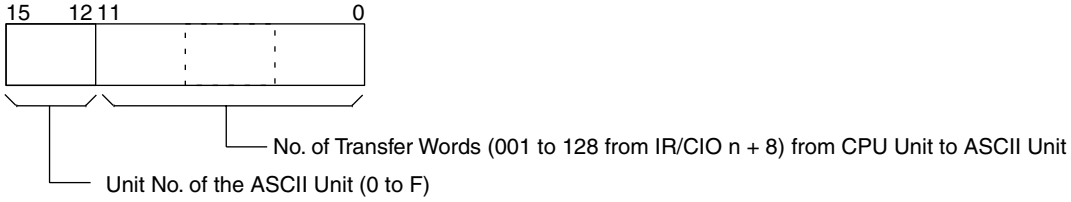
C:

Value (4-digit hex.)	Details	Combination
#FD00	Allocated IR/CIO area words IR/CIO n + 8 and IR/CIO n + 9 are transferred to the ASCII Unit's variable.	Must be used in combination with the ASCII Unit's PC QREAD@ command.

S: First source word in CPU Unit memory, from IR/CIO n+8 and IR/CIO n+9.

C200HX/HG/HE

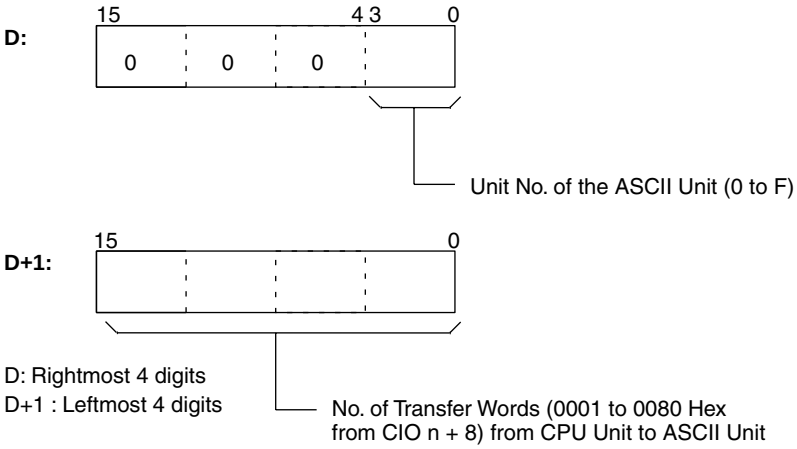
D:



Data

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

CS1-series PCs



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

Writing to the Shared Memory of the ASCII Unit

C:

Value (4-digit hex.)	Details	Combination
#00□□	Specified word data of the specified memory area is transferred to the first word* + □□ of the ASCII Unit's shared memory IOWR Area. *The first word in IOWR Area is set in DM m + 6 allocated in the DM Setup Area.	Can be used in combination with the ASCII Unit's PC EGET command. Can also be used on its own.

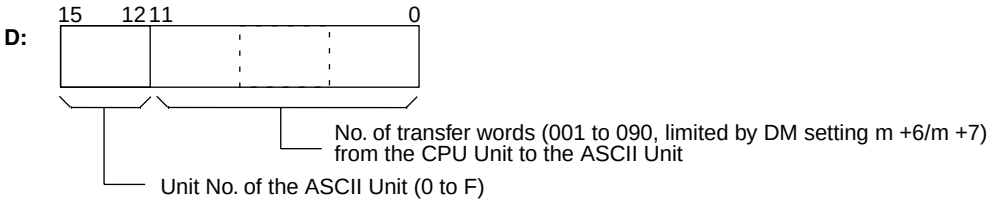
S: First source word in CPU Unit memory.

Data

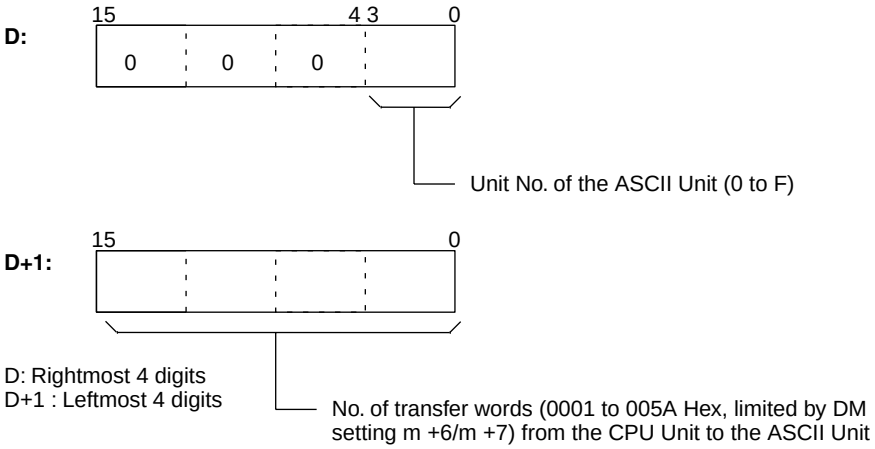
Item	Detail
PC	C200HX/HG/HE
Operand	S
IR Area 1	IR 000 to IR 235
SR Area 1	SR 236 to SR 255
SR Area 2	SR 256 to SR 299
IR Area 2	IR 300 to IR 511
HR Area	HR 00 to HR 99
AR Area	AR 00 to AR 27
LR Area	LR 00 to LR 63
TC Area	TC 000 to TC 511
TR Area	---
DM Area	DM 0000 to DM 6655
EM Area	---
Indirect DM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

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

C200HX/HG/HE



CS1-series PCs



Interrupts to the ASCII Unit

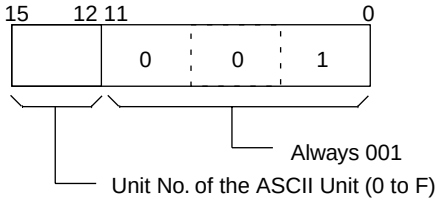
C:

Value (4-digit hex.)	Details	Combination
#CC00	Sends an interrupt from the CPU Unit to the ASCII Unit.	Must be used in combination with the ASCII Unit's ON PC (Interrupt Number) command.

S: Interrupt Number (corresponds to the ASCII Unit's ON PC (Interrupt Number) command).

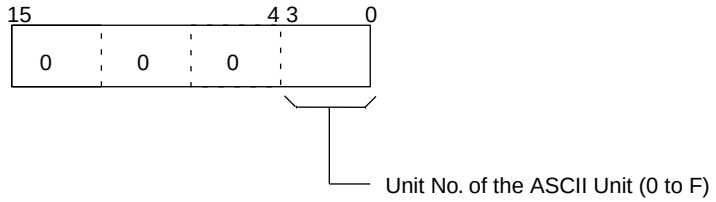
C200HX/HG/HE

D:

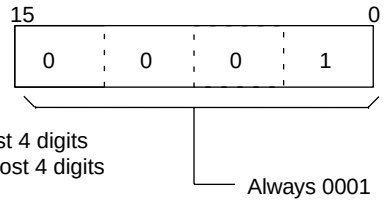


CS1-series PCs

D:



D+1:



D: Rightmost 4 digits
D+1 : Leftmost 4 digits

Clearing Errors in the ASCII Unit

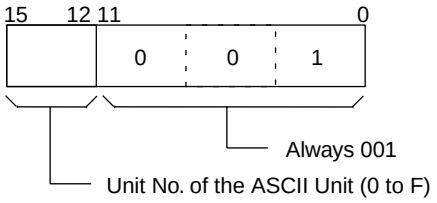
C:

Value (4-digit hex.)	Details
#EC00	The current error status is cleared and the current errors are registered in the error history table as old errors.

S: Always #0000

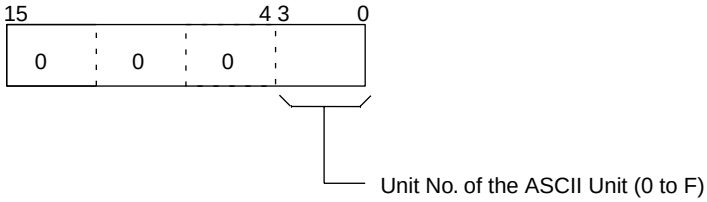
C200HX/HG/HE

D:

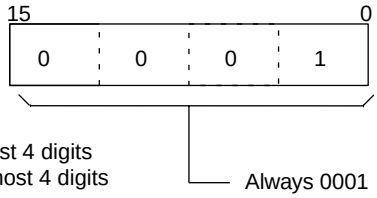


CS1-series PCs

D:



D+1:



D: Rightmost 4 digits
D+1 : Leftmost 4 digits

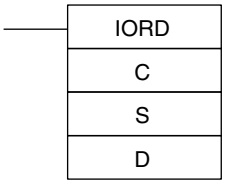
Flags

Value (see note)	ON	OFF
Instruction Execution Error Flag (ER) (SR 25503)	• The number of transfer words is not in BCD, or it is 0 words or greater than 128 words • The indirectly addressed DM address is greater than 6656 or not BCD. • The Unit No. of the ASCII Unit is outside the range 0 to F, or is on a SYSMAC BUS Slave Rack. • The number of transfer words is not BCD. • The instruction was not correctly completed.	C, S, and D settings are correct.
Carry Flag (CY) (25504)	---	---
Greater Than Flag (GR) (25505)	---	---
Equals Flag (EQ) (25506)	Writing was correctly completed.	Writing was not correctly completed.

Value (see note)	ON	OFF
Less Than Flag (LE) (25507)	---	---
Overflow Flag (OF) (25404)	---	---
Underflow Flag (UF) (25405)	---	---
Negative Flag (N) (25402)	---	---

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

6-5-2 IORD Instruction (SPECIAL I/O UNIT READ)



C: Control code
S: Source Unit No., No. of transfer words
D: First destination word

There are two methods of using the IORD instruction for the ASCII Unit as follows:

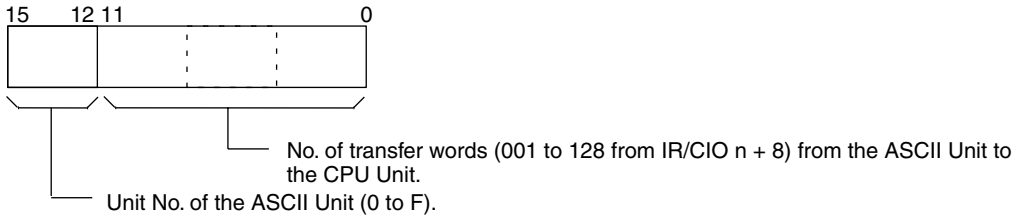
Reading from ASCII Unit Variables

C:

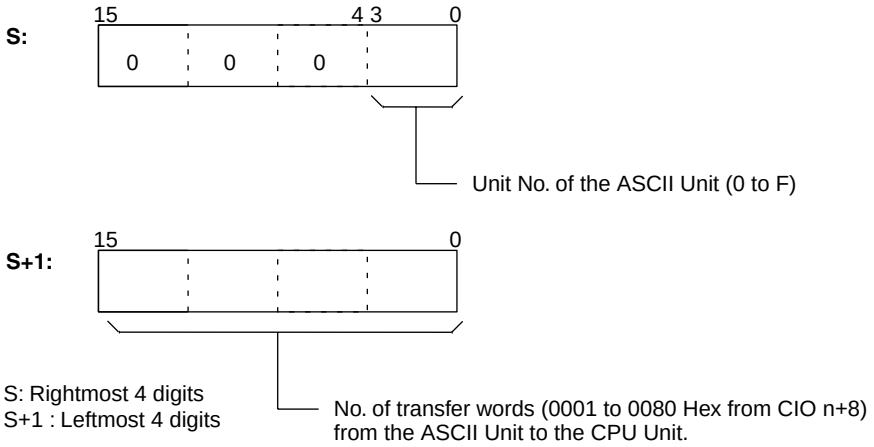
Value (4-digit hex.)	Details	Combination
#FD00	Transfers from the ASCII Unit's variable to the words specified in the allocated IR/CIO area words IR/CIO n + 8 and IR/CIO n + 9.	Must be used in combination with the ASCII Unit's PC QWRITE@ command.

C200HX/HG/HE

S:



CS1-series PCs



D: First destination word in CPU Unit's I/O memory from CIO n+8 and CIO n+9.

Data

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

Item	Detail
PC	CS1
Operand	D
CIO Area	0000 to 6143
Work Area	W000 to W511
Holding Bit Area	H000 to H511
Auxiliary Bit Area	A448 to A959
Timer Area	T0000 to T4095
Counter Area	C0000 to C4095
DM Area	D00000 to D32767
EM Area without bank	E00000 to E32767
EM Area with bank	En_00000 to En_32767 (n = 0 to C)
Indirect DM/EM addresses in binary	@D00000 to @D32767 @E00000 to @E32767 @En_00000 to @En_32767 (n = 0 to C)

Item	Detail
Indirect DM/EM addresses in BCD	*D00000 to *D32767 *E00000 to *E32767 *En_00000 to *En_32767 (n = 0 to C)
Constants	---
Data Registers	---
Directly addressing Index Registers	---
Indirect addressing using Index Registers	,IR0 to ,IR15 –2048 to +2047 ,IR0 to ,IR15 DR0 to 15, IR0 to IR15 ,IR0 to ,IR15+(++) – (– –) IR0 to IR15

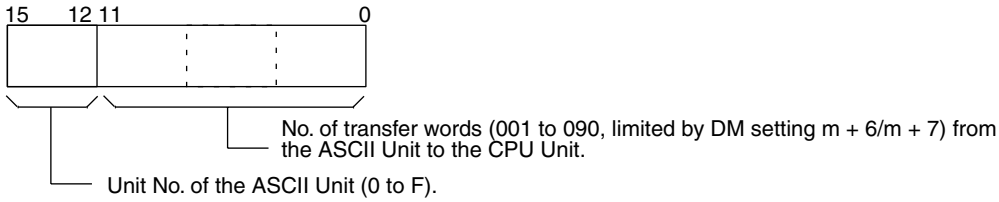
Reading from the ASCII Unit's Shared Memory

C:

Value (4-digit hex.)	Details	Combination
#00□□	Data in the specified number of words is transferred from the first word* + □□ in the IORD Area of the ASCII Unit's shared memory to the specified words in the CPU Unit memory. *Set in DM m + 7 word allocated in the DM Setup area.	May be used in combination with the ASCII Unit's PC EPUT command. May also be used on its own.

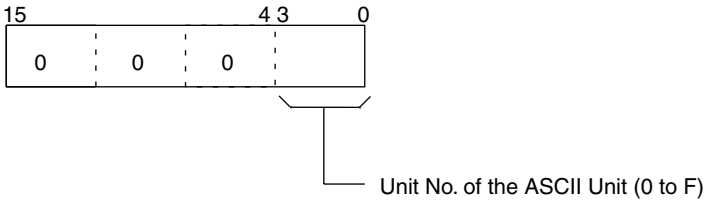
C200HX/HG/HE

S:

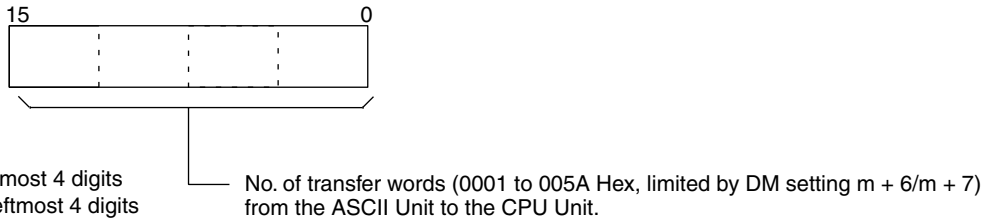


CS1-series PCs

S:



S+1:



D: First destination word in the CPU Unit's I/O memory.

Data

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

Item	Detail
PC	CS1
Operand	D
CIO Area	0000 to 6143
Work Area	W000 to W511
Holding Bit Area	H000 to H511
Auxiliary Bit Area	A448 to A959
Timer Area	T0000 to T4095
Counter Area	C0000 to C4095
DM Area	D00000 to D32767
EM Area without bank	E00000 to E32767
EM Area with bank	En_00000 to En_32767 (n = 0 to C)
Indirect DM/EM addresses in binary	@D00000 to @D32767 @E00000 to @E32767 @En_00000 to @En_32767 (n = 0 to C)
Indirect DM/EM addresses in BCD	*D00000 to *D32767 *E00000 to *E32767 *En_00000 to *En_32767 (n = 0 to C)
Constants	---
Data Registers	---
Directly addressing Index Registers	---
Indirect addressing using Index Registers	,IR0 to ,IR15 –2048 to +2047 ,IR0 to ,IR15 DR0 to 15, IR0 to IR15 ,IR0 to ,IR15+(++) – (– –) IR0 to IR15

Flags

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

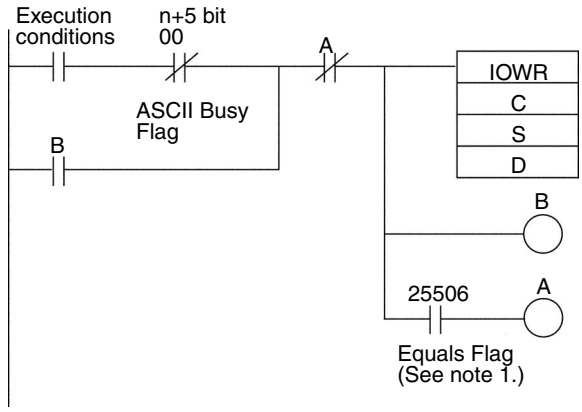
Note SR Area addresses for the C200HX/HG/HE, C200H, and C200HS are given in parentheses.

6-5-3 Using the IOWR and IORD Instructions

1,2,3...

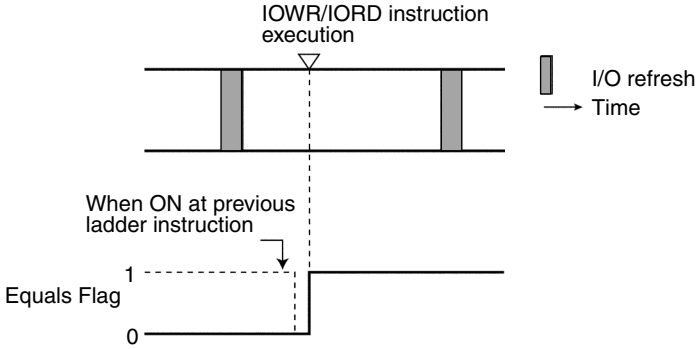
1. To allow retry processing if data transfer using the IOWR/IORD instructions fails, be sure to set a self-holding bit in the execution conditions and program the Equals Flag (SR 25506 for the C200HX/HG/HE) to release the bit when the Equals Flag is ON (1).
2. Program an NC condition for the ASCII Busy Flag (IR/CIO n+5 bit 00) as an execution condition so that the ASCII Unit does not execute an IOWR/IORD instruction during data transfer with the CPU Unit.

Note Only one IOWR/IORD instruction (control code #FD00) can be executed during a single cycle.

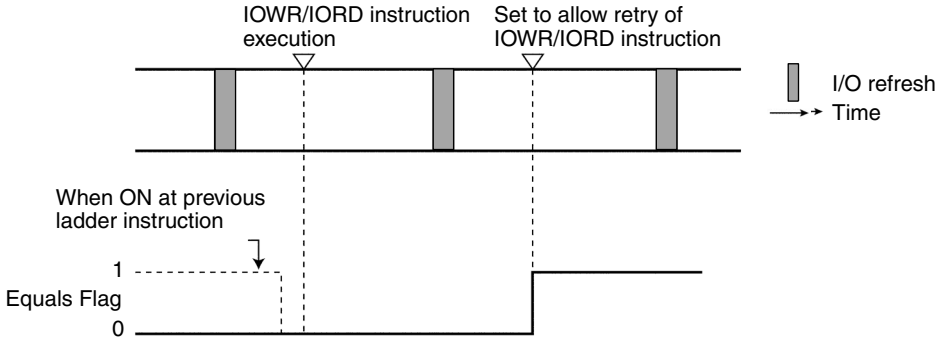


Note 1. The Equals Flag (SR 25506 for the C200HX/HG/HE) turns ON when processing to the ASCII Unit by IOWR/IORD instructions has finished normally. It remains OFF when processing to the ASCII Unit does not finish normally. After turning ON, it turns OFF at the execution of the next IOWR/IORD instruction. Since the Equals Flag will change at the execution of other instructions/commands as well (such as comparison instructions) be careful when writing programs.

Normal Completion



Abnormal Completion



2. The ASCII Busy Flag (IR/CIO n + 5 bit 00) turns ON when the ASCII Unit is exchanging data with the CPU Unit.

6-6 Restrictions for CS1-series PCs

There are five methods that can be used to transfer data between the CPU Unit and an ASCII Unit. Refer to *6-2 Selecting the Data Exchange Method* for details. When an ASCII Unit is used with a CS1-series PC, however, there are addressing restrictions for some of these methods, as shown in the following table.

Method	CPU Unit	ASCII Unit	Area restrictions
1	MOV and other instructions	PC WRITE or PC READ	Yes
2	No programming required	PC WRITE@ or PC READ@	Yes
3	MOV and other instructions	PC GET or PC PUT	None
4	IOWR (#00□□) or IORD(#00□□)	PC EGET or PC EPUT	None
5	IOWR (#FD00) or IORD(#FD00)	PC QWRITE@ or PC QREAD@	None, but dummy addresses may need to be used for PC QWRITE@ or PC QREAD@

The specific restrictions are as follows:

1,2,3...

1. Method 1: There are restrictions in the areas and words that can be addressed when using allocated words in memory to transfer data.
2. Method 2: There are restrictions in the areas and words that can be addressed when using PC WRITE@ or PC READ@ in the BASIC program in the ASCII Unit to transfer data.
3. Method 4, 5: IOWR and IORD can be used to transfer data to any memory area words in the CS1-series CPU Unit. Dummy addresses may have to be used in PC QWRITE@ or PC QREAD@.

Method 1: PC WRITE or PC READ

Any words that can be addressed using the allocated words n+3 and n+4 can be used for data transfer. Possible addresses are shown in the following table.

Area	CS1 address (see note)	Designation method	
		Area (bits 12 to 15 in word n+3, 1 BCD digit)	First word address (bits 00 to 15 in word n+4, 4 BCD digits)
DM Area	D00000 to D06655	0	0000 to 6655
CIO Area	CIO 0000 to CIO 0511 (except CIO 0281)	1	0000 to 0511 (except 0281)
Data Link Area (in CIO Area)	CIO 1000 to CIO 1063	2	0000 to 0063
Holding Bit Area, part 1	H000 to H099	3	0000 to 0099
Holding Bit Area, part 2	H100 (except H 101)	4	0000 (except 0001)
	H102 to H127		0002 to 0027
EM Area, bank 0	E0_00000 to E0_06143	5	0000 to 6143
Timer Area	T0000 to T0511	6	0000 to 0511

Note Other addresses cannot be accessed directly in CS1-series CPU Unit memory. Use methods 3 to 5 for other addresses.

Method 2: PC READ@ or PC WRITE@

The addresses shown in the following table can be specified for PC READ@ or PC WRITE@.

Area	CS1 address (see note)	Designation method	
		Code	First word address
DM Area	D00000 to D06655	@D	0000 to 6655
CIO Area	CIO 0000 to CIO 0511 (except CIO 0281)	@R	0000 to 0511 (except 0281)
Data Link Area (in CIO Area)	CIO 1000 to CIO 1063	@L	0000 to 0063
Holding Bit Area, part 1	H000 to H099	@H	0000 to 0099
Holding Bit Area, part 2	H100 (except H 101)	@A	0000 (except 0001)
	H102 to H127		0002 to 0027
EM Area, bank 0	E0_00000 to E0_06143	@E	0000 to 6143
Timer Area	T0000 to T0511	@G	0000 to 0511

Note Other addresses cannot be accessed directly in CS1-series CPU Unit memory. Use methods 3 to 5 for other addresses.

Method 3: PC GET or PC PUT

There are no restrictions.

Method 4: Reading/Writing Shared Memory in ASCII Unit Using PC EGET/PC EPUT or IOWR/IORD #00

There are no restrictions.

Note The number of read/write words for IOWR/IORD must be specified in hexadecimal for the CS1 Series.

Method 5: IOWR/IORD #FD00 and PC QREAD@/PC QWRITE@

There are no restrictions in the words that can be transferred and the words specified for the IOWR/IORD #FD00 instruction will take priority, The addresses specified for the PC QREAD@/PC QWRITE@ commands will be stored in words n+8 and n+9 for monitoring by the CPU Unit. If addresses are specified for IOWR/IORD #FD00 that cannot be specified for PC QREAD@/PC QWRITE@, then use dummy addresses in PC QREAD@/PC QWRITE@.

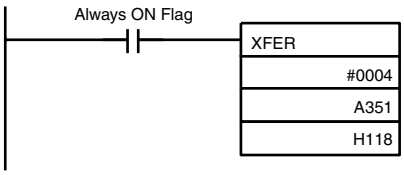
The CPU Unit will not monitor the specified words when dummy addresses are used.

Note The number of read/write words for IOWR/IORD must be specified in hexadecimal for the CS1 Series.

6-6-1 Using the Sync Time Set Bit for CS1-series PCs

The clock (RTC) in the ASCII Unit can be changed to the time set in the Calendar/Clock Area (AR 18 to AR 21) in C200HX/HG/HE CPU Units by turning ON the Sync Time Set Bit (word n, bit 05). With CS1-series PCs, however, the equivalent area (the Clock Data Area) is in words A351 to A354, so the clock will not be set correctly even if the Sync Time Set Bit is turned ON. To enable using the Sync Time Set Bit with CS1-series PCs, insert the following instructions into the program. (This requires that the ladder program is being exe-

cuted to use the Sync Time Set Bit, i.e., it cannot be used in PROGRAM mode.)



In the physical memory structure, words H118 to H121 in the CS1-series CPU Units correspond to words AR 18 to AR 21 in C200HX/HG/HE CPU Units. The above XFER instruction uses this information to enable using the Sync Time Set Bit with CS1-series PCs by transferring the time data in the normal CS1-series words to those used by the ASCII Unit (the time format is the same for both CS1-series and C200HX/HG/HE CPU Units).

SECTION 7
Editing BASIC Programs

This section describes the procedures for editing, saving, starting, and stopping BASIC programs. An overview of BASIC program configuration, language definitions, and the use of various BASIC commands, statements, and functions is provided as well.

7-1 Programming Procedure 112
- 7-1-1 Editing BASIC Programs 112
- 7-1-2 Saving BASIC Programs 116
- 7-1-3 Transferring BASIC Programs Between ASCII Unit and Terminal. 117

7-2 Character Variable Space Allocations 118

7-3 Starting/Stopping the BASIC Program 120
- 7-3-1 Starting the Program 120
- 7-3-2 Stopping the Program 120
- 7-3-3 Stopping the Program at a Specified Line Number 120
- 7-3-4 Continuing the Program after Stopping at a Line 120

7-4 Program Configuration 121
- 7-4-1 Command Execution Modes. 121
- 7-4-2 Programs, Lines, and Statements 121
- 7-4-3 Character Set. 123
- 7-4-4 Constants. 123
- 7-4-5 Variables 125
- 7-4-6 Variable Arrays 125
- 7-4-7 Type Conversion 126
- 7-4-8 Expressions 127

7-5 List of BASIC Commands 129

7-6 BASIC Commands 139

7-7 User-defined BASIC Functions. 231

7-8 Debugging 232
- 7-8-1 Overview. 232
- 7-8-2 Breakpoint Function (BRKPT). 232
- 7-8-3 Step Function (STEP) 233
- 7-8-4 Variable Monitor Function (WATCH) 233
- 7-8-5 Trace Function (TRON and TRACE) 234

7-1 Programming Procedure

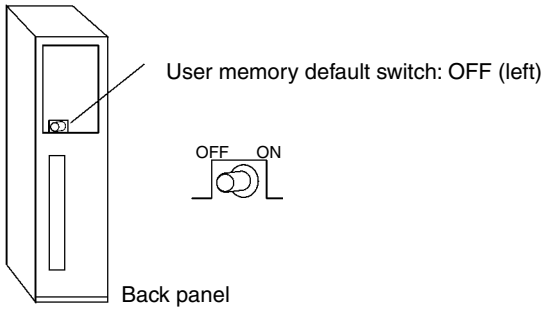
7-1-1 Editing BASIC Programs

The following procedure shows how to edit BASIC programs using Windows 95 Hyperterminal communications software on a personal computer used as a terminal.

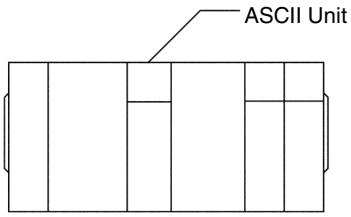
Connections and Default Settings

1,2,3...

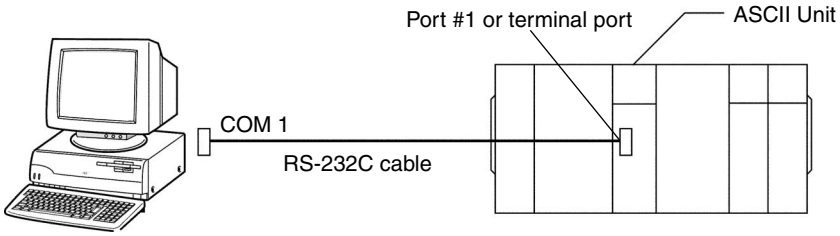
1. Make sure that the user memory default switch is turned OFF (set to the left). If the default switch is turned ON (set to the right) and power is turned ON, the memory will return to the default settings (be formatted). This switch should always be turned OFF, except when processing errors.



2. Mount the ASCII Unit to the Backplane.

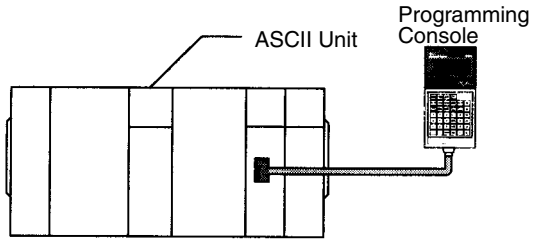


3. Turn OFF the power to the CPU Unit, and connect the RS-232C port (COM 1 in this example) to communications port #1 for the C200H-ASC11/ASC21 or the terminal port for the C200H-ASC31 using an RS-232C cable.



4. Set the switches on the front panel of the ASCII Unit as follows:
   - Unit No. switch.
   - Set the START/STOP switch to START.
   - Terminating resistor switch, 2-wire/4-wire switch. (Only set when using communications port #2 of the C200H-ASC21.)
5. Turn ON the power to the CPU Unit.
6. Using a PC peripheral device (such as a Programming Console), perform the following operations.
   a) Create an I/O table
   b) Set the DM Area of the CPU Unit (DM m to m + 7). (See *3-2 DM Area Allocations* for details.)

c) Once the settings are completed, turn ON the power, or using a PC peripheral device (such as Programming Console), toggle the ASCII Unit's Restart Bit from OFF to ON to OFF again.



Connecting to the Terminal

1,2,3...

1. In Windows 95, click **Start** and then point to **Programs**.
2. Point to **Accessories** and then click **Hyperterminal**.
3. Double click on the **Hyperterminal** icon.

   A message asking whether to install a modem will be displayed. Select "No."
4. The **Connection Settings** dialog box will be displayed. Enter your name.
5. The **Telephone No.** dialog box will be displayed. In the connection method field, select the personal computer's communications port number. Point to **Direct to COM 1**, and then click **OK**. The **COM 1 Properties** dialog box will be displayed.
6. In the **Port Settings** field, specify the following settings. (If the settings are already correct, then click **OK**.)

   Baud rate: 9,600 bps
   Data bits: 8
   Parity: None
   Stop bits: 2 bits
   Flow control: Xon/Xoff

   Note The default communications settings are 9,600 bps, 8-bit data, no parity, 2 stop bits, and no flow control for the ASCII Unit communications port #1 and the terminal port.
7. Point to **File** and then click **Properties**. The **[Your Name] Properties** dialog box will appear.
8. Select **VT100** in the **Emulation** field of the **Settings** sheet, then click **OK**.
9. Press the Ctrl+X Keys on the computer keyboard.
   When the following message is displayed at the terminal, connection is completed.

```
C200H-ASCII UNIT      Version 1.0x   1998/04/17
(c)Copyright OMRON Corporation 1998
READY
>
```

   Note The above message is displayed only when the Ctrl+X Keys are pressed for the first time after the CPU Unit power is turned ON. If the terminal is disconnected and then reconnected, only the prompt ">" will be displayed.
10. If the connection has not been successfully made, check the following.
    - Cable connections (whether the computer is connected to the port correctly)

- That the communications parameters are the same for the ASCII Unit and terminal (DM m + 2 or m + 4 contain the terminal communications parameters)
- If the ERR indicator on the front panel of the ASCII Unit is lit, indicating there is an error in word n + 7 (bits 00 to 11: Error Code, bits 12 to 15: Error Type), correct the error, reset the CPU Unit power supply, and then press the Ctrl+X Keys.

Refer to *SECTION 3 IR/CIO and DM Area Allocations* for specific word allocations.

The ">" prompt may not always be displayed properly after stopping a BASIC program to open a port using the Start/Stop Switch. If the prompt is not displayed, enter Ctrl + X before continuing terminal operations.

Inputting Programs

Perform the following steps to input a BASIC program directly into the ASCII Unit.

1,2,3...

1. Input **New** and press the Enter Key to clear the ASCII Unit's program.

   >NEW_

   Note The ASCII Unit has four program areas (No. 1 to 4) within the user memory area (RAM). Programs input using the default settings will be input in program area 1. To input a program into another area, switch to the desired program area executing the PGEN command as shown in the following example.

   Example:
   >PGEN 2 .... (Program area 2 will become the active program area.)

2. Input **Auto** and press the Enter Key to automatically display line numbers.

   >NEW
   >AUTO_

3. Input the BASIC program.

   Example:
   >NEW
   >AUTO
   10 A$=CHR$(67)
   20 PRINT A$
   30_

   To quit AUTO, press the Ctrl+X Keys.

   Note a) Execute the LIST command to display the BASIC program that has been written.

   b) Execute the PNAME command to name the program that has been written. See *7-6 BASIC Commands* for details on the PNAME command.
   Example: >PNAME "SAMPLE"
   (In this example, **SAMPLE** will be the program name.)

   c) Execute the PINF command to display the information on the ASCII Unit's user memory area (RAM). See *7-6 BASIC Commands* for details on the PINF command.

   Example: >PINF ALL

```
>PINF ALL
P1      :       [R/W]   76 bytes
P2      :       [R/W]    0 bytes
P3      :       [R/W]    0 bytes
P4      :       [R/W]    0 bytes
```

LIB	:	0 bytes
I–CODE	:	41 bytes

Transferring the BASIC Program Using a Text Editor

Perform the following steps to transfer a BASIC program to the ASCII Unit after compiling it using a text editor:

1,2,3...

1. Edit the program using a text editor such as the Note Pad, and save it in text format.
2. Set the START/STOP switch to STOP and input **New** to clear the ASCII Unit's program.
3. Input the LOAD command to put the ASCII Unit into file transmission wait state. The BASIC indicator (green) will flash quickly.
4. Point to **Hyperterminal** and then **Transfer,** and click **Transmit text file** to transmit the saved file to the ASCII Unit. The file will be stored in the active program No. area and overwrite any existing files.
5. Input Ctrl + C when the transfer has been completed. The prompt (>) will be displayed again.
   The T/R indicator will go out when the transfer has been completed.

Precautions

Be sure to use the flow control or send delay setting when transmitting files to the ASCII Unit using **Transmit text file**. Not doing so may cause errors in file reception.

Example: Setting Flow Control

At the terminal: In the **Port settings** field, specify **Xon/Xoff** for the flow control.

At the ASCII Unit: Set the Xon/Xoff control to ON when inputting the LOAD command.

LOAD #1, "TERM: 9600, 8, N, 2, CTS_OFF, RTS_OFF, DSR_OFF, XN_ON"

When executing a program, the ASCII Unit reads the source program and creates intermediate code. Memory is consumed by creating intermediate code. When writing comparatively large programs, occasionally execute RUN and check the memory consumption by executing PINF ALL.

Closing the Terminal

1,2,3...

1. To save the connection information (session), point to **File** and then **Save as...**.
2. To close the hyperterminal, point to **File** and then **Close Hyperterminal**. A message will be displayed asking whether to disconnect the modem. Click "OK."

Reconnection Procedures

1,2,3...

1. Double-click the icon (session file) that was created.

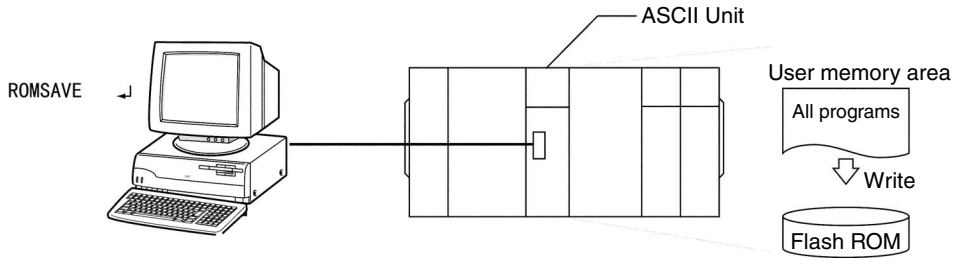
   or Point to **File** and then to **Open**. Select the session file.
2. A message will be displayed asking whether to install a modem. Select "No."
3. Enter the Ctrl+X Keys. If the previous message is displayed, the connection has been successfully made.

7-1-2 Saving BASIC Programs

Writing the BASIC Program to Flash ROM

The BASIC programs in the user memory (RAM) area can be written to the flash ROM in the ASCII Unit by entering the ROMSAVE command from the terminal.

>ROMSAVE



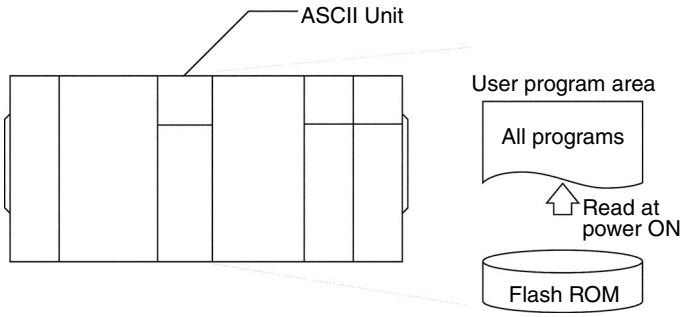
Reading the BASIC Program from Flash ROM

The BASIC programs in the flash ROM of the ASCII Unit can be read to the user memory (RAM) area of the ASCII Unit by entering the ROMLOAD command from the terminal.

>ROMLOAD

Automatically Reading the Program in Flash ROM at Power ON

The programs in the flash ROM can be automatically read to the user memory (RAM) area when power is turned ON or when the Unit is restarted, by setting bits 08 to 15 of word DM m to $5A_{hex}$ in the DM Setup Area allocated to the Unit.



⚠WARNING Never turn OFF the power supply of the PC while data is being written to flash memory by the ROMSAVE command until the completion display is shown at the terminal. By turning OFF the power supply, there is a possibility that the flash ROM will be electrically destroyed and that it will no longer be capable of storing data.

Do not turn OFF the power supply or restart the PC while data is being read from the flash ROM by the ROMLOAD command or automatically being transferred at startup. ROMLOAD command execution has been completed when the prompt appears on the terminal. Automatic transfer at startup has been completed when the command input prompt appears on the terminal. (Automatic transfer will be performed when power is turned ON or the Unit is restarted if the Startup Mode in the DM Area words allocated to the ASCII Unit is set for automatic transfer.)

Also, if the IOWR or IORD instruction is being used in the ladder program in the CPU Unit to exchange data with the ASCII Unit, always stop ladder program execution before writing to the flash ROM from the BASIC program using ROMSAVE.

⚠Caution Do not turn OFF

7-1-3 Transferring BASIC Programs Between ASCII Unit and Terminal

Transferring BASIC Programs to the Terminal

Use the following procedure to send BASIC programs from the user memory (RAM) of the ASCII Unit to the terminal.

Terminal: VT100 Mode (Windows: Hyperterminal)

1,2,3...

1. Connect the terminals and input “SAVE #1” or “SAVE #3” (for C200H-ASC31 only) and then press the Enter Key. (See note.)

   Example:

   SAVE #1, “TERM: 9600, 8, n, 2, CTS_OFF, RTS_OFF, DSR_OFF, XN_ON”
2. Enable the text data receiving mode at the terminal. (See note.)
3. When the terminal indicates that it is waiting to receive data, press the Ctrl+C Keys. When the ASCII Unit begins to transfer data, the T/R indicator will flash. (See note.)
4. Once the ASCII Unit has finished transferring all the data, end the text data receiving mode at the terminal.
5. Press the Ctrl+C Keys to display the prompt again.

Note If a password has been set in the BASIC program with the PWORD command, use the following procedure.

1,2,3...

1. Input “SAVE #1” or “SAVE #3” (for C200H-ASC31 only) and press the Enter Key. “PASSWORD” will be displayed at the ASCII Unit. Input the password.
2. Enable the text data receiving mode at the terminal.
3. When the terminal is ready to receive data, press the Ctrl+C Keys.
   Steps 4 and 5 are the same as for the above procedure.

Terminal: FIT10 Mode

1,2,3...

1. Set the START/STOP switch to STOP.
2. Input “SAVE #1” or “SAVE #3” (for C200H-ASC31 only) and press the Enter Key at the terminal. The ASCII Unit will be in a transfer wait state. This is indicated by the flashing BASIC indicator. (See note.)

   Example:

   SAVE #1, “TERM: 9600, 8, n, 2, CTS_OFF, RTS_OFF, DSR_OFF, XN_ON”
3. Enable the text data receiving mode at the terminal.
4. Set the START/STOP switch to START. When the ASCII Unit begins transferring data, the T/R indicator will flash.
5. Once the ASCII Unit has finished transferring all the data, set the START/STOP switch to STOP, and the prompt will appear again.
6. End the text data receiving mode at the terminal.

Note If a password has been set in the BASIC program, “PASSWORD” will be displayed at the ASCII Unit. Input the password and press the Enter Key. Once the ASCII Unit is in a transfer wait state, continue with the same procedure.

Writing Data from the Terminal to the ASCII Unit

To write BASIC programs from the terminal to the user memory (RAM) of the ASCII Unit, execute the LOAD command, and transfer the text file once the ASCII Unit is in a transfer wait state.

BASIC Program Transfer Commands

Operation		Command	Example
Transferring BASIC programs between the terminal and ASCII Unit	Terminal-to-ASCII Unit user memory (RAM)	LOAD	>LOAD #1
	ASCII Unit user memory (RAM)-to-terminal	SAVE	>SAVE #1
Transferring BASIC programs between flash ROM and the ASCII Unit	Flash ROM-to-ASCII Unit user memory (RAM)	ROMLOAD	>ROMLOAD
	ASCII Unit user memory (RAM)-to-flash ROM	ROMSAVE	>ROMSAVE

7-2 Character Variable Space Allocations

When executing the BASIC program in a C200H-ASC11/ASC21/ASC31 ASCII Unit, the method can be selected for allocating space for character variables to user memory. The following two methods are available.

Note For numeric variables, the size of the variable area is fixed based on the type, and is therefore allocated in user memory using the static (fixed) method outlined below.

Static (Fixed) Method

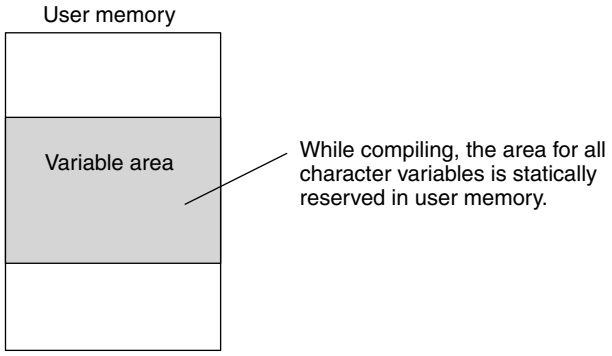
When compiling the BASIC program, space for all character variables in the BASIC program will be statically reserved in user memory.

Setting

At the beginning of the BASIC program, the maximum character variable content to be used in the BASIC program is specified using the OPTION LENGTH command. The maximum length can be specified from 1 to 255 characters.

Content

The space for character variables is reserved statically and therefore free memory cannot be generated as with the dynamic method. Even if not actually used, the space allocated for all variables will still be used.



Restrictions with this method are outlined below:

- The maximum length specified for the character variable content using the OPTION LENGTH command cannot be exceeded. If the maximum is exceeded, the amount of data over the limit will not be stored in the string variable "truncated."
- For the static method, once the variable type is specified under a particular variable name, that variable cannot be changed to a different type under the same name during execution of the program.
- For the static method, the DEF (variable type declaration) command cannot be executed more than once for the same variable name during the program.
- For the static method, at compile time, the memory size for the array variable is allocated statically, based on the number of subscripts. Therefore, only a fixed number of DIM command subscripts can be specified. Numeric expressions or numeric variables cannot be specified. (For the dynamic method, it is possible to alter the number of subscripts as required by using numeric expressions or numeric variables during the program to specify the number of subscripts.)

Dynamic (Free) Method

While executing the BASIC program, the space for each character variable in the BASIC program is reserved dynamically in user memory.

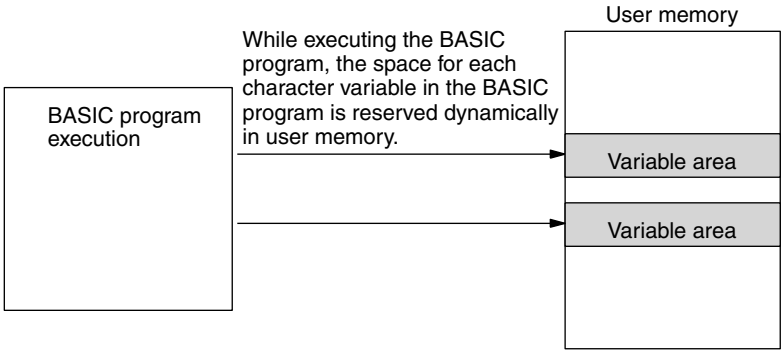
Setting

The OPTION LENGTH command is not used.

Content

Only the required amount of user memory is used. If the size of the variable becomes larger during the execution of the BASIC program, the space that had been reserved will be released once and then reserved again. (The released area will become free area of that size.)

If the allocation cannot be made when trying to reserve the space again, due to insufficient user memory capacity, an error will be generated and the program will be stopped. In such cases, it is necessary to change the program and the way in which the character variables are used, or use the ON ERROR command to free some memory.



Note The ASCII Unit is operated differently, depending on which of the two above methods is used. The differences are as follows:

- The compile time is longer for the static method than for the dynamic method.
- Conversely, when substituting character variables in the dynamic method, user memory release or allocation processing may be generated. Therefore, the execution time for commands is sometimes longer than for the static method.

- Place the PWORD command at the beginning of a statement if it is used with an OPTION LENGTH command.

7-3 Starting/Stopping the BASIC Program

7-3-1 Starting the Program

Selecting the Program Number

When transferring DM data allocated in the CPU Unit to the ASCII Unit at power up or start up, the start program number is specified in bits 00 to 07 of DM m in the DM Setup Area allocated to the Unit. Refer to *SECTION 3 IR/CIO and DM Area Allocations* for specific word allocations.

00: No. 1 01: No. 1

02: No. 2 03: No. 3

04: No. 4

Use the PGEN command to change the program number from the terminal.

Example: >PGEN 2 (The program area will switch to number 2.)

Starting the Program

1,2,3...

1. If starting from the terminal, make sure that the START/STOP switch on the front panel of the Unit is in START position, and input the RUN command from the terminal.
2. If using the START/STOP switch on the front panel of the Unit, toggle the START/STOP switch from STOP to START.
3. If starting automatically when power is turned ON or when the Unit is restarted, set the START/STOP switch on the front of the Unit to START, and set bits 08 to 11 of DM m + 1 to $5A_{hex}$ (automatic start) in the DM Setup Area allocated to the Unit. Refer to *SECTION 3 IR/CIO and DM Area Allocations* for specific word allocations.

7-3-2 Stopping the Program

Press the Ctrl+X Keys to stop the program from the terminal.

If the port connected to the terminal has been opened as a COMU device or an output device, the program cannot be stopped by pressing the Ctrl+X Keys.

or Turn OFF the START/STOP switch on the front panel of the Unit.

7-3-3 Stopping the Program at a Specified Line Number

Perform one of the following steps to stop the program from the terminal.

- Input RUN TO and the line number, and run the program.
- Input the BRKPT command, set the break point, and run the program.
- Input the STEP command and only run the specified lines.

or In the program, set a STOP command at the specified line number before running the program.

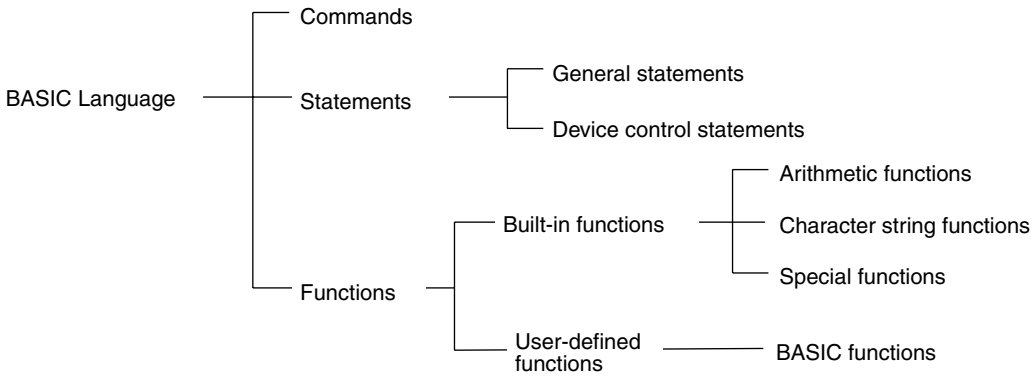
7-3-4 Continuing the Program after Stopping at a Line

1,2,3...

1. Input the CONT command from the terminal.
2. Input the STEP command for the desired number of lines.

7-4 Program Configuration

A BASIC program consists of commands, statements, and functions.



Basic Statements designate and control the flow of programs and are generally used in program lines within a program. Statements are executed by the RUN command and can be executed directly from a terminal without line numbers.

Basic Commands are usually entered from the command line and control operations external to the program, such as printing and listing. Commands are executed when the ASCII Unit is in Command Mode. They cannot be used in the program and executed with the RUN command.

Functions are self-contained programs which accept one or more arguments, perform predefined calculations, and return a result(s). There are predefined BASIC functions for arithmetic and string operations as well as user-defined functions. Functions cannot be used on their own, but must be combined with statements.

7-4-1 Command Execution Modes

There are two modes for executing commands.

Command Mode

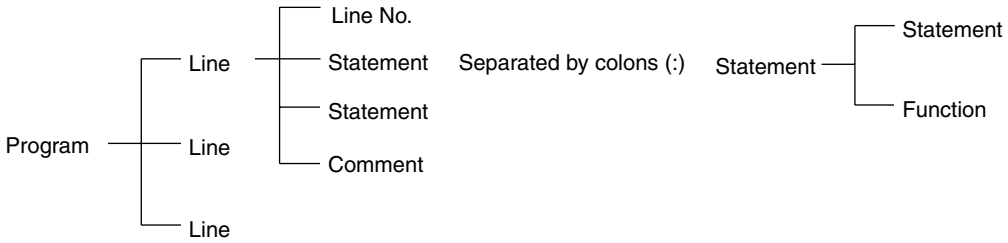
By entering a command and pressing the Enter Key when “>” is displayed on the terminal screen, the command will be immediately executed. The command input standby condition is called Command Mode.

Program Mode

A line is created by inputting a BASIC command with a line number at the beginning and pressing the Enter Key. A program consists of a collection of these lines. The execution of commands using these programs is called Program Mode.

7-4-2 Programs, Lines, and Statements

A BASIC program consists of a collection of lines. A line is made up of one or more statements, and a statement can contain a function.



Line Numbers

- Every BASIC program line begins with a line number. (When there is no line number, the program is executed directly.)

- All inputs up until the Enter Key is pressed are stored in the user memory as a single line, in the order of line numbers. Execution is carried out in the order of line numbers. Line numbers are also used as references for branching destinations such as GOSUB and GOTO, and arguments in LIST and DEL.
- Line numbers must be between 0 and 65535. Each line is written in ascending order of magnitude of the line numbers.
- A line consists of up to 255 characters (including the line number and spaces).

Line number	[Label:]	Statement	:	Statement	:	['Comment]	CR

Note A period (.) may be used in DELETE, LIST, and EDIT commands to refer to the current line number of a program (the most recently entered or displayed line number).

Examples of the use of the period:

```
LIST.
EDIT.
DEL 100–.
```

Labels

Labels are used in place of line numbers in GOTO and GOSUB commands. At the point being labeled, alphabetic characters must follow an asterisk (*) as the line's first statement. The maximum length of a label is 16 characters, including the asterisk. Text strings that begin with reserved words cannot be used. When characters continue after a label in a line, they are separated by a colon (:). When two identical label names exist, the label in the line with the lowest line number is referred to.

Example

```
10 ON COM 1 GOSUB *ABC
                   └─Label

100 *ABC
     └─Label
```

Note If the line number of the branch destination is specified, the line number will also have to be corrected when editing the program. By using labels, it is not necessary to correct the line number, making program revisions quicker and more effective. If text strings starting with reserved words are used as labels, a SYNTAX ERROR will occur.

Statements

- Statements can occur alone or they can contain functions.
- If there is more than one statement on the same line, separate the statements with a colon (:).

Example:

```
10      FOR L=1 TO 100:   J=L*I:      PRINT J:      NEXT L
Line No. Statement 1      Statement 2 Statement 3   Statement 4
```

Comments

Input comments after single quotations ('').

Note Comments cannot be used in lines containing the DATA command.

7-4-3 Character Set

The BASIC character set comprises alphabetical characters, numeric characters, and special characters.

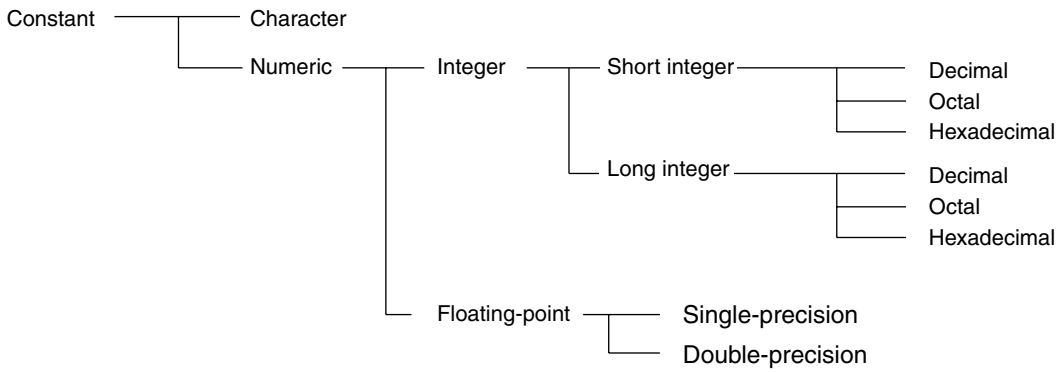
The alphabetic characters in BASIC are the uppercase and lowercase letters of the alphabet. The numeric characters in BASIC are the digits 0 through 9.

The following special characters are recognized by BASIC:

SP (space) ! ” # $ & ’ ( ) * + , – . / : ; < = > ? [\ ] ^ _

7-4-4 Constants

The following can be used as constants.



Character Constants

A character constant is a character string enclosed by double quotation marks (”). It can be up to 255 characters long. If it contains no characters, it is called an “empty character string” or the null string.

Example: “CF–BASIC”

Integer Constants

There are two types of integers:

- Short integers (16-bit)
- Long integers (32-bit)

Decimal Constants

Whole numbers between –32,768 and 32,767 can be used for short integers, and whole numbers between –2,147,483,648 and 2,147,483,647 can be used for long integers. An optional percent sign (%) for an integer, and an “and” sign (&) for a long integer can be added to specifically indicate an integer constant. Integer constants do not have decimal points.

Example: 1234 –1234 12% (integer) 12345678& (long integer)

Octal Constants

Octal numbers 0 to 7 beginning with the prefix “&” and between &0 and &177777 can be used for integers. For long integers, between &0 and &37777777777 can be used.

Example:& 0127 &7777

Hexadecimal Constants

Hexadecimal numbers 0 to F (0 to 9, A to F) beginning with the prefix “&H” and between &H0000 and &HFFFF can be used for integers. Between &H0 and & HFFFFFFFF can be used for long integers.

Example: &H5E &HBF4

Floating Point Constants

These are positive or negative numbers that are expressed in exponential format. E (single-precision) or D (double-precision) is entered after the mantissa, and then the exponent is entered.

Example: For the value 12,300,000, the following equations apply.

$= 1.23 \times 10{,}000{,}000$
$= 1.23 \times 10^7$
= 1.23E + 7 (single-precision)
= 1.23D + 7 (double-precision)

1.23 is the mantissa and E + 7 or D + 7 is the exponent.

Single Precision

This type of constant is stored with seven-digit precision and is output as a six-digit constant with the seventh digit rounded off within the range of $\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{-38}$. It is represented by one of the following methods:

- As a number with six or fewer digits: 1234.5
- As a number in exponential form using E: –1.2E+3
- As a number with the character "!" at the end: 2.34!

Double Precision

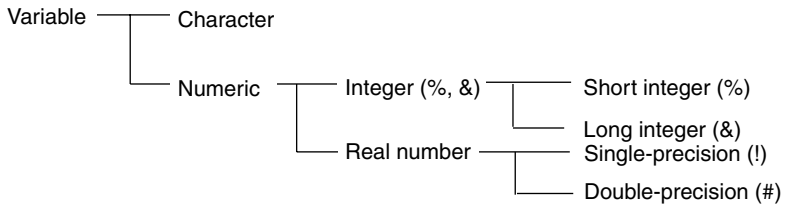
This type of constant is stored with 16-digit precision and is output with up to 15 digits or less with the 16th digit rounded off within the range of $\pm 2.23 \times 10^{-308}$ to $\pm 1.79 \times 10^{-308}$. It is represented by one of the following methods:

- As a number with seven or more valid digits: 1.23456789
- As a number in exponential form using D: –1.2D–3
- As a number with the character "#" at the end: 2.34#

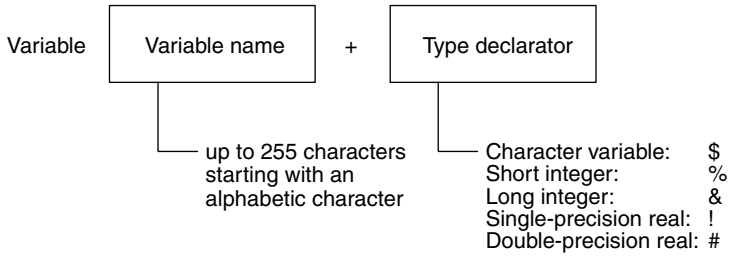
Note Precautions on Real Numbers:
A real number expressed in the floating point format is advantageous in allowing a wide range of numbers to be handled. However, not all numbers are necessarily expressed correctly due to the nature of the binary expression and as a result, some of them are expressed in approximate values. Therefore, most of the values and calculation results stored in memory contain some amount of error. Although this error is normally so small that it cannot cause any problem, it may be interpreted as a large error depending on the application.
For example, when the number is truncated below the decimal point after multiplying "0.1" by 10, the result of "1" will be expected. However, "0.1" is stored in the memory as an approximate value of "0.0999..." because it cannot be expressed in the binary format. When this value of "0.0999..." is multiplied by 10, the result will be "0.999..." When the number is truncated below the decimal point, the result will be "0".
The solutions to this problem vary depending on the application. When reviewing design proposals, check if a desired operation can be achieved by using the integer type or if desired integers can be obtained by rounding the calculation results.

7-4-5 Variables

Variables are names used to represent values that are used in a BASIC program. The value of a variable may be assigned as the result of calculations or explicitly by the programmer with an assignment statement. If no value is assigned to a numeric variable, it is assumed to be zero. If no value is assigned to a character variable, it is assumed to be the null string.



Variables are made up of the variable name and the type declarator.



Variable Name

A variable may be up to 255 alphanumeric characters long. All characters are valid. No variable can start with “FN” or a BASIC command name.

If a parameter begins with a reserved word, a SYNTAX ERROR will occur. TOTAL and ABSOL, for example, cannot be used because they include reserved words TO and ABS.

Type Declarator

The variable TYPE can be declared. This is done using a type declarator that is placed after the variable name. Even if two variables have the same name, they will be treated differently if they are declared as different types of variables.

%	Short Integer	Uses 2 bytes per variable.	Example: A%
&	Long integer	Uses 4 bytes per variable	Example: A&
!	Single-precision real	Uses 4 bytes per variable.	Example: A or A!
#	Double-precision real	Uses 8 bytes per variable.	Example: A#
$	Character	Uses up to 255 characters.	Example: A$

Note There is a second way to declare variable types. The BASIC statements DEFINT, DEFLNG, DEFSTR, DEFSNG, and DEFDBL may be used to declare the types for certain variable names, starting with a certain letter. By default, all variables are declared as single-precision.

7-4-6 Variable Arrays

An array is a group of values of the same TYPE (either numeric or alphabetic) that is stored and referenced as a unit by the same variable name. Each element in an array has a unique position and is referenced by the name of the array subscripted with an integer or integer expression.

There can be many dimensions to an array. The most common types are one-, two-, and three-dimensional arrays. The number of dimensions of an

array and the length of its name are dictated by the amount of available memory. An array has one subscript for each dimension in the array.

For example, T(4) would reference the fourth element in the one-dimensional array T. R(2,3) would reference the value located in the second row and third column of the two-dimensional array R.

The maximum number of dimensions of an array is 255. The maximum number of elements per dimension is 32767. The array size and number of dimensions must be declared with the DIM statement. The subscript value zero is the position of the first element in an array. All elements of an array must be of the same TYPE. When an array is declared, numeric arrays are initialized as zero, and alphabetic arrays are initialized as null strings. When a variable array is used without being declared, it will be declared as a one-dimensional array with elements 0 to 10, or 1 to 10 depending on the option base.

Note In this example, the option base = 0

Example:
DIM A$(3) This array is the one-dimensional, alphabetic A$ variable array. The following 4 variables can be used.

A$(0)	A$(1)	A$(2)	A$(3)

DIM B(2,3) This array is the two-dimensional, numerical B variable array. The following 12 variables can be used.

B(0,0)	B(1,0)	B(2,0)
B(0,1)	B(1,1)	B(2,1)
B(0,2)	B(1,2)	B(2,2)
B(0,3)	B(1,3)	B(2,3)

7-4-7 Type Conversion

When necessary, BASIC will automatically convert a numeric constant from one TYPE to another. The following rules apply.

1,2,3...

1. If the numeric data on the right side of an assignment statement differs from the type of data on the left side, the right side is converted to match the left. Character data, however, cannot be converted to numerical data, or vice versa without using functions (VAL, STR$)

   Example:
   A = 12.3: if A is an integer, then “12” is assigned to A.

2. Double-precision data is converted to single-precision data when assigned to a single-precision variable.

   Example:
   If “A” is a single-precision variable and the statement:

   LET A = 12.3456789# occurs in a program, then 12.3456789# will be converted to a single-precision number and then assigned to “A.” (A value rounded to 6 digits will be displayed.)

3. When an arithmetic operation is performed using both single-precision and double-precision values, the single-precision value is converted to double-precision first, and then the operation is performed. Therefore, the result is a double-precision value.

   Example:
   10#/3 (double-precision)

4. In logic operations, all numeric data is first converted into long-integer data unless both inputs are short integers. If any value cannot be converted into

a long integer within the range of –2147483648 to 2147483647, an overflow error will occur.

Example:
LET A = NOT 12.34, –13 is assigned as A.

5. When a real number is converted into an integer, the value is rounded.

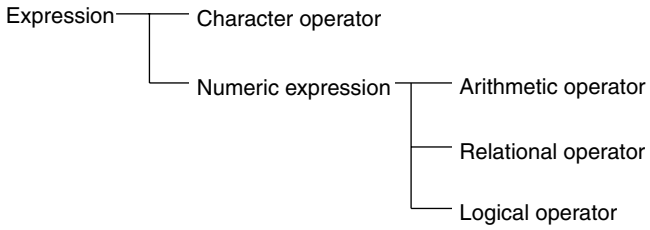
   Example:
   A = 12.3: “12” is assigned to A.
   B = 12.6: “13” is assigned to B.

7-4-8 Expressions

Expressions refer to constants, variables, and functions that have been combined by operators. Numeric values, variables, or characters alone can also form expressions. There are four types of expressions.

- Arithmetic
- Relational
- Logical
- Character

Of these, the first three produce numeric values as a result and are thus called “numeric expressions.” The last type is called a “character expression.”



Character Operators

A character expression is made up of character constants and variables that are linked with the character operator “+.” Instead of adding characters together, the “+” operator links the characters together to form one character string.

Input: A$ = “CF”
B$ = “BASIC”
PRINT A$ + “–” + B$

Output: CF–BASIC” is displayed.

Arithmetic Operators

An arithmetic expression is made up of constants, variables, and functions combined using arithmetic operators. A list of valid arithmetic operators is shown in the following table.

Arithmetic operator	Example	Operation
+	A + B	Addition
–	A – B, –A	Subtraction or negation
*	A * B	Multiplication
/	A / B	Real number division
\	A \ B	Integer division
MOD	A MOD B	Remainder after integer division
^	A ^ B	Exponentiation

If A or B is a real number in an expression using the \ or MOD operator, the decimal part is first rounded up to convert the real number into an integer, and then the operation is performed.

Relational Operators

Relational operators compare two values. The output is "–1" (&HFFFF) if the two values are equal and "0" if they are not.

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

Logical Operators

Logical operators perform tests on multiple relations, bit manipulations, or Boolean operations. The logical operator returns a bitwise result that is either "true" (not 0) or "false" (0). In an expression, logical operations are performed after arithmetic and relational operations. The outcome of a logical operation is determined as shown in the following table. The operators are listed in the order of precedence.

Logical Operator	Description, Example, and Result	
NOT (negation)	A	NOT A
	1 0	0 1
AND (logical product)	A B	A AND B
	1 1 1 0 0 1 0 0	1 0 0 0
OR (logical sum)	A B	A OR B
	1 1 1 0 0 1 0 0	1 1 1 0
XOR (exclusive-OR)	A B	A XOR B
	1 1 1 0 0 1 0 0	0 1 1 0
EQV (equivalence)	A B	A EQV B
	1 1 1 0 0 1 0 0	1 0 0 1
IMP (implication)	A B	A IMP B
	1 1 1 0 0 1 0 0	1 0 1 1

7-5 List of BASIC Commands

The following table lists all BASIC commands alphabetically and gives an outline for each. See *7-6 BASIC Commands* for further details on each command.

Square brackets ([and]) in the syntax column indicate items that can be omitted. Normal brackets ({ and }) indicate items that can be repeated. (See *7-6 BASIC Commands* for further details on syntax.)

Command	Syntax	Details	Command	Statement	Function	Page

@IF	@IF <numeric expression>
	THEN
	<program>
	ELSE
	<program>
	ENDIF
		Selects either THEN and the following characters, or ELSE and the following characters, based on the results of a numeric expression.		General		138

ABS	ABS (numeric expression)	Determines the absolute value of the value given by the argument.			Arithmetic operation	139
ACOS	ACOS (numeric expression)	Determines the arc cosine of the value given by the argument.			Arithmetic operation	139
ALARM ON/ OFF/STOP	ALARM ON or ALARM OFF or ALARM STOP	Enables, disables, or stops a one-shot timer (ON ALARM) interrupt.		General		140
ASC	ASC (character expression)	Determines the ASCII code of the first character of the character string given by the argument.			Character string	140
ASIN	ASIN (numeric expression)	Determines the arc sine of the value given by the argument.			Arithmetic operation	141
ATN	ATN (numeric expression)	Determines the arc tangent of the value given by the argument.			Arithmetic operation	141
AUTO	AUTO [line number][increment]	Automatically generates numbers at the beginning of lines (if not specified, increment = 10)	Command			141
BITOFF/BITON	BITOFF <integer variable>, <bit position> BITON <numeric variable>, <bit position>	Turns OFF or ON bit position (0 to 31) of integer numeric variables.		General		142
BRKPT	BRKPT [SET <line number> or BRKPT [DEL <line number> or BRKPT [DEL ALL] BRKPT	Sets (SET), deletes (DEL), and displays a list of breakpoints when no arguments are given.	Command			142
CDBL	CDBL (numeric expression)	Converts a value to a double-precision value real number.			Arithmetic operation	143
CHR$	CHR$ (numeric expression)	Finds a character that corresponds to the ASCII code (integer) given by the argument.			Character string	143
CINT	CINT (numeric expression)	Rounds off and changes into a short integer			Arithmetic operation	144

Command	Syntax	Details	Command	Statement	Function	Page
CLEAR	CLEAR	Initializes numeric and character variables and closes all ports.		General		144
CLNG	CLNG (numeric expression)	Converts a value to a long integer.			Arithmetic operation	144
CLOSE	CLOSE [#port number]	Closes a port.		Device control		145
CLS	CLS [#port number]	Clears the screen.		Device control		145
COM ON/OFF/ STOP	COM [port number] ON or COM [port number] OFF or COM [port number] STOP	Enables, disables, or stops a communication interrupt (ON COM).		General		145
CONT	CONT	Resumes execution of a program that has stopped.	Command			146
COS	COS (numeric expression)	Determines the cosine of the value given by the argument.			Arithmetic operation	146
CSNG	CSNG (numeric expression)	Converts to single-precision real number.			Arithmetic operation	146
CTS	CTS (port number)	Reads CS signal status.			Special	147
DATA	DATA [Constant{,constant}]	Defines the numeric and character constants assigned using the READ command.		General		147
DATE$	DATE$ [="YY/MM/DD"]	Sets or gives date.			Special	147
DAY	DAY [=numeric expression]	Sets or gives day.			Special	148
DEF FN	DEF FN Name [(argument[,argument,...])]= expression	Defines a BASIC function written by the user.		General		148
DEFINT/ DEFLNG/ DEFSNG/ DEFDBL/ DEFSTR	DEFINT character [–character]{character [–character]} or DEFLNG character [–character]{character [–character]} or DEFSNG character [–character]{character [–character]} or DEFDBL character [–character]{character [–character]} or DEFSTR character [–character]{character [–character]}	Declares variable types as integer, long integer, single-precision, double-precision or character.		General		149
DEL	DEL [line number 1] [–[line number 2]]	Deletes part of the program line.	Command			149
DIM	DIM variable name (maximum subscript) {maximum subscript})){variable name(maximum subscript{maximum subscript})}	Specifies the array variable dimensions and the maximum value for the subscript. For character string variables, the maximum number of characters can be specified.		General		150
DMA	DMA (port number)	Determines transmitting/ not transmitting for DMA data transmission.			Special	151

Command	Syntax	Details	Command	Statement	Function	Page
DSR	DSR (port number)	Reads status of DR signal.			Special	151
DTR	DTR (port number) SET or DTR (port number) RESET	Sets/resets ER signal.		Device control		151
EDIT	EDIT [Line number]	Edits one line of the program.	Command			152
END	END	Terminates program execution and closes all ports.		General		153
EOF	EOF (port number)	Checks if the reception port buffer for the specified port is empty.			Special	153
ERC	ERC	Deletes the error code in the Error Log Table, or deletes the error status being generated.		General		153
ERL	ERL	Determines the line number being executed at the point that an error is generated.			Special	154
ERR	ERR	Determines the error code generated by the error.			Special	154
ERROR	ERROR <error number>	Simulates the error generation.		General		155
EXP	EXP (numeric expression)	Determine the exponential of a numeric expression.			Arithmetic operation	155
FCS	FCS (character expression, [FCS type])	Determines the error check code for the character expression.			Arithmetic operation	156
FIX	FIX (numeric expression)	Determines the integer part of value given by the argument.			Arithmetic operation	157
FOR ... TO ... STEP TO NEXT	FOR <variable> = <numeric expression> TO <numeric expression> [STEP <numeric expression>]: <program> NEXT variable {,variable} or FOR <variable> = <numeric expression> TO <numeric expression> [STEP <numeric expression>] line end program NEXT variable {,variable}	Repeats the loop from FOR to NEXT a specified number of times.		General		158
FRE	FRE	Assigns the amount of unused user memory.			Special	158
GOSUB TO RETURN	GOSUB line number to RETURN or GOSUB label to RETURN	Calls up and executes subroutines, and returns on RETURN command.		General		159
GOTO	GOTO <line number> or <label>	Branches to specified line number or label.		General		159

Command	Syntax	Details	Command	Statement	Function	Page
HEX$	HEX$ (numeric expression)	Changes the decimal value given by the argument to a hexadecimal character string.			Character string	160
IF	IF <numeric expression> THEN[statement{:statement} or <line number> or <label>] GOTO <line number> or <label> [ELSE[statement {:statement} or <line number> or <label>]}	Selects an executable statement or destination based on the results of a numeric expression.		General		161
INKEY$	INKEY$ [#port number]	Finds the characters that have been newly received by the reception buffer of a specified port.			Special	161
INPUT	INPUT[;] [#port number,] [“prompt”;] variable{,variable} INPUT [#port number,] [“prompt”,] variable {,variable}	Reads data separated by carriage returns from the reception buffer of a specified port into a specified variable.		General		162
INPUT$	INPUT$ (number of characters [,#port number])	Reads a string of characters of a specified number of characters from reception buffer of specified port.			Special	163
INSTR	INSTR ([search start position,] <character expression 1>, <character expression 2>)	Searches for character expression 2 in character expression 1, and obtains the position of the first character.			Character string	163
INT	INT (numeric expression)	Rounds down a decimal fraction to the nearest integer.			Arithmetic operation	164
INTRB	INTRB	Determines the line number being executed when an interrupt is generated.			Special	164
INTRR	INTRR	Determines the original interrupt type in an interrupt subroutine.			Special	165
INTRS	INTRS	Determines the line number that defined the generated interrupt.			Special	165
KEY ON/OFF/ STOP	KEY [key number] ON or KEY [key number] OFF or KEY [key number] STOP	Enables, disables or stops a key input interrupt (ON KEY).		General		166
LEFT$	LEFT$ (<character expression>, <character number>)	Finds the specified number of characters beginning from the leftmost character of the character string.			Character string	166
LEN	LEN (character expression)	Determines the number of characters in a character string.			Character string	166
LET	[LET] <variable name>=<expression>	Assigns the value of an expression to a variable name.		General		167

Command	Syntax	Details	Command	Statement	Function	Page
LINE INPUT	LINE INPUT[;] [#port number,] [“prompt”;] character variable	Reads one line of data from the reception buffer of a specified port into the character variable.		General		167
LIST	LIST [line number 1] [–line number 2]	Displays the program.	Command			168
LLIST	LLIST [line number 1] [–line number 2]	Outputs the program to the printer.	Command			169
LOAD	LOAD #<port number> [communications conditions character expression]	Transfers a program from the terminal to the ASCII Unit user memory.	Command			170
LOC	LOC (port number)	Provides the number of bytes of data stored in the reception buffer of a specified port.			Special	170
LOG	LOG (numeric expression)	Determines a natural logarithm.			Arithmetic operation	170
LPRINT [USING]	LPRINT [#port number] [USING print format;] expression {[,]expression} or LPRINT [#port number] [USING print format;] expression {[;]expression} or LPRINT [#port number] [USING print format;] expression { [space]expression}	Assigns LPRT as the device symbol (using the specified format), and outputs the expression value to a specified port.		General		171
MID$	MID$ (<character expression>, <character position> [number of characters])	Fetches the specified number of characters from a specified character position.			Character string	172
MID$	MID$ (<character expression>, <character position> [,number of characters])=<character expression>	Replaces the number of characters from a specified character position with specified character string.		General		173
MODEL	MODEL	Determines ASCII Unit model.			Special	173
NEW	NEW	Deletes the program currently being used and clears all variables.	Command			174
OCT$	OCT$ (numeric expression)	Converts decimal value to an octal character string.			Character string	175
ON ALARM	ON ALARM <time expression> GOSUB <line number> or ON ALARM <time expression> GOSUB <label>	Branches to interrupt subroutine after one-shot timer.		General		176
ON COM GOSUB	ON COM [port number] GOSUB <line number> or ON COM [port number] GOSUB <label>	Branches to interrupt subroutine when interrupted by specified port.		General		177

Command	Syntax	Details	Command	Statement	Function	Page
ON ERROR GOTO LINE NUMBER	ON ERROR GOTO <line number> or ON ERROR GOTO <label>	Branches to interrupt subroutine when error is generated.		General		178
ON NUMERIC EXPRESSION GOSUB OR GOTO	ON <numeric expression> GOSUB <line number> {,<line number> or <label>} or ON <numeric expression> GOSUB <label> {,<line number> or <label>} or ON <numeric expression> GOTO <line number> {,<line number> or <label>} or ON <numeric expression> GOTO <label>{,<line number> or <label>}	Branches to specified line number based on value of numeric expression.		General		179
ON KEY GOTO OR GOSUB	ON KEY [key number] GOTO <line number> or <label> or ON KEY [key number] GOSUB <line number> or <label>	Branches to interrupt subroutine on specific key input.		General		180
ON PC GOSUB	ON PC [interrupt number] GOSUB <line number > or ON PC [interrupt number] GOSUB <label>	Branches to interrupt subroutine when interrupted from PC.		General		181
ON TIME$	ON TIME$= <time character expression> GOSUB <line number> or <label>	Branches to interrupt subroutine at a certain time.		General		182
ON TIMER	ON TIMER <time expression> GOSUB <line number> or <label>	Branches to interrupt subroutine after interval time has expired, and repeats this process.		General		183
OPEN	OPEN #<port number>; “device symbol :[baud rate[,data length[parity[stop bit[,CS_ON or OFF[,RS_ON or OFF[,DS_ON or OFF[,XN_ON or OFF]]]]]]]]”	Opens a specified port under specified conditions (device type, communications conditions, control signal operation)		Device control		184
OPTION BASE	OPTION BASE 0 or OPTION BASE 1	Designates the first number of the array variable subscript as 1 or 0.		General		186
OPTION LENGTH	OPTION LENGTH <numeric expression>	Sets the maximum length (1 to 255) for character string variable content.		General		187
PC EGET	PC EGET #<shared memory address>, <read number of words>,“format{,format}”; variable{,variable}	Reads data from shared memory of ASCII Unit itself to the variables.		General		188

Command	Syntax	Details	Command	Statement	Function	Page
PC EPUT	PC EPUT #<shared memory address>, <write number of words>,“format{,format}”; <numeric expression> {<numeric expression>}	Writes shared memory data of ASCII Unit itself.		General		189
PC GET	PC GET variable name 1 [,variable name 2]	Reads the output data value from the PC’s allocated I/O to the variables.		General		190
PC ON/OFF/ STOP	PC [interrupt number] ON or PC [interrupt number] OFF or PC [interrupt number] STOP	Enables, disables, or stops a PC interrupt.		General		187
PC PUT	PC PUT <numeric expression>	Writes the numeric expression value to the input data area of the allocated I/O in the PC.		General		190
PC QREAD@	PC QREAD”@area, address, <read number of words>, PC format {,PC format} ,variable {,variable}	Generates a request to execute an IOWR instruction at the PC, and reads the specified I/O memory area.		General		191
PC QWRITE@	PC QWRITE”@area, address, <write number of words>, PC format {,PC format}, numeric expression {,numeric expression}	Generates a request to execute an IORD instruction at the PC, and writes data to the specified I/O memory area.		General		193
PC READ OR PC READ@	PC READ” PC format {,PC format}”; variable {,variable} or PC READ” @ area, address, <read number of words>, PC format {,PC format}”; variable {variable}	Converts the format of data written from the PC and reads it.		General		195
PC WRITE (“@”)	PC WRITE” PC format {,PC format}”; variable {,variable} or PC WRITE” @area, address, <write number of words>, PC format {,PC format}”; variable {variable}	Converts the format of the numeric expression value and transfers it to the PC.		General		200
PEEK	PEEK (address)	Reads the contents of a specified memory address.			Special	205
PGEN	PGEN <numeric expression>	Switches to the program area number to be used.	Command			205
PINF	PINF [ALL] or PINF [Program No.]	Displays information on the user memory area.	Command			206
PMEM	PMEM ON or PMEM OFF	Switches between enabling and disabling write protection for the user memory area.	Command			207
PNAME	PNAME “program name”	Assigns or cancels a program name.	Command			207
POKE	POKE address,value	Writes data to specified address in the user memory.		General		208

Command	Syntax	Details	Command	Statement	Function	Page
PRINT [USING]	PRINT [#port number] [USING print format;] expression {[,]expression} PRINT [#port number] [USING print format;] expression {[;]expression} or PRINT [#port number] [USING print format;] expression {[space] expression}	Outputs expression value to specified port (in the specified format).		General		208
PWORD	PWORD <character expression>	Assigns a password to the BASIC program area.		General		210
RANDOM	RANDOM (expression)	Changes a random sequence based on a random number seed given by the expression.		General		210
READ	READ variable name [,variable name. . .]	Reads data defined using the DATA command into a variable.		General		211
REM	REM (comment statement)	Inserts a comment statement in a program.		General		211
RENUM	RENUM [new line number] [,old line number] [,increment]]	Reallocates line numbers in the program.	Command			212
RESTORE	RESTORE [line number] or RESTORE [label]	Specifies the line number of the DATA command that is starting to be read using the READ command.		General		212
RESUME	RESUME [line number] or RESUME [NEXT]	Resumes program execution after an error handling procedure has been performed.		General		213
RIGHT$	RIGHT$ (character expression, number of characters)	Finds the specified number of characters from the rightmost character of the character string.			Character string	213
RND	RND [(numeric expression)]	Determines a random number between 0 and 1.			Arithmetic operation	214
ROMLOAD	ROMLOAD	Reads the BASIC program from flash ROM and stores it in user memory (RAM).	Command			214
ROMSAVE	ROMSAVE	Stores BASIC program from user memory (RAM) in flash ROM.	Command			215
ROMVERIFY	ROMVERIFY	Compares BASIC programs in RAM and flash ROM.	Command			215
RTS	RTS #port number SET or RTS #port number RESET	Sets/resets the RS signal.		Device control		216
RUN	RUN [first line number] [TO final line number]	Starts executing a program, and continues executing until a specified final line number.	Command			216

Command	Syntax	Details	Command	Statement	Function	Page
SAVE	SAVE #port number [communications conditions character expression]	Reads the BASIC program in user memory area to terminal.	Command			217
SEARCH	SEARCH (integer type array variable, numeric expression 1, [numeric expression 2] [,numeric expression 3]	Searches an integer array for the specified value. Returns the array subscript of the first instance found.			Special	217
SGN	SGN (numeric expression)	Determines the sign of an argument.			Arithmetic operation	218
SIN	SIN (numeric expression)	Determines the sine (SIN) given by the argument.			Arithmetic operation	218
SPACE$	SPACE$ (number of characters)	Creates a string of spaces of the length of the number of characters.			Character string	218
SPC	SPC (numeric expression)	To output a number of spaces in a PRINT or LPRINT statement.			Character string	219
SQR	SQR (numeric expression)	Calculates the square root.			Arithmetic operation	219
STEP	STEP [numeric expression]	Executes the specified number of lines in BASIC program.	Command			220
STOP	STOP	Terminates program execution.		General		220
STR$	STR$ (numeric expression)	Converts the numeric value into a character string.			Character string	221
STRING$	STRING$ (numeric expression, numeric expression) or STRING$ (numeric expression, character expression)	Finds a character string of the specified number of characters.			Character string	221
TAB	TAB (digit position)	Outputs blank space until the specified digit position in the PRINT and LPRINT commands.			Character string	222
TAN	TAN (numeric expression)	Determines the tangent (TAN) of the value given by the argument.			Arithmetic operation	222
TIME$	TIMES$ [“HH:MM:SS”]	Sets or gives the time.			Special	222
TIME$ ON/OFF/ STOP	TIME$ ON or TIME# OFF or TIME$ STOP	Enables, disables. or stops a time interrupt.		General		223
TIMER ON/ OFF/STOP	TIMER ON or TIMER OFF or TIMER STOP	Enables, disables, or stops a interval timer interrupt.		General		223
TRACE	TRACE	Displays the contents of the trace buffer.	Command			224
TROFF	TROFF	Cancels the tracing of the following line numbers.		General		224
TRON	TRON [M]	Starts tracing the following line numbers. Outputs to the trace buffer when the M option is used.		General		225

Command	Syntax	Details	Command	Statement	Function	Page
VAL	VAL (character expression)	Converts a character string into a numeric value.			Character string	225
VARPTR	VARPTR (variable name)	Determines or returns the memory address where the variable is stored.			Special	226
WAIT	WAIT “waiting time”[,line number]	Sets the monitoring time until execution of following instruction is completed.		General		227
WATCH	WATCH [SET variable name] or WATCH [DEL variable name]	Sets (SET), deletes (DEL), or displays the list (when not displayed) of the variables for the WATCH command.	Command			228
WHILE/WEND	WHILE <numeric expression> <program> WEND	Repeats the program body as long as the expression is not 0.		General		229

7-6 Details of BASIC Commands

This section explicitly defines the functions, commands and statements of the BASIC language and gives examples of their use. The conventions used to describe the syntax of the BASIC Language are:

1. <> Brackets in the syntax denote a term that is expanded elsewhere in the table. E.g. *<expression>* means that the definition of an expression can be found elsewhere in the table.
2. [] Brackets surround optional items. E.g., ***pinf*** *[<Program number>]* means that the Program number is optional in the pinf command.
3. BASIC keywords and pre-defined functions are written in boldface, e.g., in the example above pinf is a BASIC keyword.
4. A choice between several selections is denoted by ( ) parentheses. If parentheses are to be entered as they are, they will be denoted by **( )**. The items to be selected are separated by a vertical bar, | .
5. When the differences between two selections is too great they will be shown in a separate RHS entry.
6. Repetition is denoted by {} brackets. Items enclosed in these brackets are repeated zero or more times. e.g. ***data*** *<data field> {, <data field>}*
7. Items enclosed in quotes, “”, should be entered as they are written.

@if	
Syntax:	@IF <numerical expression> THEN [(<program>] [ELSE [(<program>]] ENDIF
Description:	Statement. Controls the flow of a program based on the results of a numerical expression. This statement differs from the IF statement in that it supports multiple line statements. @IF statements can be nested.
Remarks:	The results of <numerical expression> determine which branch of the conditional statement is executed. If the result of the <numerical expression> is non-zero (TRUE) then the statements after THEN are executed, otherwise the statements after ELSE are executed. It is the user’s responsibility to ensure that an ENDIF is encountered. If there is no ELSE statement then execution will continue with the next statement. All @IF statements must be terminated by an ENDIF. If an ENDIF is encountered before a corresponding @IF or if the program ends without encountering an ENDIF (if an @IF has already been encountered), the “mismatch: @IF / ENDIF” error (code B029) will occur.

Examples:	`> 10 A= -5`
	`> 20 @IF A<0 THEN`
	`> 30  B = -A`
	`> 40  PRINT “ABS OF “; A; “ IS “; B`
	`> 50 ELSE`
	`> 60  PRINT “ABS OF “; A; “ IS”; A`
	`> 70  PRINT “END”`
	`> 80 ENDIF`
	`> 90 END`
	`> RUN`
	`ABS OF -5 IS 5`

See also:	IF

abs	
Syntax:	ABS(<numerical expression>)
Description:	Function. Calculates the absolute value of a numerical expression.
Remarks:	<numerical expression> may be any short or long integer, single-precision floating point or double-precision floating-point expression. The range of the input argument, and therefore the output is equivalent to the allowable range of a double precision floating point. The return type is the same as that of the argument.

Examples:	`> 10 A = -4`
	`> 20 PRINT ABS(A)`
	`> RUN`
	`4`

See also:	

acos	
Syntax:	ACOS(<numerical expression>)
Description:	Function. Calculates the arc cosine of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating-point expression. The valid range for the input is [-1, 1]. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point. The return value is in radians and will be in the range [0, π]

Examples:	`> 10 A = 0.4`
	`> 20 PRINT ACOS(A)`
	`> RUN`
	`1.15928`

See also:	ASIN, ATN

alarm on/off/stop

Syntax:	ALARM (ON | OFF | STOP)
Description:	Statement. Enables, disables or stops the time interrupt
Remarks:	ON enables the interrupt. When this statement is executed the ON ALARM interrupt is unmasked. If an ALARM interrupt occurs, program execution branches to the defined GOSUB routine for interrupt processing. OFF disables the interrupt. All subsequent ALARM interrupts are ignored. STOP masks and enables the interrupt. If an interrupt is received, it is stored in memory but program execution is not branched to the interrupt subroutine. Program execution will be branched to the stored interrupt's subroutine when the interrupt is unmasked with the ALARM ON statement. **Note:** 1. Only one ON ALARM interrupt may be valid. If more than one is set in a program then the last one executed is valid. 2. After branching processing the ALARM interrupt, the interrupt is disabled and all subsequent ALARM interrupts are ignored until it is re-enabled with the ALARM ON statement. 3. The ALARM ON/OFF/STOP statements can be executed only after the ON ALARM statement has been executed. 4. Interrupts are disabled immediately after execution of ON ALARM GOSUB.

Examples:	`> 10 ON ALARM 30 GOSUB 1000`
	`> 20 ALARM ON`
	`> 30 GOTO 30`
	`> 1000 PRINT "ALARM INTERRUPT OCCURRED"`
	`> 1010 RETURN`
	`> RUN`
	`ALARM INTERRUPT OCCURRED`

See also:	ON ALARM

asc

Syntax:	ASC(<string expression>)
Description:	Function. Returns the character code of the first character in the <string expression>.
Remarks:	The return value is an integer representing the ASCII character code of the first character in the string expression. **Note:** 1. The return value will be in the range: *[0…255]*

Examples:	`> 10 B$ = "B"`
	`> 20 PRINT ASC(B$)`
	`> RUN`
	`66`

See also:	CHR$

asin

Syntax:	ASIN(<numerical expression>)
Description:	Function. Calculates the arc sine of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating-point expression. The valid range of the input argument is [-1, 1]. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point. The return value is in radians and will be in the range [-π/2, π/2]

Examples:	`> 10 A = 0.4`
	`> 20 PRINT ASIN(A)`
	`> RUN`
	`.411517`

See also:	ACOS, ATN

atn

Syntax:	ATN(<numerical expression>)
Description:	Function. Calculates the arc tangent of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating-point expression. The return value is in radians and will be in the range [-π/2, π/2] The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point.

Examples:	`> 10 A = 0.4`
	`> 20 PRINT ATAN(A)`
	`> RUN`
	`.380506`

See also:	ACOS, ASIN

auto

Syntax:	AUTO [<line number>] [, [<increment>]]
Description:	Command. Automatically generates line numbers for program lines as they are being input.
Remarks:	<line number> is an integer in the range: *[1…65535]*. <increment> is an integer in the range: *[1…65535]*. auto begins line numbering at the line specified by <line number> and increments subsequent lines by <increment>. Both <line number> and <increment> default to 10. **Note:** 1. The AUTO statement can be cancelled with CTRL-X. 2. An asterisk (*) immediately after the specified line number denotes that an already existing line number could be overwritten. 3. A new line number can be entered by typing the new line number followed by the CR key. If the CR key is pressed without entering a new program line next line number in the sequence will be displayed.
Examples:	`> AUTO 10,10`
See also:	EDIT

bitoff/biton

Syntax:	BITOFF <integer variable> , <bit position> BITON <integer variable> , <bit position>
Description:	Statement. turns ON (1) or OFF (0) the specified bit of an integer variable.
Remarks:	<bit position> is an integer expression returning a value in the range: *[0…31]*. **Note:** 1. A <bit position> of zero specifies the least significant bit. 2. A value in the range *[0…15]* can be used for an integer <bit position> and a value in the range *[0…31]* can be used for a long integer <bit position>.

Examples:	`> 10 N% = 182`
	`> 20 PRINT N%`
	`> 30 BITON N%,3`
	`> 40 PRINT N%`
	`> 50 BITOFF N%,5`
	`> 60 PRINT N%`
	`> RUN`
	`182`
	`190`
	`158`

See also:	

brkpt

Syntax:	BRKPT [(SET | DEL)] (<line number> | ALL)
Description:	Command. Sets, deletes or lists breakpoints in the BASIC debugger.
Remarks:	<line number> is any valid line in the BASIC program. Valid range: *[1…65535]*. SET sets a breakpoint at the specified <line number>. DEL deletes the breakpoint from the specified <line number>. If the BRKPT command is used without SET or DELETE, it lists all the currently set breakpoints. If a line number is specified that does not exist, then an "Undefined Line Number" error (code B008) will occur. A maximum of 255 breakpoints can be set at any given time. On reaching a program line that has a breakpoint set the execution will stop before executing that line. If `BRKPT SET ALL` is executed, a "NO SUPPORT" error will occur. The setting of the maximum of 255 breakpoints may not be possible if there is insufficient free space in the memory area. A breakpoint can be set while executing an interrupt program to stop the program, but the program cannot be restarted from the location of the breakpoint.
Examples:	`> BRKPT SET 200`
See also:	CONT, STEP, STOP, END

cdbl

Syntax:	CDBL (<numerical expression>)
Description:	Function. Converts the <numerical expression> into a double-precision floating point.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating-point expression. If the <numerical expression> is an integer or single-precision floating-point integer, then CDBL will convert it to a double-precision floating-point integer. If the <numerical expression> is already a double-precision floating-point integer, then the precision of the returned value will be the same as it was before conversion. Even after conversion to a double-precision floating-point integer, only the digits for single precision may sometimes be shown. The number, however, will have the number of digits required for double precision. If the number contains consecutive zeros, only the digits for single precision may sometimes be output, as described above.

Examples:	`> 10 A# = CDBL(22)/CDBL(7)`
	`> 20 B# = 22!/7!`
	`> 30 PRINT A#, B#`
	`> RUN`
	`3.14285714285714  3.14286`

See also:	CINT, CSNG

chr$

Syntax:	CHR$ (<numerical expression>)
Description:	Function. Converts the <numerical expression> into a character.
Remarks:	<numerical expression> may be any integer in the range: *[0…255]*. Non-integer numerical expressions are rounded to the nearest integer. The decimal equivalent of the ASCII character code is used and converted to the corresponding character. **Note:** 1. If the integer expression lies out with the specified range an “Illegal Function call” (code B005) or “Overflow” (code B006) error will result. 2. The ASC function is the opposite of CHR$. 3. As an example of the rounding function, CHR$(67.4) returns ‘C’ and CHR$(67.5) returns ‘D’.

Examples:	`> 10 A$ = CHR$(67)`
	`> 20 PRINT A$`
	`> RUN`
	`C`

See also:	ASC

cint

Syntax:	CINT (<numerical expression>)
Description:	Function. Converts the <numerical expression> into an integer.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. the valid range of the input is the valid range of a short integer: [-32768... 32767]
	Note: 1. Fractions are rounded up if positive and down if negative. For instance, CINT(67.5) returns 68 but CINT(-67.5) returns -68 not -67.

Examples:	`> 10 A% = CINT(67.45)`
	`> 20 B% = CINT(67.55)`
	`> 30 PRINT A%, B%`
	`> RUN`
	`67  68`

See also:	CDBL, CLNG, CSNG, FIX, INT

clear

Syntax:	CLEAR
Description:	Statement. Initializes all numeric and character variables.
Remarks:	All numerical variables are initialized to 0. All character strings are initialized to be empty.
	Note: Refer to *7-2 Character Variable Space Allocations* for details on allocating space in the character variable area. Dynamic Method: 1. All values = 0. Delete all temporary memory blocks. 2. Close all open ports. 3. For character variables, all temporary-holding areas will be reset. Static Method: 1. Close all ports that are open and set all values to 0. 2. Fixed memory areas in user memory will remain held, but the values within these areas will be cleared.
Examples:	`CLEAR`
See also:	

clng

Syntax:	CLNG (<numerical expression>)
Description:	Function. Converts the <numerical expression> into a long integer.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. the valid range of the input is the valid range of a long integer: [-2147483648... 2147483647]
	Note: 1. Fractions are rounded up if positive and down if negative. For instance, CLNG(67.5) returns 68 but CLNG(-67.5) returns -68 not -67.

Examples:	`10 A&= CLNG(40000.45)`
	`20 B&= CLNG(40000.55)`
	`30 PRINT A&, B&`
	`> RUN`
	`40000  40001`

See also:	CDBL, CINT, CSNG, FIX, INT

close

Syntax:	CLOSE [# <port expression>]
Description:	Statement. Closes a communications port..
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. It is also possible in this command to omit the # preceding the port expression. On closing a port, the data remaining in the output buffer of the communications port is sent, the data in the input buffer of the communications port is cleared. If the port expression is omitted, all open ports will be closed. Once the port has been closed it cannot be used for data transfer until it has been opened again. The END and NEW commands automatically close the communications ports. The STOP command does not. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result. 3. For the status of the RTS and DTR signal lines on closing a port, refer to *4.5.2 Transmission Control Signal Timing Chart*.
Examples:	`CLOSE #2`
See also:	END, NEW, OPEN

cls

Syntax:	CLS [# <port expression>]
Description:	Statement. Clears the screen.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. If the port number is omitted, the number of the terminal port will be used (#1 for the C200H-ASC11/21 and #3 for the C200H-ASC31). **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`CLS`
See also:	

com on/off/stop

Syntax:	COM [<port expression>] (ON | OFF | STOP)
Description:	Statement. Enables, disables, or stops the interrupt defined by the ON COM interrupt statement.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 2]*. If the port number is omitted, the port number will be #1. ON enables the interrupt. If a COM interrupt occurs while the interrupt is enabled, program execution branches to the defined subroutine for interrupt processing. OFF disables the interrupt. All subsequent ON COM interrupts are ignored. STOP masks and enables the interrupt. If an interrupt is received, it is stored in memory but program execution is not branched to the interrupt subroutine. Program execution will be branched to the stored interrupt's subroutine when the interrupt is unmasked with the COM ON statement. **Note:** 1. The COM ON/OFF/STOP statement can only be executed after an ON COM statement has been executed. 2. Interrupts are disabled immediately after execution of ON COM GOSUB.
Examples:	`COM 1 STOP`
See also:	ON COM

cont

Syntax:	CONT
Description:	Command. Resumes execution of the BASIC program after CTRL+X, after execution of END, STEP or STOP statements, after encountering a breakpoint or after the occurrence of an error.
Remarks:	Execution of the BASIC program resumes at the point after the break occurred. If the program has been changed or if the break occurs during data transfer with an external device then the BASIC program cannot be resumed. If there is a break during the execution of an interrupt program, the program cannot be resumed using the CONT command. When the BASIC program cannot be resumed, a "CANNOT CONTINUE" error message (code B017) will result.
Examples:	CONT
See also:	STEP, STOP, END, BRKPT

cos

Syntax:	COS (<numerical expression>)
Description:	Function. Calculates the cosine of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point. The return value will be in the range [-1, 1]. **Note:** 1. The argument should be specified in radians.

Examples:	`> 10 A = 0.4`
	`> 20 PRINT COS(A)`
	`> RUN`
	`.921061`

See also:	SIN, TAN

csng

Syntax:	CSNG(<numerical expression>)
Description:	Function. Converts a numerical expression into a single-precision floating point.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating-point expression. If the <numerical expression> is an integer or double-precision floating-point type then CSNG will convert it to a single-precision floating-point type. If the <numerical expression> is already a single-precision floating point then the precision of the returned value will be the same as it was before conversion.

Examples:	`> 10 A# = 1.23456789`
	`> 20 A! = A#`
	`> 30 B! = CSNG(A#)`
	`> 40 PRINT A#, A!, B!`
	`> RUN`
	`1.23456789  1.23457  1.23457`

See also:	CDBL, CINT

cts

Syntax:	CTS(<port expression>)
Description:	Function. Reads the CTS line of the specified communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. CTS returns -1 if the CTS line of the specified port is active and 0 if the CTS line is not active. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or if port #2 or port #3 is specified on ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`> 10 IF CTS(2) THEN PRINT("PORT 2 IS READY")`
See also:	DSR. DTR, RTS

data

Syntax:	DATA [<data field> {, <data field>}]
Description:	Statement. Defines numeric and character constants that can be used in subsequent read statements
Remarks:	<data field> is any valid numerical or character constant. In order to include the ',', ':' and '(space)' characters in data fields they must be enclosed in quotes. The DATA statement is usually used in conjunction with the READ statement. The constants' type and order are specified by the READ statement and the variables' type and order must match. The data defined in these statements will be read in order by READ statements. When the last data item in a DATA statement is read then the next data item read will be the first data item in the following DATA statement, if it exists. The variable type in the READ statement must correspond to the type of data read from the DATA statement. **Note:** Comments cannot be attached to DATA statements.

Examples:	`> 10 DATA 10, "HELLO", 1.6, "WORLD"`
	`> 20 READ A, B$, C, D$`
	`> 30 PRINT B$+" "+D$`
	`> 40 PRINT A+C`
	`> RUN`
	`HELLO WORLD`
	`11.6`

See also:	READ, RESTORE

date$

Syntax:	DATE$ [= <date string expression>]
Description:	System variable. Returns the date of the internal clock or sets the internal clock's date.
Remarks:	<date string expression> is an absolute date in the format "yy/mm/dd". Where yy is the year and has a valid range *[00…99]*, mm is the month and has a valid range *[01…12]* and dd is the day and has a valid range *[01…31]*. If the <date string expression> is not a valid date then an "ILLEGAL FUNCTION CALL" error (code B005) will occur. If used on its own the DATE$ statement returns the current date of the system clock in the format specified above.
Examples:	`DATE$ = "97/06/26"`
See also:	DAY, TIME$

day

Syntax: DAY [= <numerical expression>]

Description: System variable. Returns the current day of the week or sets the day of the week.

Remarks: <numerical expression> is an integer expression representing the day of the week. The days of the week are represented in the following way:

Day code	Day
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

If the <numerical expression> is not within the range specified in the table above then an "ILLEGAL FUNCTION CALL" error (code B005) will occur.

Once the DAY variable has been set, it is automatically incremented by 1 with each passing day (repeating the cycle 0 through 6).

The DAY variable has no relationship with the internal clock, it is quite possible to set a DAY that is inconsistent with the current DATE$

Examples:

```
DAY = 4
```

See also: DATE$, TIME$

def fn

Syntax: DEF FN<function name>[(<variable name 1> [, <variable name n>])] = <expression>

Description: Statement. Declares a user defined BASIC function.

Remarks: <function name> is the name of the defined function. It may be any valid BASIC variable name. Names starting with reserved words cannot be used. The following characters cannot be used at the end of a name: $ % & ! #

<Variable name 1> [, <variable name n>] is an optional list of arguments that may be input to the user defined function. Any valid variable name may be used as an input argument. It is not possible to have arguments with the same name as the function being defined (i.e., <variable name 1>).

The <expression> is the constant or string expression that uses <Variable name 1> [, <variable name n>].

Note:

1. The type of variable used when the function is called as a BASIC function must match the variable used when the function was defined with the DEF FN statement. A "TYPE MISMATCH" error (code B013) will occur if the variable types do not match.
2. The defined basic function is called by entering FN + the function name. For example, A = FNMYFUNC(1,2) will call the MYFUNC function defined in the example below.
3. The result of the user-defined function must be assigned a variable when the function is called. A "SYNTAX ERROR IN LINE …" error (code B002) will occur if a variable is not assigned.

Examples:

```
> 10 DEF FNMYFUNC(A,B) = A+B
> 20 PRINT FNMYFUNC(1,2)
> RUN
 3
```

See also:

def int/lng/sng/dbl/str

Syntax:	DEF (INT | LNG | SNG | DBL | STR) <letter> [- <letter>] {, <letter> [- <letter>]}
Description:	Statement. Defines the type of variables beginning with the specified letters..
Remarks:	The variables can be declared to be of type short integer (INT), long integer (LNG), single-precision floating point (SNG), double-precision floating point (DBL) or string (STR). Variables beginning with the letters specified, or in the range of letters specified will be assigned to be of the declared type. If the type is declared using the type definition characters (%, &, !, # or $) then this will take precedence over the types declared using DEF INT/LNG/SNG/DBL/STR. If no type declarators are used all numeric variables are assumed to be single-precision floating point. When the OPTION LENGTH statement is used in a program, a variable name defined by a DEF statement cannot be used before the DEF statement.

Examples:	`> 10 DEFINT A`
	`> 20 A = 1.2`
	`> 30 PRINT A`
	`> RUN`
	` 1`

See also:	

del

Syntax:	DEL ((<line number> | .) [- [(<line number> | .)]]
Description:	Command. Deletes the specified program lines.
Remarks:	<line number> is any valid line in the BASIC program. Valid range: *[1…65535]*. **Note:** 1. A period (.) can be used instead of the current line number. 2. The first number in any given range must be less than the second number of that range. If this is not true then an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`DEL 100-200` deletes a range of lines from 100 to 200 inclusive. `DEL 100-` deletes a range of lines from 100 to the end of the program. `DEL -600` deletes a range of lines from the beginning of the program to line 600
See also:	

dim

Syntax:

DIM <variable> (<array index> {, <array index>}) {,<variable> (<array index> {, <array index>})}

Description:

Statement. Specifies the order of the array and the maximum value for the array variable subscripts. Allocates the necessary memory for the array.

Remarks:

The <variable> is any valid variable name. When array variables are declared, the constant array variables are initialized to 0 and character string array variables are initialized to NULL. An array may have up to 255 subscripts (dimensions), but an array cannot be declared that is larger than the available memory in the ASCII Unit.

The <array index> is any valid integer in the range: *[0…65535]*. A constant must be used when the static method is used for character variable area allocation.

The default value for the <array index> is 10 when an array variable is used without the DIM statement (without declaring the array).

Note: The DIM command must be executed before any substitute expression.

Note:

1. An array variable allows multiple elements of data to be specified with a single variable name. The data element is specified by the number in parentheses after the variable name (constant or string type). This element number is called the subscript. An array with one subscript is one-dimensional, an array with two subscripts is two dimensional, and so on.
2. The statement DIM A$(3) declares a one-dimensional array of string variables with variable name A$. The following four variables can be used: A$(0), A$(1), A$(2), and A$(3).
3. The statement DIM B(3,2) declares a two-dimensional array of constant variables with variable name B. The following twelve variables can be used: B(0,0), B(1,0), B(2,0), B(3,0), B(0,1), B(1,1), B(2,1), B(3,1), B(0,2), B(1,2), B(2,2), and B(3,2).
4. Array subscripts begin from 0 as shown in the examples above. If arrays are to be started with subscript 1, use the OPTION BASE statement.

Examples:

```
>  10 DIM A(5)
>  20 FOR I = 0 TO 5
>  30 A(I) = I
>  40 PRINT A(I)
>  50 NEXT I
>  RUN
0
1
2
3
4
5
```

See also:

OPTION BASE

dma

Syntax:	DMA(<port expression>)
Description:	Function. Reads the DMA status of the specified communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1…2]*. DMA returns -1 if DMA is active for the specified port and 0 if DMA is not active. **Note:** 1. If the port expression is outside the specified range or if it refers port #1 on ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`IF DMA(2) THEN PRINT "DMA OF PORT 2 IS ACTIVE"`
See also:	

dsr

Syntax:	DSR(<port expression>)
Description:	Function. Reads the DSR line of the specified communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. DSR returns -1 if the DSR line of the specified port is active and 0 if the DSR line is not active. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or if port #2 or port #3 is specified on ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`IF DSR(2) THEN PRINT "DSR OF PORT 2 IS ACTIVE"`
See also:	CTS, DTR, RTS

dtr

Syntax:	DTR # <port expression> (SET | RESET)
Description:	Statement. Sets or resets the DTR line of the specified communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. If accompanied with the SET command, the ER line is made active. If the RESET command is used, then the ER becomes inactive. This command can only be used if the port is open. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or if port #2 or port #3 is specified on ASC21 a "BAD PORT NUMBER" error (code 0050) will result.
Examples:	DTR #2 SET
See also:	CTS, DSR, RTS

edit

Syntax: EDIT (<line number> | .)

Description: Command. Edits the specified line(s) of the BASIC program.

Remarks: Allows editing of the BASIC program. If only one line number is specified then the editor will be in single-line edit mode. Characters can be inserted, deleted, or copied in the specified line and the cursor can be moved.
When "EDIT." Is input, the most recently displayed line or most recently input line will be displayed.

It is also possible to edit programs that have been input with the PNAME command.

The following edit keys can be used during edit.

Key	Action
Right Key (→)	Moves the cursor to the right. If the cursor is already at the last position in the line then this key has no effect.
Left Key (←)	Moves the cursor to the left. If the cursor is already at the first position in the line then this key has no effect.
Down Key (↓)	Inputs the line currently being edited and displays the next program line. If the Down Key is pressed when the bottom line is already displayed, editing is terminated, and the editor returns to command mode.
Up Key (↑)	Inputs the line currently being edited and displays the previous program line. If the Up Key is pressed when the top line is already displayed, editing is terminated, and the editor returns to command mode.
CTRL+ O (Insert or Overwrite)	Toggles the editor between insert and overwrite mode, the status of insert mode is not shown.
DEL or BS	Deletes the character to the left of the cursor, moving subsequent characters in the buffer to the left. This key has no effect at the beginning of a line.
RETURN	The carriage return key enters the line currently being edited to the BASIC program. When the line number being displayed is changed and the return key is pressed, a new line will be created with the new line number.
CTRL+X	Quits editing and returns to command mode. If a line is currently being edited, it will return to its original contents.

Note:
1. When just the line numbers are edited, the contents of the line before editing are copied as the contents of the line after editing.
2. If the specified line number does not exist then an “UNDEFINED LINE NUMBER” error (code B008) will occur.

Examples: `EDIT 200` This edits line 200 of the active program

See also:

end

Syntax:	END
Description:	Statement. Terminates program execution and returns to the command mode. All currently open ports are closed.
Remarks:	END can be used anywhere in a BASIC program. Like the CLOSE statement, the data remaining in the output buffer of the communications port is sent and the data in the input buffer of the communications port is cleared. The END statement can be omitted from the end of the program, but the ports will not be closed in this case.

Examples:	`> 10 INPUT A`
	`> 20 @IF A = 1 THEN GOSUB 1000`
	`> 30 ELSE`
	`> 40 PRINT "A WAS NOT 1"`
	`> 50 ENDIF`
	`> 60 END`
	`> 1000 PRINT "A WAS 1"`
	`> 1010 RETURN`

See also:	STOP

eof

Syntax:	EOF(<port expression>)
Description:	Function. Checks if the input buffer of the specified port is empty..
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. EOF returns -1 if the input buffer of the specified port is empty and 0 not. **Note:** 1. The port specified must be open and configured for input. 2. Port #3 corresponds to the terminal port of the ASC31. 3. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`IF EOF(2) THEN PRINT "PORT 2 IS EMPTY"`
See also:	

erc

Syntax:	ERC
Description:	Statement. Clears errors from the error history table in the ASCII Unit.
Remarks:	When ERC is executed as a command, all of the errors are cleared from the error history table. When ERC is executed as a statement in the program, the current error status is cleared and those current errors are registered in the error history table as old errors. ERC cannot be used to clear errors 90 to 99 (DM setting errors). The error code table is a table that records the error codes of up to 30 errors that have occurred in the ASCII Unit. The error codes indicate the type of error that has occurred. Refer to *9-1 List of Error Messages* for details on error codes.
Examples:	ERC
See also:	ERL, ERR

erl

Syntax:	ERL
Description:	System Variable. Stores the line number executing at the time an error occurred.
Remarks:	ERL is a read-only system variable; it cannot be assigned a value. ERL can be helpful in error processing when used with ERR and the ON ERROR GOTO statement. The contents of ERR and ERL can be checked from an error-processing interrupt subroutine and the information (error code and line number) can be used for error processing. **Note:** 1. ERL will have a value 0 if statements causing an error are executed at the command prompt. 2. When an error occurs, the flow of the program can be controlled by using the ERL function in an interrupt subroutine.

Examples:	`> 10 ON ERROR GOTO 1000`
	`> 20 A=A/0`
	`> 30 END`
	`> 1000 PRINT "ERROR #";ERR;" OCCURRED AT LINE ";ERL`
	`> 1010 END`
	`> RUN`
	`ERROR #11  OCCURRED AT LINE  20`

See also:	ERC, ERR

err

Syntax:	ERR
Description:	System Variable. Stores the error code after an error has occurred.
Remarks:	ERR is a read-only system variable; it cannot be assigned a value. ERR can be helpful in error processing when used with ERL and the ON ERROR GOTO statement. The contents of ERR and ERL can be checked from an error-processing interrupt subroutine and the information (error code and line number) can be used for error processing. **Note:** 1. ERR will have a default value of 0 if no error has occurred. 2. When an error occurs, the flow of the program can be controlled by using the ERR function in an interrupt subroutine.

Examples:	`> 10 ON ERROR GOTO 1000`
	`> 20 A=A/0`
	`> 30 END`
	`> 1000 PRINT "ERROR #";ERR;" OCCURRED AT LINE ";ERL`
	`> 1010 END`
	`> RUN`
	`ERROR #11 OCCURRED AT LINE 20`

See also:	ERC, ERL

error

Syntax:	ERROR <error number>
Description:	Statement. Simulates the occurrence of an error.
Remarks:	<error number> is any valid error number in the range: *[1…255]*. If a system error or a BASIC program error number is used the relevant error code will be put in the IR n+7 word and, if applicable, the error message will be printed to the screen. If an error number is specified that does not contain an error description, for instance an undefined user-defined error or a system error code that is not used, then an "UNDEFINED ERROR" error code will occur. For both system and user errors it is possible to define error handling using the ON ERROR GOTO statement. ERROR and ON ERROR GOTO can be used together to simulate the occurrence of an error and test the operation of an error-processing program in an interrupt subroutine.

Examples:	`> 10 ON ERROR GOTO 1000`
	`> 20 ERROR 6`
	`> 30 END`
	`> 1000 PRINT "ERROR #";ERR;" OCCURRED AT LINE ";ERL`
	`> 1010 END`
	`> RUN`
	`ERROR #6 OCCURRED AT LINE 20`

See also:	ON ERROR GOTO, ERL, ERR, ERC

exp

Syntax:	EXP(<numerical expression>)
Description:	Function. Calculates the exponential of a numerical expression with base *e*.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point.

Examples:	`> 10 A = EXP(1)`
	`> 20 PRINT A`
	`> RUN`
	`2.71828`

See also:	

fcs

Syntax: FCS(<string expression>[, <fcs type>])

Description: Function. Calculates the frame checksum of a string expression.

Remarks: The FCS is calculated by performing one of the following operations on all bytes in the string expression.

The type of FCS calculation is selected by <fcs type>. This is an integer value in the range: *[0…9]*. The following calculations can be performed. The type of the return depends on whether a BINARY or ASCII calculation is requested. BINARY calculations return integers and ASCII calculations return character strings.

<fcs type>	Calculation method	FCS code data	
		Data type	Number of bytes
0	LRC	Binary	1
1		ASCII	2
2	CRC-CCITT	Binary	2
3		ASCII	4
4	SUM (1 byte)	Binary	1
5		ASCII	2
6	SUM (2 bytes)	Binary	2
7		ASCII	4
8	CRC-16	Binary	2
9		ASCII	4

Note:

1. The calculation logic for CRC-CCITT is XMODEM.
2. The calculation logic for CRC-16 is MODBUS. For example, character string data is handled as follows:

   "1 2 3 4" = 31 32 33 34

   $x^n + x^{n-1} \ldots x^1 = x^0$

Examples:

```
>  10 B$ = "RD89"
>  20 PRINT FCS(B$, 1)
>  RUN
17
```

See also:

fix

Syntax:	FIX(<numerical expression>)
Description:	Function. Returns the integer part of the numerical expression specified as the argument.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is the type of argument. **Note:** 1. Returns the integer part without rounding down or up after the decimal point. 2. If the argument value is positive, FIX will return the same value as INT. If the argument value is negative, FIX will return a value of 1 more than INT. For example, for FIX(-6.1), -6 is returned. 3. Refer to *Precautions on Real Numbers* in *7-4-4 Constants.*

Examples:	`> 10 A = FIX(6.1)`
	`> 20 B = FIX(6.9)`
	`> 30 C = FIX(-6.1)`
	`> 40 D = FIX(-6.9)`
	`> 50 PRINT A,B,C,D`
	`> RUN`
	`6  6  -6  -6`

See also:	INT

for to step / next

Syntax:	FOR <variable> = <numeric expression 1> TO <numeric expression 2> [STEP <numeric expression 3>] (: | <end-of-line>) <program> NEXT <variable> {,<variable>}
Description:	Statement. Allows the program segment between the FOR and the NEXT statement to be repeated a number of times.
Remarks:	The FOR statement specifies the beginning of the loop and the NEXT statement specifies the end of the loop. The loop is executed a specified number of times depending on the value of the expressions in the FOR statement. <Variable> is used as a counter and is incremented by the amount specified by <numeric expression 3> after the STEP statement each time the loop is executed. The initial value of the variable is equal to <numeric expression 1> in the FOR statement. The execution of the loop terminates when the value of <variable> surpasses that of <numeric expression 2> defined after the TO statement. If the STEP increment is zero the loop is executed indefinitely. If the initial value of the variable is greater than the TO value then the increment specified must be negative. If not, then the loop is never executed. The NEXT statement can be used to terminate more than one FOR loop by specifying more than one loop counter variable. For instance if two nested loops terminate at the same point then one NEXT statement specifying both variables can be used. Care must be taken to terminate nested loops in the correct order. Example : 10 FOR A = 1 TO 10 20 FOR B = 1 TO 10 30 PRINT A, B 40 NEXT B, A The variable(s) used in the NEXT statement must correspond to that used in the FOR statement(s)

Examples:	`> 10 FOR A=1 TO 3`
	`> 20 PRINT A`
	`> 30 NEXT A`
	`> RUN`
	`1`
	`2`
	`3`

See also:	WHILE/WEND

fre

Syntax:	FRE
Description:	System Variable. Reports amount of free user memory .
Remarks:	FRE is a read-only system variable; it cannot be assigned a value. The result is the amount of available user memory, in bytes. **Note:** 1. The result is the total user memory available, there is no differentiation between program and data memory areas.

Examples:	`> 10 PRINT "THE FREE USER MEMORY IS "; FRE; " BYTES"`
	`> RUN`
	`THE FREE USER MEMORY IS 204656 BYTES.`

See also:	

gosub

Syntax:	GOSUB (<line number> | <label>) … RETURN
Description:	Statement Branches unconditionally to and returns from the subroutine specified by the line number or label.
Remarks:	<line number> is the first line number in the subroutine and may be any valid line number in the range: *[0 …65535]*. If the line number specified does not exist in the program then an "UNDEFINED LINE NUMBER" error will occur. <label> is a BASIC program label. It references a line number somewhere in the BASIC program. The RETURN statement will cause the subroutine to branch back and begin execution at the next valid line from the original calling line. If the specified line number or label is a non-executable statement, program execution will continue at the next line. Subroutines may be called from other subroutines (nested subroutines), the only limit is the free user memory. To prevent inadvertent entry to a subroutine it is advisable to precede the entry to the subroutine with END, STOP or GOTO statements.

Examples:	`> 70 INPUT C`
	`> 80 IF C=1 THEN GOSUB 1000`
	`> 90 IF C=2 THEN GOSUB 2000`
	`> 100 GOTO 70`
	`> 1000 CLS`
	`> 1010 PRINT "1 SELECTED"`
	`> 1020 RETURN`
	`> 2000 CLS`
	`> 2010 PRINT "2 SELECTED"`
	`> 2020 RETURN`
	`> RUN`
	`>?1<CR>`
	`1 SELECTED`
	`?`

See also:	END, GOTO, STOP

goto

Syntax:	GOTO (<line number> | <label>)
Description:	Statement Branches unconditionally to the specified line number or label.
Remarks:	<line number> is any valid line number in the range: *[1 …65535]*. If the line number specified does not exist in the program then an "UNDEFINED LINE NUMBER" error will occur. <label> is a BASIC program label. It references a line number somewhere in the BASIC program.

Examples:	`> 10 INPUT A`
	`> 20 PRINT HEX$(A)`
	`> 30 GOTO 10`
	`> RUN`
	`?11`
	`B`
	`?`

See also:	GOSUB

hex$

Syntax:	HEX$(<numerical expression>)
Description:	Function Converts the numerical expression into a hexadecimal character string.
Remarks:	The numerical expression must be in the range: *[-2147483648…2147483647]*. The resulting character string will be in the hexadecimal range: *[0…FFFFFFFF]*. **Note:** 1. Single-precision or double-precision floating-point expressions will be truncated before the conversion. 2. If a floating-point value is within the short integer range, the value will be converted to a hexadecimal integer. If the floating-point value is within the range of a long integer, the long integer will be converted to a hexadecimal. If the value is outside the long integer range, an "OVERFLOW" error (code B006) will occur.

Examples:	`> 10 PRINT "DEC", "HEX"`
	`> 20 FOR I = 1 TO 16`
	`> 30 PRINT I,HEX$(I)`
	`> 40 NEXT I`
	`> RUN`
	`DEC  HEX`
	`1  1`
	`2  2`
	`3  3`
	`4  4`
	`5  5`
	`6  6`
	`7  7`
	`8  8`
	`9  9`
	`10  A`
	`11  B`
	`12  C`
	`13  D`
	`14  E`
	`15  F`
	`16  10`
	`>`

See also:	OCT$

if	
Syntax:	IF <numerical expression> (THEN [(<statement> {: <statement>} | <line number> | <label>)] | GOTO (<line number> | <label>)) [ELSE [(<statement> {: <statement>} | <line number> | <label>)]]
Description:	Statement. Controls the flow of a program based on the results of a numerical expression. This statement differs from the @IF statement in that it supports statements only on a single line.
Remarks:	The results of <numerical expression> determine which branch of the conditional statement is executed. If the result of the <numerical expression> is not zero then the statements after THEN are executed, otherwise the statements after ELSE are executed. <label> is a BASIC program label. It references a line number somewhere in the BASIC program. If there is no ELSE statement then execution will continue with the next statement. If a valid line number is used instead of statements after the THEN or ELSE clauses then execution will continue at the specified line number. The GOTO statement can be used in conjunction with a valid line number for the same purpose. Nesting of IF statements is limited only by the length of the line. The IF statements must fit in one line.

Examples:	`> 10 A = 3`
	`> 20 IF A=2 THEN PRINT A IS 2" ELSE PRINT "A IS NOT 2"`
	`> RUN`
	`A IS NOT 2`

See also:	@IF

inkey$	
Syntax:	INKEY$ [#<port expression>]
Description:	Function Returns the next character in the specified port buffer.
Remarks:	If the specified port buffer is empty, a NULL string is returned. Any character, except CTRL-X, can be returned by INKEY$ **Note:** 1. If the port expression is not specified the terminal port is default.

Examples:	`> 10 IF INKEY$ <> "Q" GOTO 10`
	`> 20 PRINT "QUIT!"`
	`> 30 END`

See also:	

input

Syntax:	INPUT[;] [# <port expression>,][<prompt> (; | ,)]<variable> {, <variable>}
Description:	Statement. Accepts input to a specified variable or variables during program execution.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. If the <port expression> is omitted, the terminal port's port number is the default (port #1 for the ASC11/21 or port #3 for the ASC31). <prompt> is an optional text string that is output to the terminal to prompt for the input. If the prompt is followed by a semi-colon then a '?' will be appended, this can be suppressed by using a comma after the prompt string. If the port specified is not configured as a "TERM" or a "COMU" port then the prompt and the ? will be suppressed. (When the ? is not to be displayed for a "COMU" port, use the LINE INPUT statement.) Strings input to the INPUT statement need not be surrounded by quotes. The <variables> specified to receive the input can be any valid variable name. The number of values entered must be the same as the number of variables specified in the INPUT statement and the type of each value entered at the prompt must correspond to the types of the variables specified in the INPUT statement. Otherwise, a "NO SUPPORT" error (code 0064) will result. Input is terminated by a carriage return. Multiple values can be input by separating them by commas. **Note:** 1. If the port expression is not specified the terminal port is default. 2. Port #3 corresponds to the terminal port of the ASC31. 3. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result. 4. Execution of the BASIC program will be paused and the BASIC LED indicator on the front of the Unit will flash slowly until the <CR> code is input. 5. If an interrupt occurs while the INPUT statement is waiting for an input and the interrupt is not disabled or stopped, the interrupt subroutine will be processed immediately. Program execution will resume from the INPUT command when the interrupt subroutine has been completed. 6. If a semi-colon is added after the INPUT, the cursor will remain on the same row after input. 7. When the terminator section of the receive data is CR + LF, only CR will be accepted as the INPUT statement, so LF will remain in the receive buffer.

Examples:	`> 10 INPUT "ENTER YOUR NAME : ", A$`
	`> 20 PRINT "HELLO ";A$`
	`> 30 END`
	`> RUN`
	`ENTER YOUR NAME : FRED <CR>`
	`HELLO FRED`

See also:	LINE INPUT, WAIT

input$

Syntax:	INPUT$ (<numerical expression> [, #<port expression>])
Description:	Function Reads a string of characters from the specified port.
Remarks:	<numerical expression> is an integer expression indicating the number of input characters to be accepted. The valid range is : *[1…255]*. The system will wait until the specified number of characters has been input. The result is the specified number of characters. <port expression> is an expression returning an integer in the range: *[1… 3]*. All characters, excluding CTRL-X, can be read using INPUT$. (INPUT$ can read the CR and LF codes which cannot be read by the INPUT and LINE INPUT statements.) **Note:** 1. If the port expression is not specified the terminal port is default. 2. Port #3 corresponds to the terminal port of the ASC31. 3. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result. 4. If an interrupt occurs while the INPUT$ function is waiting for an input and the interrupt is not disabled or stopped, the interrupt subroutine will be processed immediately. Program execution will resume from the INPUT$ command when the interrupt subroutine has been completed. 5. Execution of the BASIC program will be paused and the BASIC LED indicator on the front of the Unit will blink slowly until the specified number of characters has been input.

Examples:	`> 10 OPEN #1, "TERM:"`
	`> 20 A$ = INPUT$(10,#1)`
	`> 30 PRINT A$`

See also:	WAIT

instr

Syntax:	INSTR ([<numerical expression>,] <string expression 1>,<string expression 2>)
Description:	Function Returns the position of the first occurrence of one string within another string.
Remarks:	<numerical expression> is the starting position in the string to be searched. Valid range: *[1…255]*. <string expression 1> is the string to be searched. <string expression 2> is the desired string. If the starting position is omitted then the search will begin from the first character of the string to be searched. If the starting position is greater than the length of the search string then 0 is returned. If the desired string cannot be found, 0 is returned. If the desired string is null, the result of the function is 1 or the starting position, if specified. If <string expression 1> is null, 0 will be returned. **Note:** 1. The search is case sensitive. This means the function differentiates between upper and lower case strings.

Examples:	`> 10 A = INSTR(2, "HELLO WORLD", "WORLD")`
	`> 20 PRINT A`
	`> RUN`
	`7`

See also:	

int

Syntax:	INT(<numerical expression>)
Description:	Function Returns the truncated integer part of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is integer. INT returns the largest integer value that is less than or equal to the argument. **Note:** 1. If the value of the argument is positive INT returns the same value as FIX, if the value of the argument is negative INT returns a value 1 less than FIX. 2. Refer to *Precautions on Real Numbers* in *7-4-4 Constants.*

Examples:	`> 10 A = INT(6.1)`
	`> 20 B = INT(6.9)`
	`> 30 C = INT(-6.1)`
	`> 40 D = INT(-6.9)`
	`> 50 PRINT A,B,C,D`
	`> RUN`
	`6  6  -7  -7`

See also:	FIX

intrb

Syntax:	INTRB
Description:	System Variable. Stores the number of the line that was executing when the interrupt occurred.
Remarks:	INTRB is a read-only system variable; it cannot be assigned a value. The value of INTRB is the line number of the line that was executing when an interrupt occurred. The INTRB function returns the line number where the program branch occurred when the interrupt was generated and execution branched to the interrupt subroutine.

Examples:	`> 10 ON KEY 1 GOSUB 1000`
	`> 20 KEY ON`
	`> 30 GOTO 30`
	`> 1000 PRINT "YOU PRESSED A KEY"`
	`> 1010 PRINT "THE BASIC LINE NUMBER THAT WAS INTERRUPTED BY THIS ROUTINE IS LINE "; INTRB`
	`> 1020 RETURN`

See also:	INTRR, INTRS

intrr

Syntax: INTRR

Description: System Variable. Stores a number indicating the source of the interrupt.

Remarks: INTRR is a read-only system variable; it cannot be assigned a value.

The value of INTRR is set only in the interrupt routine, outside the interrupt routine INTRR is 0.

The following table shows the relationship between the value of and INTRR the interrupt source.

INTRR	Interrupt Source
1,2	COM1, COM2
3,4	*Not used*
5	ALARM
6	TIMER
7	TIME$
10-19	KEY (0 TO 9)
20-100	*Not used*
101-199	ERROR (1 TO 99)
200	*Not used*
201-299	PC (1 TO 99)

Examples:

```
> 10 ON KEY 1 GOSUB 1000
> 15 KEY ON
> 20 GOTO 20
> 1000 PRINT "YOU PRESSED A KEY"
> 1010 PRINT "THE SOURCE OF THIS INTERRUPT IS TYPE "; INTRR
> 1020 RETURN
```

See also: INTRB, INTRS

intrs

Syntax: INTRS

Description: System Variable. Stores the number of the program line that has set up the interrupt processing routine.

Remarks: INTRS is a read-only system variable; it cannot be assigned a value.

On initialization of the BASIC program, the value of INTRS is set to 0. In the interrupt subroutine, INTRS contains the number of the program line that defined the interrupt that jumped to the interrupt subroutine.

The value of INTRS is set when program execution jumps to the interrupt subroutine and reverts to its original value when program execution is returned by the RETURN statement.

Examples:

```
> 10 ON KEY 1 GOSUB 1000
> 20 KEY ON
> 30 GOTO 30
> 1000 PRINT "YOU PRESSED A KEY"
> 1010 PRINT "THIS ISR WAS SET-UP IN LINE "; INTRS; " OF THE BASIC
PROGRAM"
> 1020 RETURN
```

See also: INTRB, INTRR

key on/off/stop

Syntax:	KEY [<key expression>] (ON | OFF | STOP)
Description:	Statement. Enables, disables or stops the KEY interrupt
Remarks:	<key expression> is an integer expression returning a value in the range : *[0…9]*. This number refers to the number keys on the keyboard. For instance, in the above example the interrupt connected to the '5' key on the keyboard is enabled. If the <key expression> is omitted then the action is performed for all keys. ON enables (unmasks) the KEY interrupt. Until KEY OFF is executed, program execution will automatically branch to the defined subroutine for interrupt processing when a KEY interrupt occurs. OFF disables the interrupt. All subsequent KEY interrupts are ignored. STOP masks and enables the interrupt. All subsequent KEY interrupts are held until they are unmasked by KEY ON. The KEY ON/OFF/STOP statements can be executed only after the ON KEY statement has been executed.

Examples:	`> 10 ON KEY 1 GOSUB 1000`
	`> 15 KEY ON`
	`> 20 GOTO 20`
	`> 1000 PRINT "YOU PRESSED A KEY"`
	`> 1010 PRINT "THE BASIC LINE NUMBER THAT WAS INTERRUPTED BY THIS ROUTINE IS LINE "; INTRL`
	`> 1020 RETURN`

See also:	ON KEY GOSUB/GOTO

left$

Syntax:	LEFT$ (<string expression>, < numerical expression>)
Description:	Function Returns the specified number of characters starting from the leftmost position in a string expression.
Remarks:	<string expression> is the string to be searched. <numerical expression> is the number of characters to be returned. Valid range: *[0…255]*. If the number of characters to be returned is 0, a null string is returned. If the number of characters to be returned is greater than the number of characters in the search string then the entire search string is returned.

Examples:	`> 10 A$ = LEFT$("HELLO WORLD", 5)`
	`> 20 PRINT A$`
	`> RUN`
	`HELLO`

See also:	MID$, RIGHT$

len

Syntax:	LEN (<string expression>)
Description:	Function Returns the number of characters in a string expression.
Remarks:	<string expression> is the input string to be counted. If the string expression is empty then the number of characters returned is 0. **Note:** 1. Non-printing characters and blanks are included in the count.

Examples:	`> 10 A = LEN("HELLO WORLD AGAIN")`
	`> 20 PRINT A`
	`> RUN`
	`17`

See also:	

let

Syntax:	[LET] <variable> = <expression>
Description:	Statement. Assigns a value of the <expression> on the right hand side of the statement to the variable on the left-hand side of the expression.
Remarks:	The keyword LET is optional in the assignment statement. It is sufficient to use the "=" character for assignment. Assigning character expressions to numeric variables and vice-versa result in a "TYPE MISMATCH" error (code B013). If the numerical expression on the right hand side of the assignment is a different type to the variable on the left hand side then the expression on the right hand side is converted to the type of the variable on the left hand side before assignment.

Examples:	`> 10 LET A = 12.34`
	`> 20 PRINT A`
	`> RUN`
	`12.34`
	`>`

See also:	

line input

Syntax:	LINE INPUT [;][# <port expression>,][<"prompt"> (;|,)]<character variable>
Description:	Statement. Inputs an entire line of characters from the keyboard, or some other input device, without delimiters.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. If the <port expression> is omitted the default terminal port is assumed. <prompt> is an optional text string that is output to the terminal to prompt for the input. If the port specified is not configured as a "TERM" or a "COMU" port then the prompt will be suppressed. Unlike the INPUT statement the '?' is only possible by including it in the prompt. All input will be assigned to the character variable until a carriage return is entered. The maximum length of the character string is 255 characters. while waiting for input the BASIC LED will be blinking. Commas and colons are treated as part of the input string. **Note:** If a semi-colon is added after the LINE INPUT, the cursor will remain on the same row after input.

Examples	`> 10 OPEN #1, "TERM:"`
	`> 20 LINE INPUT #1, "ENTER YOUR FULL NAME : "; A$`
	`> 30 PRINT "HELLO ";A$`
	`> 40 END`
	`> RUN`
	`ENTER YOUR FULL NAME : JOE BLOGGS<CR>`
	`HELLO JOE BLOGGS`
	`>`

See also:	INPUT

list

Syntax:	LIST [(<line number> | .)][-[(<line number> | .)]]
Description:	Command. To list the program in the current memory area to screen.
Remarks:	<line number> is any valid BASIC program line number in the range: *[1…65535]*. The combination of <line numbers> and dashes specifies the lines that will be listed. 1. If a single line number is given then only that line will be listed. 2. If a range of lines is specified (two line numbers separated by a dash) then that entire range will be listed. 3. If a line number followed by only a dash is specified then all program lines from the specified line number to the end of the program will be listed. 4. If a dash is followed by a line number then all program lines from the beginning of the program to the specified line number will be listed. 5. If only a . (dot) is used the line entered, or displayed, last will be listed. If the output of list does not fit on a screen device, it will be scrolled, displaying only the last most lines. The output of the LIST statement can be interrupted using either the CTRL+X or CTRL+B keys. If CTRL+B is used, output can be resumed by pressing CTRL+B again. The difference between LIST and LLIST is as follows: LIST outputs to port #1 on the ASC11 and ASC21 and port #3 on the ASC31. LLIST outputs to port #2. If the port is not configured correctly for output then a "CANNOT WRITE TO A READ_ONLY DEVICE" error (code 0059) may occur. **Note:** The output port is port #1 on ASC11 and ASC21 and port #3 on ASC31

Examples:	`> LIST 100-200`
	`100 REM THIS IS THE BEGINNING OF A SECTION TO BE LISTED`
	`110 PRINT "HELLO WORLD"`
	`200 REM THIS IS THE END OF A SECTION TO BE LISTED`
	`>`

See also:	LLIST

llist

Syntax:	LLIST [(<line number> | .)][-[(<line number> | .)]]
Description:	Command. To list the program in the current memory area to LPRT:
Remarks:	<line number> is any valid BASIC program line number in the range: *[1…65535]*. The combination of <line numbers> and dashes specifies the lines that will be listed. 1. If a single line number is given then only that line will be listed. 2. If a range of lines is specified (two line numbers separated by a dash) then that entire range will be listed. 3. If a line number followed by only a dash is specified then all program lines from the specified line number to the end of the program will be listed. 4. If a dash is followed by a line number then all program lines from the beginning of the program to the specified line number will be listed. 5. If only a . (dot) is used the line entered, or displayed, last will be listed. The output of the LLIST statement can be interrupted using the CTRL+X or CTRL+B keys. If CTRL-B is used, output can be resumed by pressing CTRL+B again. LLIST will be executed under the following conditions even if the OPEN command is not specified: "CTS_ON, RTS_OFF, DSR_ON" The difference between LIST and LLIST is as follows: LIST outputs to port #1 on the ASC11 and ASC21 and port #3 on the ASC31. LLIST outputs to port #2. If the port is not configured correctly for output then a "CANNOT WRITE TO A READ_ONLY DEVICE" error (code 0059) may occur. **Note:** The output port is port #2 on ASC11, ASC21 and ASC31.
Examples:	`> LLIST 100-200` `>`
See also:	LIST

load

Syntax:	LOAD # <port expression> [,<"communication definition string expression">]
Description:	Command. Loads an ASCII program to the ASCII Unit.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. See the OPEN statement for a description of the <communication definition string expression>. The communications conditions specified by the LOAD command's <communication definition string expression> are valid while the LOAD command is being executed. The original conditions are valid again after execution of the LOAD command is completed. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result. 3. The transfer can be interrupted during the LOAD operation by toggling the start/stop switch from start to stop. Refer to *7-1 Programming Procedure* for details on this procedure and for information about loading a BASIC program to the ASCII Unit.

Examples:	`> NEW`
	`> LOAD #1, "COMU:9600,8,N,2"`
	`> LIST`
	`10 REM THIS IS THE SAMPLE PROGRAM THAT WAS LOADED`
	`20 PRINT "HELLO WORLD"`
	`>`

See also:	OPEN, SAVE

loc

Syntax:	LOC (<port expression>)
Description:	Function Returns the number of bytes in the input buffer of the specified communications port.
Remarks:	<port expression> is a numerical expression returning a valid port number. Valid Range: *[1…3]*. **Note:** 1. The port must be opened for input. 2. Port #3 corresponds to the terminal port of the ASC31. 3. If Port #3 is specified on the ASC11 or ASC21, an "ILLEGAL FUNCTION CALL" error (code B005) will result.

Examples:	`> 10 OPEN #2, "COMU:"`
	`> 20 A = LOC(2)`
	`> 30 PRINT A`

See also:	

log

Syntax:	LOG (<numerical expression>)
Description:	Function. Calculates the natural logarithm of a numerical expression.
Remarks:	<numerical expression> must be greater than 0. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double precision floating point.

Examples:	`> 10 A = 0.4`
	`> 20 PRINT LOG(A)`
	`> RUN`
	`-.916291`

See also:	

lprint

Syntax: LPRINT [# <port expression>][USING "<print format>";] <expression>{(,|;|<space>)<expression>}

Description: Statement. Outputs data or text to a communications port. The device will be the LPRT regardless of the settings in the OPEN command.

Remarks:

<port expression> is an expression returning an integer in the range: *[1 or 2]*.

<expression> is any valid numerical or character expression.

If the <expressions> are separated by semicolons or spaces, and no formatting is used, they are output one immediately after the other on the output. If they are separated by commas, they are output with a tab character separating them on the output stream.

If the terminating comma, or semicolon, is omitted a carriage return is appended at the end of the output.

All numerical expressions are output with a space character either side of the expression. The space before the numerical expression can be used for an optional minus sign.

If no expressions are specified this statement outputs a carriage return on the output.

An optional format string can be used by applying the USING statement. The <print format> string is a string expression containing the following control characters.

When specifying several expressions, the format for each expression must be separated by a space. If there is no space, expression may not be output in the correct format.

Format type	Code	Description
String	!	Prints only the first character of a string.
	&&	Prints the first n characters, where n is the number of blanks enclosed between & plus 2.
	@	Prints the corresponding character string.
Numerical	#	Indicates a digit position.
	.	Inserts a decimal point at any desired place
	+	Specifies the position of the sign of the numeric value.
	-	If specified at the end of the numeric value it specifies that the sign for negative numbers is suffixed to numeric value.
	**	Fills the left-most unused positions of the numeric value with *. Also if - is specified at the end of a number then * will fill the sign position if the value is positive.
	\\	Prefixes \ at the beginning of the numeric value. \ allocates space for one digit position.
	**\	A combination of ** and \\.
	,	Separates the integer portion of a numeric value by a comma every three digits from the right of the integer part.
	^^^^	When specified at the end of a numeric field it specifies that exponential format should be used. The exponential format is (E+nn,D+nn).
	^^^^^	When specified at the end of a numeric field it specifies that specification format should be used. The specification format is (E+nnn,D+nnn).
	_	Allows the printing of the above special characters in the format string. If the special character is preceded by the '_' it is printed as normal.

	If the port number is omitted then port #1 is assumed for the PRINT statement and port #2 is assumed for the LPRINT statement. If the specified port is not configured as an output device then a "CANNOT WRITE TO A READ-ONLY DEVICE" error (code 0059) will occur. LPRINT will execute according to the "CTS_ON, RTS_OFF, DSR_ON" status, even if the OPEN command is not specified.
	Note: 1. The port must be opened for output. 2. If the specified number of digits to be output is greater that that specified in the numerical format a% will be output before the numerical value. 3. If <expression> does not correspond to the <print format> string then a "TYPE MISMATCH" error (code B013) will occur.

Examples:	`> 10 LINE INPUT "ENTER A STRING : "; A$`
	`> 20 OPEN #2, "LPRT:"`
	`> 30 LPRINT #2, A$`
	`> 40 PRINT "THE STRING YOU ENTERED - ";A$;" - WAS SENT TO THE PRINTER"`
	`> RUN`
	`ENTER A STRING : TEST`
	`THE STRING YOU ENTERED - TEST - WAS SENT TO THE PRINTER`

See also:	PRINT, WAIT

mid$	
Syntax:	MID$ (<string expression>, < numerical expression 1>[, < numerical expression 2>])
Description:	Function Returns the specified number of characters starting from the specified position in a string expression.
Remarks:	<string expression> is the string to be searched. <numerical expression 1> is the starting position in the string to be searched. Valid range: *[1…255]*. <numerical expression 2> is the number of characters to be returned. Valid range: *[0…255]*. If the number of characters to be returned is 0, or if the starting position is greater than the length of the search string, then a null string is returned. If the number of characters to be returned is greater than the number of characters remaining in the search string then the entire search string from the starting position is returned. If <numerical expression 2> is omitted, or exceeds the number of characters to the right of the starting position, then all of the characters to the right of the starting position are returned.

Examples:	`> 10 A$ = "HELLO WORLD AGAIN"`
	`> 20 B$ = MID$ (A$, 7, 5)`
	`> 30 PRINT B$`
	`> RUN`
	`WORLD`

See also:	LEFT$, RIGHT$

mid$

Syntax:	MID$ (<string 1>, < numerical expression 1>[, < numerical expression 2>]) = <string 2>
Description:	Statement Replaces a portion of one string expression with another string expression.
Remarks:	<string 1> is the string where the characters from <string 2> should be replaced. <string 2> is the string variable containing the letters to be replaced in <string 1> <numerical expression 1> is the starting position for replacing characters in <string 1>. Valid range: *[1…255]*. <numerical expression 2> is the number of characters from <string 2> to be inserted. Valid range: *[0…255]*. If this is omitted all of the characters from <string 2> will be inserted in <string 1>. No matter if <numerical expression 2> is omitted or not the total length of the original <string 1> cannot be exceeded. <numerical expression 1> cannot specify a character position that exceeds the number of characters in <string 1>.

Examples:	`> 10 A$ = "HELLO WORLD"`
	`> 20 MID$(A$, 7, 5)="AGAIN"`
	`> 30 PRINT A$`
	`RUN`
	`HELLO AGAIN`

See also:	LEFT$, RIGHT$, MID$ (function)

model

Syntax:	MODEL
Description:	System Variable. This read-only variable gives the model of the current ASCII Unit.
Remarks:	Returns the model of the current ASCII Unit. 1: ASC11 2: ASC21 3: ASC31

Examples:	`> 10 IF MODEL=1 THEN PRINT "THIS UNIT IS AN ASC11" : GOTO 50`
	`> 20 IF MODEL=2 THEN PRINT "THIS UNIT IS AN ASC21" : GOTO 50`
	`> 30 IF MODEL=3 THEN PRINT "THIS UNIT IS AN ASC31" : GOTO 50`
	`> 40 PRINT "UNKNOWN MODEL OF ASCII UNIT"`
	`> 50 END`
	`> RUN`
	`THIS UNIT IS AN ASC21`

See also:	

new	
Syntax:	NEW
Description:	Command. Clears the program and all variables in the current memory area.
Remarks:	This statement is used to clear a program area before beginning to write a new program. All ports are closed, all variables are cleared and the current program is deleted. Programs that are protected with a PNAME statement cannot be erased with NEW. The program name must first be cleared using PNAME " ". Attempts to clear a program protected with PNAME will result in a "PROTECTED PROGRAM" error (code 0062). If memory is write-protected with PMEM then NEW cannot clear the memory area. The write protection must first be turned OFF with PMEM OFF. The NEW command will work even if the program is password protected using PWORD.

Examples:	`> 10 REM DUMMY PROGRAM`
	`> 20 A = 5`
	`> LIST`
	`10 REM DUMMY PROGRAM`
	`20 A = 5`
	`> NEW`
	`> LIST`
	`>`

See also:	PINF, PNAME, PMEM, PWORD

oct$

Syntax:	OCT$(<numerical expression>)
Description:	Function Converts the numerical expression into an octal character string.
Remarks:	The numerical expression must be in the range: *[-2147483648…2147483647]*. The resulting character string will be in the octal range: *[0…7777777777]*. **Note:** 1. Single-precision or double-precision floating-point expressions will be truncated before the conversion.

Examples:	`> 10 PRINT "DEC", "OCTAL"`
	`> 20 FOR I = 1 TO 20`
	`> 30 PRINT I, OCT$(I)`
	`> 40 NEXT I`
	`> RUN`
	`DEC  OCTAL`
	` 1  1`
	` 2  2`
	` 3  3`
	` 4  4`
	` 5  5`
	` 6  6`
	` 7  7`
	` 8  10`
	` 9  11`
	` 10  12`
	` 11  13`
	` 12  14`
	` 13  15`
	` 14  16`
	` 15  17`
	` 16  20`
	` 17  21`
	` 18  22`
	` 19  23`
	` 20  24`
	`>`

See also:	HEX$

on alarm

Syntax:	ON ALARM <time expression> GOSUB (<line number> | <label>)
Description:	Statement. An interrupt is generated after the specified time elapses. When the specified time has elapsed after execution of ON ALARM, program execution branches to the interrupt subroutine at the specified line number.
Remarks:	<time expression> is an integer expression with a result in the range : *[1…864000]*. The units specified in the time expression are 0.1 seconds. This gives a total alarm range of 24 hours. <line number> specifies the start of the interrupt subroutine (ISR). <label> is a BASIC program label. It references a line number somewhere in the BASIC program. The ISR should be terminated by a RETURN statement. When the RETURN statement is reached execution will resume at the statement after the statement that was originally interrupted. ALARM interrupts are decrementing timer interrupts. The timer begins counting down from the time specified in the ON ALARM statement. When the timer reaches zero, program execution branches to the defined interrupt subroutine. The ALARM interrupts are one-shot interrupts. Once program execution has branched to the specified interrupt subroutine, the ALARM interrupt is invalid and another interrupt will not occur until another ON ALARM statement is executed and its specified time elapses. In this sense, ALARM interrupts are unlike the TIMER interrupts, which are "repetitive" interrupts. Only one ON ALARM interrupt can be valid at any one time. If two or more ON ALARM statements are set in a single program, only the last ON ALARM statement's interrupt will be valid. While the ALARM interrupt subroutine is being executed, other incoming interrupts will be recorded but not executed. These stopped interrupts will be executed in order of their priority after the ALARM interrupt subroutine has been completed. For more information on writing interrupts in BASIC, refer to *5-5 Interrupt Functions*.

Examples:	`> 10 A=0`
	`> 20 ON ALARM 30 GOSUB 1000`
	`> 30 ALARM ON`
	`> 40 IF A=0 THEN GOTO 40`
	`> 50 PRINT "END"`
	`> 60 END`
	`> 1000 PRINT "INTERRUPT OCCURRED"`
	`> 1010 A = 1`
	`> 1020 RETURN`
	`> RUN`
	`INTERRUPT OCCURRED`
	`END`

See also:	ALARM (ON | OFF | STOP), ON TIMER

on com

Syntax:	ON COM [<port expression>] GOSUB (<line number> | <label>) [, (CODE [= <code expression>] | BYTE = <byte expression>) | HEAD = <code expression> , (TERM [= <code expression >] | BYTE = <byte expression >)]
Description:	Statement. Defines an interrupt subroutine to handle interrupts from the communication ports and interrupt formats. There are 5 possible formats: 1) Interrupt after reception of any kind of data, 2) Reception of the specified character, 3) Reception of the specified number of bytes, 4) Reception of the specified header character and terminating character, or 5) Reception of the specified header character and number of bytes.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 2]*. If <port expression> is omitted ON COM operates on all ports. <line number> specifies the line at the start of the interrupt subroutine (ISR). <label> is a BASIC program label. It references a line number at the start of the ISR. The ISR should be terminated by a RETURN statement. When the RETURN statement is reached execution will resume at the statement that was originally interrupted. In the body of the ISR, the user should check the contents of the buffer before manipulating the data. It is possible that the ON COM has been held pending and in the meantime, the buffer data has been changed. <code expression> defines an ASCII code for a single byte character. Valid range is *[0…255]*. <byte expression> defines a number of bytes that can be entered before the interrupt takes place. Valid range is *[1…255]*. ON COM interrupts can be used in 5 different ways: 1. ON COM [<port>] GOSUB xxxx 2. ON COM [<port>] GOSUB xxxx, CODE = xx 3. ON COM [<port>] GOSUB xxxx, BYTE = yy 4. ON COM [<port>] GOSUB xxxx, HEAD = xx1, TERM = xx2 5. ON COM [<port>[GOSUB xxxx, HEAD = xx, BYTE = yy Case (1): Interrupt occurs when communication port buffer receives any character. Case (2): Interrupt occurs for a specific character xx. If xx is omitted then the default character string to initiate the interrupt is a CR. Case (3): Interrupt occurs after a specified number of characters have been received, i.e. on the yyth byte. The number specified must be in the range *[1…255]*. Case (4): Interrupt is enabled after the starting character xx1 is received and occurs after the terminating character xx2 is received. All characters from xx1 to xx2 are retained in the communication buffer. If the terminating character xx2 is omitted the default character string to initiate the interrupt is a CR. Case (5): Interrupt is enabled after the starting character xx is received and occurs after yy characters have been received. All yy characters from xx are retained in the communication buffer. Once an interrupt has been serviced, all subsequent BASIC interrupts are STOPped until the RETURN statement has been executed. On executing the corresponding ON COM ISR, all subsequent BASIC interrupts are STOPped, i.e. held pending during execution of the ISR. After the ISR has returned, pending interrupts are executed and the BASIC interrupts are returned to their previous states. Only one interrupt of the same type as the currently active interrupt can be held pending, subsequent interrupts of that type are lost.

	For more information on writing interrupts in BASIC, refer to the ON COM and ON PC examples in *5.5 Interrupt Functions.*
	Note: 1. The port must be opened for input. 2. Set the baud rate in the communications parameters as follows: Port #1 + port #2 ≦ 38.4 bps

Examples:	`> 10 ON COM 2 GOSUB 1000`
	`> 20 COM ON`
	`> 30 GOTO 30`
	`> 1000 PRINT "COMMUNICATIONS INTERRUPT OCCURRED"`
	`> 1010 A$ = INPUT$(LOC(2), #2)`
	`> 1020 PRINT A$; " RECEIVED"`
	`> 1030 RETURN`

See also:	COM (ON | OFF | STOP)

on error

Syntax:	ON ERROR GOTO (<line number> | <LABEL>)
Description:	Statement. Defines an interrupt subroutine to handle error processing.
Remarks:	<line number> is any valid line number in the range : *[0…65535]*. It specifies the first line in the error processing interrupt service routine. ON ERROR GOTO 0 disables the error processing routine. <label> is a BASIC program label. It references a line number somewhere in the BASIC program. When an error occurs the error code is assigned to the system variable ERR and the line number containing the source of the error is stored in the system variable ERL. The ON ERROR command cannot be used to branch to an interrupt subroutine for errors with error codes 80 to 99. Errors that occur during error processing cannot be handled by the BASIC error processing routine - they will be handled as normal by the system error handler. This will display an error message to the terminal and execution will terminate. For more information on writing interrupts in BASIC refer to the ON COM and ON PC examples in *5.5 Interrupt Functions.* While a command is being executed, the interrupt waits.

Examples:	`> 10 ON ERROR GOTO 1000`
	`> 20 ERROR 2`
	`> 30 END`
	`> 1000 PRINT "A SYNTAX ERROR HAS OCCURRED IN LINE ";ERL`
	`> 1010 END`
	`> RUN`
	`A SYNTAX ERROR HAS OCCURRED IN LINE 20`

See also:	ERC, ERL, ERR, ERROR, RESUME

on <expression>

Syntax:	ON <numerical expression> (GOTO | GOSUB) (<line number> | <label>) {, (<line number> |<label>)}
Description:	Statement. Branches to one of the specified subroutines depending on the value of a numerical expression.
Remarks:	The value of the <numerical expression> determines which subroutine will be executed. The <line number> specifies the starting position of the subroutine. A <label> can be used instead of a line number after a GOSUB statement. If the value of the expression is 1 then first branch will be taken, if 2 then the second branch will be taken, and so on. The valid range of the expression is : *[1…255]*. If the value of the expression is out of this valid range then an "ILLEGAL FUNCTION CALL" error (code B005) will result. If the value of the expression is 0 or if the value of the expression exceeds the number of branches after the GOTO/GOSUB then execution will continue with the next statement. If the GOSUB statement is used for branching then the subroutine should be terminated with a RETURN statement.

Examples:	`> 10 FOR I = 1 TO 3`
	`> 20 ON I GOSUB 100,200,300`
	`> 30 NEXT I`
	`> 40 END`
	`> 100 PRINT "1"`
	`> 110 RETURN`
	`> 200 PRINT "2"`
	`> 210 RETURN`
	`> 300 PRINT "3"`
	`> 310 RETURN`
	`> RUN`
	`1`
	`2`
	`3`

See also:	

on key

Syntax:	ON KEY [<key expression>] (GOTO | GOSUB) (<line number> | <label>)
Description:	Statement. An interrupt subroutine is executed when the specified key (0 to 9) is input from the keyboard. (Valid only when a terminal is connected.)
Remarks:	<key expression> is an integer expression returning a value in the range : *[0…9]*. there should only be one ON KEY GOTO/GOSUB statement for each <key expression>. If the <key expression> is omitted the interrupt is activated for all key inputs. <line number> specifies the line at the start of the interrupt subroutine (ISR). <label> is a BASIC program label. It references a line number somewhere in the BASIC program. If the GOSUB statement is used for branching, the ISR should be terminated by a RETURN statement. When the RETURN statement is reached execution will resume at the statement after the statement that was originally interrupted. A value outside this range results in a “FORMAT ERROR” (code B067). The interrupt will be on standby when the PC READ/PC WRITE (including @) commands are executing for an ON KEY GOSUB statement. The port connecting the keyboard must first be opened using an OPEN statement. Pressing the “1” key on the keyboard will execute the key(1) branch, pressing the “2” key will execute the key(2) branch, and so on. During the execution of ON KEY GOSUB routines all BASIC interrupts are STOPped until the RETURN statement has been executed. On executing the corresponding ON KEY ISR, all subsequent BASIC interrupts are stopped, i.e. held pending during execution of the ISR. After the ISR has returned, pending interrupts are executed and the BASIC interrupts are returned to their previous states. Only one interrupt of the same type as the currently active interrupt can be held pending, subsequent interrupts of that type are lost. For more information on writing interrupts in BASIC refer to the ON COM and ON PC examples in *5.5 Interrupt Functions.*

Examples:	`> 10 ON KEY 1 GOSUB 60`
	`> 20 ON KEY 2 GOSUB 80`
	`> 30 ON KEY 3 GOSUB 100`
	`> 40 KEY ON`
	`> 50 GOTO 50`
	`> 60 PRINT "1 PRESSED"`
	`> 70 RETURN`
	`> 80 PRINT "2 PRESSED"`
	`> 90 RETURN`
	`> 100 PRINT "3 PRESSED"`
	`> 110 RETURN`
	`> RUN`

See also:	KEY (ON | OFF | STOP)

on pc

Syntax:	ON PC [<interrupt number>] GOSUB (<line number> | <label>)
Description:	Statement. Executes the specified interrupt number's interrupt subroutine in response to allocated I/O interrupts or IOWR (#CC00) instruction interrupts from the PC (CPU Unit).
Remarks:	The <interrupt number> is an integer expression returning a value in the range : *[1…99]*. If the <interrupt number> is omitted then the interrupt subroutine will be executed for any PC interrupt. The default value of the interrupt number in IR n+2 is 00. This means that no PC interrupt has been requested. The <line number> specifies the line at the start of the interrupt subroutine (ISR). A <label> can be used instead of a line number after a GOSUB statement. PC interrupts are enabled by the PC ON statement and disabled by the PC OFF statement. The PC STOP statement masks and enables the PC interrupts so the interrupts are recorded but the ISRs are not executed until the next PC ON statement unmasks the interrupts. The ISR should be terminated by a RETURN statement. When the RETURN statement is reached, execution will resume at the statement where the program was interrupted. The PC can initiate a BASIC interrupt by either of the following methods: 1. Allocated I/O (PC Interrupt Flag) Interrupts The PC interrupt number can be set in the PC's CPU Unit by turning ON the corresponding bit in IR n+2 bits 00 to 07. By turning bit 04 of IR n (the PC Interrupt Flag) from OFF to ON at the same time, the specified PC interrupt will be executed if PC interrupts have been enabled. 2. IOWR (#CC00) Instruction Interrupts When the interrupt number is indicated to the ASCII Unit by the IOWR (#CC00) instruction from the PC's CPU Unit, the specified PC interrupt will be executed if PC interrupts have been enabled. PC interrupts cannot be executed if the ASCII Busy Flag is set. To execute an interrupt the ASCII Busy Flag should be checked. While the PC interrupt subroutine is being executed, other incoming interrupts will be recorded but not executed. These stopped interrupts will be executed in order of their priority after the PC interrupt subroutine has been completed. For more information on PC interrupts, refer to *6-3 Details of the Data Exchange Methods*. For more information on writing interrupts in BASIC, refer to the ON COM and ON PC examples in *5-5 Interrupt Functions*.

Examples:	`> 10 ON PC 1 GOSUB 70`
	`> 20 ON PC 2 GOSUB 90`
	`> 30 ON PC 3 GOSUB 110`
	`> 50 PC ON`
	`> 60 GOTO 60`
	`> 70 PRINT "PC INTERRUPT #1"`
	`> 80 RETURN`
	`> 90 PRINT " PC INTERRUPT #2"`
	`> 100 RETURN`
	`> 110 PRINT " PC INTERRUPT #3"`
	`> 120 RETURN`
	`> 130 END`

See also:	PC (ON | OFF | STOP)

on time$	
Syntax:	ON TIME$ = <time string expression> GOSUB (<line number> | <label>)
Description:	Statement. Defines an interrupt subroutine to handle time$ interrupts.
Remarks:	<time string expression> is an absolute time in the format "hh:mm:ss". Where hh has a range *[00…23]*, mm has a range *[00…59]* and ss has a range *[00…59]*. Wildcards, specified using '*', can be used to define multiple time interrupts, for instance: "02:30:**" means that from 0230 hrs 60 interrupts will occur at one second intervals. "02:3*:00" means that from 0230 hrs 10 interrupts will occur at one-minute intervals. <line number> specifies the start of the interrupt subroutine (ISR). <label> is a BASIC program label. It references a line number somewhere in the BASIC program. The time$ interrupts are enabled by the TIME$ ON statement and disabled by the TIME$ OFF statement. The TIME$ STOP statement masks and enables the time$ interrupts so the interrupts are recorded but the ISRs are not executed until the next TIME$ ON statement unmasks the interrupts. The ISR should be terminated by a RETURN statement. When the RETURN statement is reached, execution will resume at the statement where the program was interrupted. Only one time$ interrupt can be valid at any one time. If two or more ON TIME$ statements are set in a single program, only the last ON TIME$ statement's interrupt will be valid. While the time$ interrupt subroutine is being executed, other incoming interrupts will be recorded but not executed. These stopped interrupts will be executed in order of their priority after the time$ interrupt subroutine has been completed. For more information on writing interrupts in BASIC, refer to the ON COM and ON PC examples in *5-5 Interrupt Functions*.

Examples:	`> 10 ON TIME$ = "08:00:00" GOSUB 100`
	`> 20 TIME$ ON`
	`> 30 GOTO 30`
	`> 100 PRINT "WAKE-UP!!!"`
	`> 110 RETURN`

See also:	TIME$ (ON | OFF | STOP), ON ALARM

on timer

Syntax:	ON TIMER <time expression> GOSUB (<line number> | <label>)
Description:	Statement. Defines an interrupt subroutine to handle recurring timer interrupts. When the specified time has elapsed after execution of ON TIMER, program execution branches to the specified interrupt subroutine. After completion of the interrupt subroutine, the timer is reset and the subroutine will be executed again after the specified time has elapsed. The subroutine will be executed repeatedly until the TIME OFF statement is executed.
Remarks:	<time expression> is an integer expression with a result in the range : *[1…864000]*. The units specified in the time expression are 0.1 seconds. This gives a total alarm range of 24 hours. <line number> specifies the start of the interrupt subroutine (ISR). <label> is a BASIC program label. It references a line number somewhere in the BASIC program. The timer interrupts are enabled by the TIMER ON statement and disabled by the TIMER OFF statement. The TIMER STOP statement masks and enables the timer interrupts so the interrupts are recorded but the ISRs are not executed until the next TIMER ON statement unmasks the interrupts. The ISR should be terminated by a RETURN statement. When the RETURN statement is reached, execution will resume at the statement where the program was interrupted. Timer interrupts are decrementing timer interrupts. The timer begins counting down from the time specified in the timer statement. When the timer reaches zero, program execution branches to the defined interrupt subroutine as long as the interrupt is enabled (not masked). Timer interrupts are “repetitive” interrupts. The timer interrupt remains valid after program execution has branched to the specified ISR, so the same ISR will be executed repeatedly each time that the timer counts down to zero. The timer interrupts will repeat until they are disabled with the TIMER OFF statement, thus these are interval timer interrupts. In this sense, timer interrupts are unlike the alarm interrupts, which are “one-shot” interrupts.

Examples:	`> 5 T = 0`
	`> 10 ON TIMER 300 GOSUB 100`
	`> 20 TIMER ON`
	`> 30 GOTO 30`
	`> 100 T = T+300`
	`> 110 PRINT T; " MILLISECONDS HAVE ELAPSED SINCE THE START OF EXECUTION."`
	`> 120 RETURN`

See also:	TIMER (ON | OFF | STOP)

open

Syntax: OPEN # <port expression>, <"communication definition string expression">

Description: Statement. Opens a communication port with the specified device, port communications conditions, and transfer control signal specifications.

Remarks: <port expression> is an expression returning an integer in the range: *[1… 3]*. If the port expression is omitted, port #1 is the default.

The <communication definition expression> is a string containing the following elements:

"<device>**:**[<baud rate> [**,** <data length> [**,** <parity> [**,** <stop bits> [, CS [, RS [, DS [, XN]]]]]]]]]"

If any of the elements of the <communication definition expression> are omitted, the following default setting is used:

"xxxx, 8, N, 2, yy, yy, yy, yy"

Refer to page 175

Specified in the DM settings

The <communication definition expression> format of the C200H-ASC02 is still supported but new BASIC programs should use the new format.

<device> describes the type of the device that is connected to the port. The following table shows the input and output buffer usage and transfer control signal (RS and ER) status for each type of the device.

Device	Peripheral devices (external devices)	Buffer used	When the port is opened			During execution of I/O commands (PRINT#, INPUT#)	
			RS		ER	RS	ER
			At RS_OFF under OPEN# command	At RS_ON under OPEN# command		At RS_ON/OFF under OPEN# command	
TERM	Terminal	Both send and receive buffers	OFF	ON	Unchanged	ON	Unchanged
SCRN	Display	Send buffer only	OFF	ON	Unchanged	ON	Unchanged
KYBD	Keyboard	Receive buffer only	OFF	ON	Unchanged	ON	Unchanged
COMU	General communications devices	Both send and receive buffers	OFF	ON	Unchanged	ON	Unchanged
LPRT	Printer	Send buffer only	OFF	ON	Unchanged	ON	Unchanged
EKPRT	Printer	Send buffer only	OFF	ON	Unchanged	ON	Unchanged
NKPRT	Printer	Send buffer only	OFF	ON	Unchanged	ON	Unchanged

The <baud rate> can be set to **300, 600, 1200, 2400, 4800, 9600, 19200,** or **38400**. The default baud rate setting is 9600 bps.

The <data length> can be set to **7** or **8**. If omitted the default data length is 8 bits.

The <parity> can be set to **N** (none), **E** (even), or **O** (odd). If omitted the default parity is **N** (none).

The number of <stop bits> can be set to **1** or **2**. If omitted the default number of stop bits is **2**.

The **CS, RS, DS** and **XN** fields determine whether these signal lines will be used to control communications.

CS = CS_ON: The ASCII Unit will monitor CTS signal during PRINT commands.

CS = CS_OFF: The ASCII Unit will not monitor CTS signal during PRINT commands.

RS = RS_ON: The ASCII Unit will turn ON the RTS signal when the communications port is opened and it will remain ON until the port is closed. BASIC RS control is disabled.

RS = RS_OFF: The ASCII Unit will turn ON the RTS signal only when I/O commands such as PRINT and INPUT are being executed. The RTS signal will be OFF at other times.

DS = DS_ON: The ASCII Unit will check the DSR signal when the input commands (PRINT, LPRINT) are executed. If the DSR signal is ON then the device is assumed to be ready. If the DSR signal is OFF then the device is assumed to be unable to communicate and the ASCII Unit will wait until the DSR signal turns ON.

DS = DS_OFF: The ASCII Unit will not check DSR signal when the port is opened.

XN = XN_ON: Specifies that Xon/Xoff flow control will be used for both the ASCII Unit's input buffer and the peripheral device's input buffer.

When the ASCII Unit is receiving data, the Unit will send the Xoff code will be sent when its input buffer is 3/4 full (384 bytes) requesting the other device to pause the data transfer. When the buffer drops to 1/4 full (128 bytes) the Xon code will be sent to request resumption of the data transfer.

When the ASCII Unit is sending data, the Unit will stop sending data when a Xoff signal is received while a PRINT command is being executed. The Unit will resume sending data when the Xon signal is received.

X = XN_OFF: Specifies that Xon/Xoff flow control will not be used.

The default values for CS, RS, DS, and XN are the values of the devices that have been opened.

When a LOAD or SAVE command is executed, the communication parameters specified in that command will be valid for the duration of the command. The original communications parameters will be restored after completion of the LOAD or SAVE command.

Note:

1. Port #3 corresponds to the terminal port of the ASC31.
2. If Port #3 is specified on the ASC11 or ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
3. Refer to the C200H-ASC02 Operation Manual for details on the communications settings for the ASC02.

Examples:

```
> 10 OPEN #2, "COMU:9600, 8, N, 2, CS_ON, RS_OFF, DS_OFF,
XN_OFF"
> 20 PRINT #2, "HELLO WORLD"
```

See also: CLOSE, LOAD

The following defaults will be used for the devices if the communications parameters and valid signals are not specified.

Device name	Baud rate	Data format	Valid signals
TERM	DM Area settings	8-bit, no parity, 2 stop bits	CS_OFF, RS_ON, DS_OFF, XN_OFF
SCRN			CS_ON, RS_ON, DS_OFF, XN_OFF
KYBD			CS_ON, RS_ON, DS_OFF, XN_OFF
COMU			CS_ON, RS_ON, DS_OFF, XN_OFF
LPRT **EKPRT** **NKPRT**			CS_OFF, RS_ON, DS_ON, XN_OFF

Note: The CS, RS, DS, and XN signals are not relevant for port #2 of the C200H-ASC21.

option base

Syntax:	OPTION BASE (0 | 1)
Description:	Statement. Declares the default subscript for the first array element.
Remarks:	The OPTION BASE declaration can be made only once; before any variables have been declared. If the declaration is not made the first element in an array has subscript 0 by default.

Examples:	`> 10 OPTION BASE 1`
	`> 20 DIM A(2)`
	`> 30 FOR I = 0 TO 1`
	`> 40 A(I) = I`
	`> 50 NEXT I`
	`> 60 FOR I = 0 TO 1`
	`> 70 PRINT A(I)`
	`> 80 NEXT I`
	`> RUN`
	`BAD SUBSCRIPT ERROR IN LINE 40`
	`> 10 OPTION BASE 1`
	`> 20 DIM A(2)`
	`> 30 FOR I = 1 TO 2`
	`> 40 A(I) = I`
	`> 50 NEXT I`
	`> 60 FOR I = 1 TO 2`
	`> 70 PRINT A(I)`
	`> 80 NEXT I`
	`RUN`
	`1`
	`2`

See also:	DIM

option length

Syntax:	OPTION LENGTH <numerical expression>
Description:	Statement. Declares the maximum length for string variables and sets the static method for allocation of memory to string variables.
Remarks:	<numerical expression> specifies the maximum length of the string variables. It has a valid range of *[1…255]*. Put the OPTION LENGTH statement in the first line of the program or just after the PWORD statement if a PWORD statement is being used, too. The dynamic allocation method is used when no settings have been made for the allocation of memory to string variables. With this method, memory is reallocated in the character string variable area as the program is being executed. When the OPTION LENGTH statement is used to declare the required length, memory is allocated before the program is executed, so memory is not reallocated in the character string variable area as the program is being executed. The OPTION LENGTH statement can be used only once. It must be used before any string variables have been declared.

Examples:	`> 10 OPTION LENGTH 32`
	`> 20 A$ = "THIS STRING IS LONGER THAN THE SPECIFIED LENGTH"`
	`> 30 PRINT A$`
	`> RUN`
	`THIS STRING IS LONGER THAN THE S`

See also:	DIM

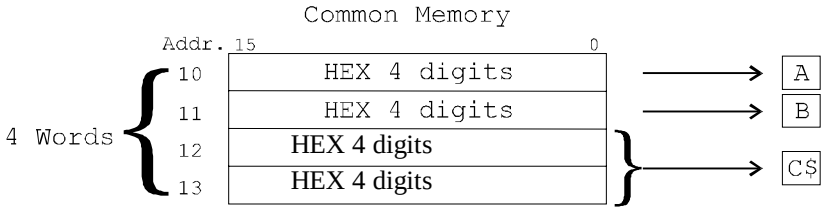
pc on/off/stop

Syntax:	PC [<pc interrupt number>] (ON | OFF | STOP)
Description:	Statement. Enables, disables or stops interrupts from the PC.
Remarks:	<pc interrupt number> specifies which interrupt from the PC to turn ON, OFF or STOP. There are 99 PC interrupts that can be defined. The valid range is *[1…99]*. If the <pc interrupt number> is omitted this statement applies to all PC interrupts. ON enables (unmasks) the PC interrupts. When a PC interrupt occurs, program execution will be branched to the specified interrupt subroutine for interrupt processing. The PC interrupts are enabled until they are disabled with the PC OFF statement. OFF disables the interrupt. All subsequent PC interrupts are ignored. STOP masks and enables the interrupt. If a PC interrupt is received, it is stored in memory but program execution is not branched to the interrupt subroutine. Program execution will be branched to the subroutine when the interrupt is unmasked with the PC ON statement. The PC ON/OFF/STOP statements can be executed meaningfully only after the ON PC statement has been executed.

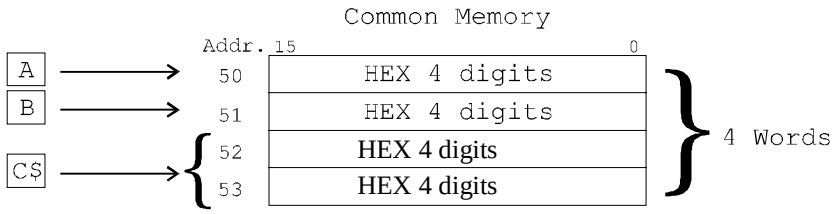
Examples:	`> 10 ON PC 1 GOSUB 70`
	`> 20 ON PC 2 GOSUB 90`
	`> 30 ON PC 3 GOSUB 110`
	`> 40 PC ON`
	`> 50 GOTO 50`
	`> 60 PRINT "PC INTERRUPT #1"`
	`> 70 RETURN`
	`> 80 PRINT " PC INTERRUPT #2"`
	`> 90 RETURN`
	`> 100 PRINT " PC INTERRUPT #3"`
	`> 110 RETURN`

See also:	ON PC

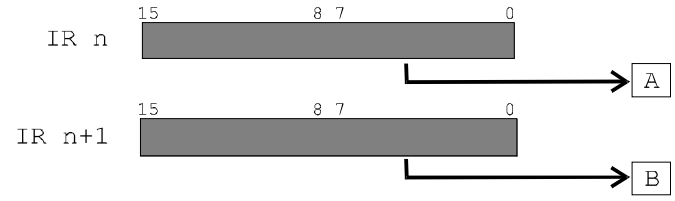
pc eget

Syntax:	[<result> =] PC EGET #<common memory address> , <number>, "<pc format> {, <pc format>}"; <variable> {, <variable>}
Description:	Statement. Reads data from the common memory area of the ASCII Unit.
Remarks:	<common memory address> gives the starting address in common memory area of the data to be read. It is an integer with a valid range of : *[0…89]*. <number> is the number of words of data to read. It is an integer value with a range of : *[1…90]*. <pc format> gives the format of the variables that are retrieved. The allowable formats in the format string are outlined in PC READ. Of course the complexity of the format string is ultimately limited by the size of the input buffer allowed to enter a single program line. <variable> is any valid variable name. It must be of the same type as specified in the format string. The PC EGET command can optionally return a <result> that can be assigned to a variable. This <result> indicates the success, or otherwise of the PC EGET command. If the <result> is 0 then the command was successfully carried out and the data read is valid. If the <result> is -1 then the PC EGET could not be carried out because the common memory area was being used by another PC EGET. If the format string and the variable types do not match then a "FORMAT ERROR" (code B067) will occur. Also if the format string specifies more memory than is allocated in the common memory area then a "FORMAT ERROR" (code B067) will result. The data in the common memory area should first be prepared by the IOWR #00xx command in the PLC program. this command is used to write up to 90 words of data from the PLC to the common data area of the ASCII Unit. There is no synchronization between the PC EGET command in the BASIC program and the IOWR #00xx command in the PLC program. For a detailed description of how to use the PC EGET statement and the IOWR #00xx PLC command refer to *6.3. Details of the Data Exchange Methods*.
Examples:	 **PC EGET #0, 4, "1H4, 1H4, 2H4"; A, B, C$**
See also:	PC EPUT, PC GET

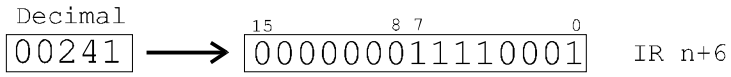
pc eput

Syntax:	[<result> =] PC EPUT #<common memory address> , <number>, “<pc format> {, <pc format>}”; <numerical expression> {, <numerical expression>}
Description:	Statement. Writes data to the common memory area of the ASCII Unit.
Remarks:	<common memory address> gives the starting address in common memory area of where the data should be written. It is an integer with a valid range of : *[0…89]*. <number> is the number of words of data to write. It is an integer value with a range of : *[1…90]*. <pc format> gives the format of the variables that are retrieved. For full details of these formats see the PC READ statement. <numerical expression> is any valid BASIC is any valid numerical expression with an integer result in the range : *[-32768…32767]*. If the value of the numerical expression is not within this range then no action will result. The PC EPUT command can optionally return a <result> that can be assigned to a variable. This <result> indicates the success, or otherwise of the PC EPUT command. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC EPUT could not be carried out because the common memory area was being used by another PC EPUT or IORD #00xx command. If the format string and the variable types do not match then a “FORMAT ERROR” (code B067) will occur. Also if the format string specifies more memory than is allocated for writing in the common memory area then a “FORMAT ERROR” (code B067) will result. The data written to the common memory area by PC EPUT may be read by the IORD #00xx command in the PLC program. this command is used to read up to 90 words of data from the common data area of the ASCII Unit to the PLC. There is no synchronization between the PC EPUT command in the BASIC program and the IORD #00xx command in the PLC program. For a detailed description of how to use the PC EPUT statement and the IORD #00xx PLC command refer to *6.3. Details of the Data Exchange Methods*.
Examples:	 **PC EPUT #0, 4, “1H4, 1H4, 2H4”; A, B, C$**
See also:	PC EGET, PC PUT

pc get

Syntax:	PC GET <variable> [,<variable>]
Description:	Statement. Reads data from IR n bits 0 through 15 to the first variable and IR n+1 bits 0 through 15 to the second variable. The data is converted from hexadecimal to decimal.
Remarks:	<variable> is any valid BASIC variable name. If the second <variable> is omitted only the byte from IR n is read. The hexadecimal value in bits 0-15 of IR n is converted to an integer decimal value *[-32768…32767]* and assigned to the first variable specified in the PC GET statement. The hexadecimal value in bits 0-15 of IR n+1 is converted to an integer decimal value *[-32768…32767]* and assigned to the second variable specified in the PC GET statement. For a detailed description of how to use the PC GET statement refer to *6.3. Details of the Data Exchange Methods*.
Examples:	 **PC GET A,B**
See also:	PC EGET, PC PUT

pc put

Syntax:	PC PUT <numerical expression>
Description:	Statement. Converts a decimal value to hexadecimal and writes the data to IR word n+6.
Remarks:	<numerical expression> is any valid BASIC is any valid numerical expression with an integer result in the range : *[-32768…32767]*. If the numerical expression is not an integer it will be rounded to an integer before transfer. If the value of the numerical expression is not within this range then zero will be written to IR n+6. The numerical expression is stored in IR n+6 in hexadecimal format. The data will be written to the PC in the next I/O refreshing cycle after the PC PUT statement is executed. Execution of the ASCII Unit's BASIC program will not be paused.
Examples:	 **PC PUT 241**
See also:	PC EPUT, PC GET

pc qread @

Syntax:

[<result> =] PC QREAD "<data area>, <address>, <number>, <pc format> {, <pc format>}"; <variable> {, <variable>}

Description:

Statement. Reads data from the specified PC memory area by issuing an IOWR request to the PC.

Remarks:

<data area> specifies which PC data area is to be used for the transfer. Any of the following specifications for the data area are valid.

Other words in the CS1-series CPU Units cannot be addressed. Refer to *6-6 Restrictions for CS1-series PCs* for details.

C200HE/HG/HX

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	C200HS/HE/ HG/HX	DM 0000 to DM 6655	@D	0000 to 6655
IR/SR Area		000 to 511	@R	0000 to 0511
LR Area		LR 00 to LR 63	@L	0000 to 0063
HR Area		HR 00 to HR 99	@H	0000 to 0099
AR Area		AR 00 to AR 27	@A	0000 to 0027
TC Area		T/C 000 to T/C 511	@G	0000 to 0511

CS1

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	CS1	D00000 to D06655	@D	0000 to 6655
I/O Area (CIO Area)		0000 to 0511	@R	0000 to 0511
Data Link Area (CIO Area)		1000 to 1063	@L	0000 to 0063
Holding Bit Area		H000 to H099	@H	0000 to 0099
		H100 to H127	@A	0000 to 0027
Timer Area		T0000 to T0511	@G	0000 to 0511

<address> specifies the starting address of data transfer within the specified data area. The starting address must be consistent with the address range of the specified data area. If the address given lies outside the allowable range then a “FORMAT ERROR” (code B067) will result.

<number> specifies the number of data words to be read from the starting address given. The maximum number of data words that can be transferred using PC QREAD is 128 words. This is only possible with a C200HE/HG/HX PLC with no remote I/O. If any of the data to be read lie outside the allowable range then a “FORMAT ERROR” (code B067) error will result.

<pc format> gives the format of the variables that are retrieved. For full details of these formats see the PC READ statement.

<variable> is any valid variable name. It must be of the same type as that specified in the format string otherwise a “FORMAT ERROR” will result.

The PC QREAD command can optionally return a <result> that can be assigned to a variable. This <result> indicates the success, or otherwise of the PC QREAD command. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC QREAD could not be carried out because another data transfer was active.

The PC QREAD statement should be used in conjunction with the CPU Unit’s IOWR (#FD00) instruction. By executing the PC QREAD statement, the ASCII Unit requests execution of the IOWR instruction (through bit 15 of IR n+5); the requested data will be stored in the specified variables when it is transferred by the IOWR (#FD00) instruction. The ASCII Unit will wait for the data transfer by the IOWR (#FD00) instruction.

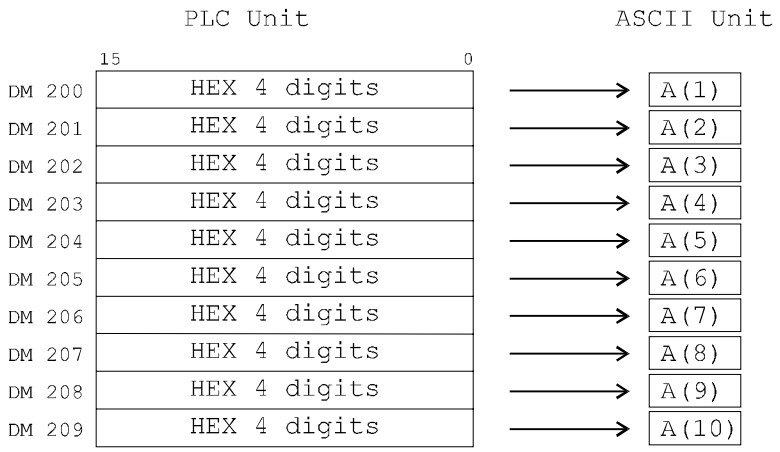
The data area, starting address, and number of bytes must be the same in both the PC QREAD statement and the IOWR instruction. If the starting address settings are different, the setting in the IOWR instruction will be used. If the settings for the number of bytes are different, an error (code 0081) will occur.

Refer to *Section 6 Data Exchange with the CPU Unit* for more details on how to use the PC QREAD statement.

The words specified for PC QREAD@ are not necessarily the words that will be transferred. The addresses of the words are stored in words n+8 and n+9 and used for monitoring by the CPU Unit. The words specified for IOWR will be transferred. (PC QREAD@ differs in this point from PC READ@, which actually uses the addresses specified for it.)

If the addresses specified for IOWR cannot be used in PC QREAD@, specify dummy addresses (in which case the CPU Unit will not monitor the specified words).

Examples:



PC QREAD “@D, 200, 10, S10H4”; A(1)

See also: PC QWRITE, PC READ, WAIT

pc qwrite @

Syntax:

[<result> =] PC QWRITE "<data area>, <address>, <number>, <pc format> {, <pc format>}"; <numerical expression> {, <numerical expression>}

Description:

Statement. Writes data to the specified PC memory area by issuing an IORD request to the PC.

Remarks:

<data area> specifies which PC data area is to be used for the transfer. Any of the following specifications for the data area are valid.

Other words in the CS1-series CPU Units cannot be addressed. Refer to *6-6 Restrictions for CS1-series PCs* for details.

C200HE/HG/HX

Area	PC	Address	Designation method	
			Code	First word address
DM Area	C200HE/ HG/HX	DM 0000 to DM 6143	@D	0000 to 6143
IR/SR Area		000 to 511	@R	0000 to 0511
LR Area		LR 00 to LR 63	@L	0000 to 0063
HR Area		HR 00 to HR 99	@H	0000 to 0099
AR Area		AR 00 to AR 27	@A	0000 to 0027
TC Area		T/C 000 to T/C 511	@G	0000 to 0511

CS1

Area	PC	Address	Designation method	
			Code	First word address
DM Area	CS1	D00000 to D06655	@D	0000 to 6655
I/O Area (CIO Area)		0000 to 0511	@R	0000 to 0511
Data Link Area (CIO Area)		1000 to 1063	@L	0000 to 0063
Holding Bit Area		H000 to H099	@H	0000 to 0099
		H100 to H127	@A	0000 to 0027
Timer Area		T0000 to T0511	@G	0000 to 0511

<address> specifies the starting address of data transfer within the specified data area. The starting address must be consistent with the address range of the specified data area. If the address given lies outside the allowable range then a "FORMAT ERROR" (code B067) will result.

<number> specifies the number of data words to write from the specified starting address. The maximum number of data words that can be transferred using PC QWRITE is 128 words. This is only possible with a C200HE/HG/HX PLC with no remote I/O. If any of the data to be read lie outside the allowable range then a "FORMAT ERROR" (code B067) will result.

<pc format> gives the format of the variables that are retrieved. For full details of these formats see the PC READ statement.

<numerical expression> is any valid BASIC is any valid numerical expression with an integer result in the range : *[-32768…32767]*. If the numerical expression is not an integer it will be rounded to an integer before transfer.

The PC QWRITE command can optionally return a <result>. This <result> indicates the success, or otherwise of the PC QWRITE command. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC QWRITE could not be carried out because another data transfer was active.
The PC QWRITE statement should be used in conjunction with the CPU Unit's IORD (#FD00) instruction. By executing the PC QWRITE statement, the ASCII Unit requests execution of the IORD instruction (through bit 14 of IR n+5) in preparation for the transmission of the data from the specified variables. The data will be read from the ASCII Unit when the IORD (#FD00) instruction is executed. The ASCII Unit will wait from the time the PC QWRITE command is executed until the data is transferred by the IORD (#FD00) instruction.

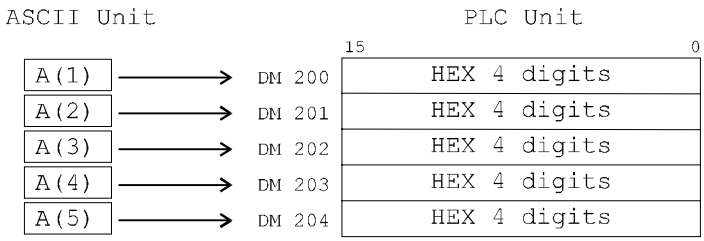
The data area, starting address, and number of bytes must be the same in both the PC QWRITE statement and the IORD instruction. If the starting address settings are different, the setting in the IORD instruction will be used. If the settings for the number of bytes are different, an error (code 0081) will occur.

Refer to *Section 6 Data Exchange with the CPU Unit* for more details on how to use the PC QWRITE statement.

The words specified for PC QWRITE@ are not necessarily the words that will be transferred. The addresses of the words are stored in words n+8 and n+9 and used for monitoring by the CPU Unit. The words specified for IORD will be transferred. (PC QWRITE@ differs in this point from PC WRITE@, which actually uses the addresses specified for it.)

If the addresses specified for IORD cannot be used in PC QWRITE @, specify dummy addresses (in which case the CPU Unit will not monitor the specified words).

Examples:



PC QWRITE "@D, 200, 5, 1H4 1H4 1H4 1H4 1H4"; A(1), A(2), A(3), A(4), A(5)

See also: PC QREAD, PC WRITE, WAIT

pc read

Syntax: [<result> =] PC READ "<pc format> {, <pc format>}"**;** <variable> {**,** <variable>}

Description: Statement. When a PC READ request is received from the CPU Unit, the data is read from the specified PC data area and stored in the specified variable with the specified PC format.

Remarks: The <pc format> indicates the data format of the specified data. The following table shows the various data formats that can be specified.

Format	Syntax	Description
I Format	*m* I *n*	Treats the m words as BCD data and stores n digits of data from each word in the variable. *m* : the number of words *n* : the number of digits read from each word (1 to 4) **_n_ / Digits read** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
H Format	*m* H *n*	Treats the m words as hexadecimal data and stores n digits of data from each word in the variable. *m* : the number of words *n* : the number of digits read from each word (1 to 4) **_n_ / Digits read** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
O Format	*m* O *n*	Treats the m words as octal data and stores n digits of data from each word in the variable. *m* : the number of words *n* : the number of digits read from each word (1 to 4) **_n_ / Digits read** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
B Format	*m* B *n*	Treats the m words as binary data, converts the nth bit of each word to decimal, and stores the converted values in the variable. *m* : the number of words *n* : the bit read from each word (0 to 15) **_n_ / Bit read** 0 / bit 0 1 / bit 1 2 / bit 2 3 / bit 3 4 / bit 4 5 / bit 5 6 / bit 6 7 / bit 7

		8 bit 8 9 bit 9 10 bit 10 11 bit 11 12 bit 12 13 bit 13 14 bit 14 15 bit 15
A Format	*m* A *n*	Treats the m words as ASCII data and stores the specified character(s) in the string variable. *m* : the number of words *n* : the specified byte(s) ***n*** **Byte(s) read** 1 byte 1 (bits 0-7) 2 byte 2 (bits 8-15) 3 bytes 1-2 (bits 0-15)
S Format	S *m* X *n*	Treats the m words as the kind of data indicated by X (I, H, O, or B) and stores each word's data in the nth array variable. *m* : the number of words X : Data type (I, H, O or B as described above) *n* : the number of elements (as described above)

With the A format, many words of data from the PC can be stored in a single ASCII Unit variable, although a maximum of 127 words can be transferred at one time.

Except for the A and S formats, and where a character variable is used as the variable, there must be a one-to-one correspondence between each word and variable. With the A format, one format must be used with each variable. With the S format, one format must be used with the variables in each array.

Except for the A and S formats, and where a character variable is used as the variable, more than one variable can be associated with a single format.

If the "m" setting is omitted, the default setting is 1 word. The setting range for "m" is 1 to 255. For numerals it is 1 to 255, for characters it is 1 to 64.

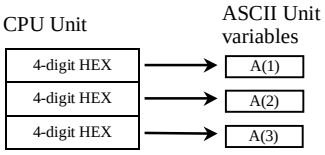
String variables can be used to specify the format, but be sure to use uppercase characters when specifying the format.

<variable> is any valid variable name. It must be of the same type as specified in the format string and have as many elements as the number of words specified in the PC READ statement.

As an option, the PC READ statement can return a <result>. This <result> indicates whether the PC READ statement was successful or not. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC READ could not be carried out, perhaps because another data transfer was being performed.

When the ASCII Unit receives a PC READ request from the CPU Unit (through bit 1 of IR n), the data from the data area location specified in IR n+3 and IR n+4 is read and stored in the specified variables with the indicated pc format. The data is transferred at the next I/O refreshing cycle, and the ASCII Unit will pause execution of the BASIC program until the data transfer has been completed.

Refer to *6.3 Details of the Data Exchange Methods* for more details on how to use the PC READ statement.

	Note:	1. If the amount of PC output data exceeds the size of variable, the excess output data will be ignored. 2. If the size of variable exceeds the amount of PC output data, the read operation will not be executed to the variable where the shortfall occurred. 3. A “FORMAT ERROR” will occur if the number of elements does not match the PC format setting. 4. Before using a character-string variable for the variable setting, be sure to read *Appendix A Operating Precautions* and *Appendix C PC Format* thoroughly, as the character-string variable operate differently from those in the previous C200H-ASC02.
Examples:	 The starting address and number of words in the CPU Unit are specified in the following words: IR n+3 and IR n+4.
See also:	PC QREAD, PC WRITE, WAIT

pc read@

Syntax:

[<result> =] PC READ "<data area>, <address>, <number of words>, <pc format> {, <pc format>}" **;** <variable> {**,** <variable>}

Description:

Statement. Allows the ASCII Unit to independently read data from the specified PC data area. The data area, starting address, and number of words are all specified at the ASCII Unit.

Remarks:

<data area> specifies which PC data area is the source for the transfer. Any of the following specifications for the data area are valid.

Other words in the CS1-series CPU Units cannot be addressed. Refer to *6-6 Restrictions for CS1-series PCs* for details.

C200H/HS/HE/HG/HX

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	C200H	DM 0000 to DM 1999	@D	0000 to 1999
	C200HS/HE/HG/HX	DM 0000 to DM 6655		0000 to 6655
IR/SR Area	C200H	000 to 255	@R	0000 to 0255
	C200HS	000 to 299		0000 to 0299
	C200HE/HG/HX	000 to 511		0000 to 0511
LR Area	C200H/HS/HE/HG/HX	LR 00 to LR 63	@L	0000 to 0063
HR Area		HR 00 to HR 99	@H	0000 to 0099
AR Area		AR 00 to AR 27	@A	0000 to 0027
EM Area, bank 0	C200HG/HX	EM 0000 to EM 6143, bank 0	@E	0000 to 6143
TC Area	C200H/HS/HE/HG/HX	T/C 000 to T/C 511	@G	0000 to 0511

CS1

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	CS1	D00000 to D06655	@D	0000 to 6655
I/O Area (CIO Area)		0000 to 0511 **Note:** Impossible to designate 0281	@R	0000 to 0511 **Note:** Impossible to designate 0281
Data Link Area (CIO Area)		1000 to 1063	@L	0000 to 0063
Holding Bit Area		H000 to H099	@H	0000 to 0099
		H100 to H127 **Note:** Impossible to designate H101	@A	0000 to 0027 **Note:** Impossible to designate 0001
EM Area, bank 0		E0_00000 to E0_06143	@E	0000 to 6143
Timer Area		T0000 to T0511	@G	0000 to 0511

Note: The EM Area is supported by the C200HE/HG/HX and CS1-series PCs only.

<address> specifies the starting address of data transfer within the specified data area. The starting address must be consistent with the address range of the specified data area. If the address given lies outside the allowable range then a “FORMAT ERROR” (code B067) will result.

<number of words> specifies the number of data words to be read from the starting address given. If any of the data to be read exceed the allowable range then a “FORMAT ERROR” (code B067) will result. The valid setting range is 1 to 255 words.

<pc format> indicates the data format of the data to be read. Refer to the explanation of <pc format> settings in the PC READ statement for details on the various data formats that can be specified.

<variable> is any valid variable name. It must be of the same type as specified in the format string and have as many elements as the number of words specified in the PC READ@ statement.

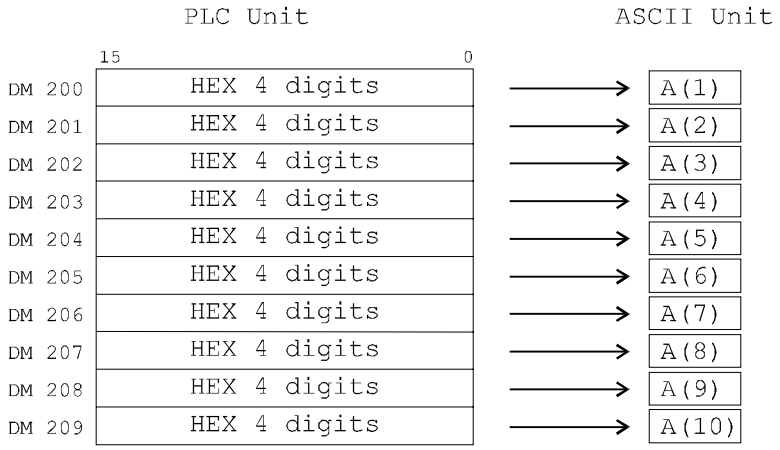
The PC READ@ statement can return a <result>. This <result> indicates whether the PC READ@ statement was successful or not. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC READ@ could not be carried out, perhaps because another data transfer was being performed.

With the PC READ@ statement, the ASCII Unit operates without direction from the CPU Unit. The ASCII Unit specifies a location in one of the PC’s data area, reads the data, and stores it in the specified variables according to the specified pc format. The data is transferred at the next I/O refreshing cycle after execution of the PC READ@ statement; the ASCII Unit will pause execution of the BASIC program until the data transfer has been completed.

Refer to *6.3 Details of the Data Exchange Methods* for more details on how to use the PC READ@ statement.

Before using a character-string variable for the variable setting, be sure to read *Appendix A Operating Precautions* and *Appendix C PC Format* thoroughly, as the character-string variable operate differently from those in the previous C200H-ASC02.

Examples:



PC READ “@D, 200, 10, S10H4”; A(1)

See also: PC QREAD, PC WRITE, WAIT

pc write

Syntax:	[<result> =] PC WRITE "<pc format> {, <pc format>}"**;** <numeric expression> {**,** <numeric expression>}
Description:	Statement. When a PC WRITE request is received from the CPU Unit, the variable data is written to the specified PC data area in the specified PC format.
Remarks:	The <pc format> indicates the data format of the specified data. The following table shows the various data formats that can be specified.

Format	Syntax	Description
I Format	*m* I *n*	Takes the first n digits of data from each of the m variables, converts the data to BCD, and stores it in the first n digits of the each of the m data area words. *m* : number of words *n* : number of digits written to each word (1 to 4) **n** / **Digits written** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
H Format	*m* H *n*	Takes the first n digits of data from each of the m variables, converts the data to hexadecimal, and stores it in the first n digits of the each of the m data area words. *m* : number of words *n* : number of digits written to each word (1 to 4) **n** / **Digits written** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
O Format	*m* O *n*	Takes the first n digits of data from each of the m variables, converts the data to octal, and stores it in the first n digits of the each of the m data area words. *m* : number of words *n* : number of digits written to each word (1 to 4) **n** / **Digits read** 1 / digit 1 (bits 0-3) 2 / digits 1-2 (bits 0-7) 3 / digits 1-3 (bits 0-11) 4 / digits 1-4 (bits 0-15)
B Format	*m* B *n*	Treats the data in each of the m variables as decimal, converts the data to binary, and stores only the nth bit's value (treating the rest as 0) in each of the m data area words. *m* : number of words *n* : bit specified in the word (0 to 15) **n** / **Specified bit** 0 / bit 0 1 / bit 1 2 / bit 2

		3 bit 3 4 bit 4 5 bit 5 6 bit 6 7 bit 7 8 bit 8 9 bit 9 10 bit 10 11 bit 11 12 bit 12 13 bit 13 14 bit 14 15 bit 15
A Format	*m* A *n*	Converts a string variable with 2×m characters to ASCII data (2 characters for each word of data) and stores the data in the m words. The value of n indicates whether the first byte, second byte, or both are stored. *m* : number of words *n* : specified byte(s) ***n*** **Byte(s)** 1 byte 1 (bits 0-7) 2 byte 2 (bits 8-15) 3 bytes 1-2 (bits 0-15)
S Format	S *m* X *n*	Converts the data in the n array variables to the kind of data indicated by X (I, H, O, or B) and stores the data in the m data area words. *m* : the number of words X : Data type (I, H, O or B as described above) *n* : the number of elements (as described above)

With the A format, many words of data from the PC can be stored in a single ASCII Unit variable, although a maximum of 127 words can be transferred at one time.

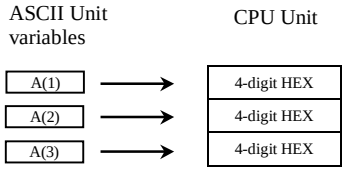
Except for the A and S formats, and where a character variable is used as the variable, there must be a one-to-one correspondence between each word and variable. With the A format, one format must be used with each variable. With the S format, one format must be used with the variables in each array.

Except for the A and S formats, and where a character variable is used as the variable, more than one variable can be associated with a single format.

If the “m” setting is omitted, the default setting is 1 word. The setting range for “m” is 1 to 255. For numerals it is 1 to 255, for characters it is 1 to 64.

String variables can be used to specify the format, but be sure to use uppercase characters when specifying the format.

<numeric expression> is any valid variable name. It must be of the same type as specified in the format string and have as many elements as the number of words specified in the PC WRITE statement. If the value of the numeric expression is not an integer, the non-integer portion will be truncated. Single-precision or double-precision floating-point values will be converted to integer values.

	The PC WRITE statement can return a <result>. This <result> indicates whether the PC WRITE statement was successful or not. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC WRITE could not be carried out, perhaps because another data transfer was being performed. When the ASCII Unit receives a PC WRITE request from the CPU Unit (through bit 2 of IR n), data in the specified variables is written to the data area location specified in IR n+3 and IR n+4. The data is transferred at the next I/O refreshing cycle after the PC WRITE request from the CPU Unit; the ASCII Unit will pause execution of the BASIC program until the data transfer has been completed. Before using a character-string variable for the variable setting, be sure to read *Appendix A Operating Precautions* and *Appendix C PC Format* thoroughly, as the character-string variable operate differently from those in the previous C200H-ASC02.
Examples:	 The starting address and number of words in the CPU Units are specified in the following words: IR n+3 and IR n+4. PC WRITE "S3HS"; A(1)
See also:	PC READ, PC QWRITE, WAIT

pc write@

Syntax:

[<result> =] PC WRITE "<data area>, <address>, <number of words>, <pc format> {, <pc format>}" **;** <numeric expression> {**,** <numeric expression>}

Description:

Statement. Allows the ASCII Unit to independently write data to the specified PC data area. The data area, starting address, and number of words are all specified at the ASCII Unit.

Remarks:

<data area> specifies which PC data area is the destination for the transfer. Any of the following specifications for the data area are valid.

Other words in the CS1-series CPU Units cannot be addressed. Refer to *6-6 Restrictions for CS1-series PCs* for details.

C200H/HS/HE/HG/HX

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	C200H	DM 0000 to DM 1999	@D	0000 to 1999
	C200HS/HE/HG/HX	DM 0000 to DM 6143		0000 to 6143
IR/SR Area	C200H	000 to 255	@R	0000 to 0255
	C200HS	000 to 299		0000 to 0299
	C200HE/HG/HX	000 to 511		0000 to 0511
LR Area	C200H/HS/HE/HG/HX	LR 00 to LR 63	@L	0000 to 0063
HR Area		HR 00 to HR 99	@H	0000 to 0099
AR Area		AR 00 to AR 27	@A	0000 to 0027
EM Area, bank 0	C200HG/HX	EM 0000 to EM 6143, bank 0	@E	0000 to 6143
TC Area	C200H/HS/HE/HG/HX	T/C 000 to T/C 511	@G	0000 to 0511

CS1

Area	**PC**	**Address**	**Designation method**	
			Code	**First word address**
DM Area	CS1	D00000 to D06655	@D	0000 to 6655
I/O Area (CIO Area)		0000 to 0511 **Note:** Impossible to designate 0281	@R	0000 to 0511 **Note:** Impossible to designate 0281
Data Link Area (CIO Area)		1000 to 1063	@L	0000 to 0063
Holding Bit Area		H000 to H099	@H	0000 to 0099
		H100 to H127 **Note:** Impossible to designate H101	@A	0000 to 0027 **Note:** Impossible to designate 0001
EM Area, bank 0		E0_00000 to E0_06143	@E	0000 to 6143
Timer Area		T0000 to T0511	@G	0000 to 0511

Note: The EM Area is supported by the C200HE/HG/HX and CS1-series PCs only.

<address> specifies the starting address of data transfer within the specified data area. The starting address must be consistent with the address range of the specified data area. If the address given lies outside the allowable range then a “FORMAT ERROR” will result.

<number of words> specifies the number of data words to be written from the starting address given. If the amount of the data to be written exceeds the allowable range then a “FORMAT ERROR” (code B067) will result. For numerals it is 1 to 255, for characters it is 1 to 64.

<pc format> indicates the data format of the data to be read. Refer to the explanation of <pc format> settings in the PC WRITE statement for details on the various data formats that can be specified.

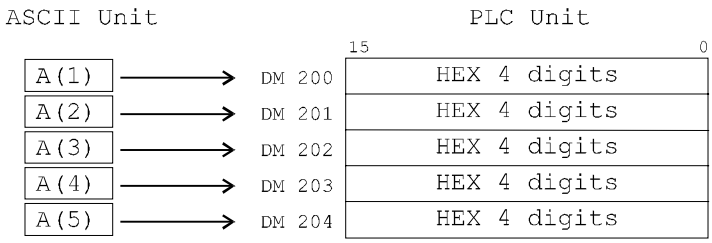
<numeric expression> is any valid variable name. It must be of the same type as specified in the format string and have as many elements as the number of words specified in the PC WRITE@ statement.

PC WRITE “@D, 200, 5, 1H4, 1H4, 1H4, 1H4, 1H4”; A(1), A(2), A(3), A(4), A(5)

The PC WRITE@ statement can return a <result>. This <result> indicates whether the PC WRITE@ statement was successful or not. If the <result> is 0 then the command was successfully carried out and the data is written successfully. If the <result> is -1 then the PC WRITE@ could not be carried out, perhaps because another data transfer was being performed.

With the PC WRITE@ statement, the ASCII Unit operates without direction from the CPU Unit. The ASCII Unit specifies a location in one of the PC’s data area and writes the data from the specified variables according to the specified pc format. The data is transferred at the next I/O refreshing cycle after execution of the PC WRITE@ statement; the ASCII Unit will pause execution of the BASIC program until the data transfer has been completed.

Refer to *6.3 Details of the Data Exchange Methods* for more details on how to use the PC WRITE@ statement.
Before using a character-string variable for the variable setting, be sure to read *Appendix A Operating Precautions* and *Appendix C PC Format* thoroughly, as the character-string variable operate differently from those in the previous C200H-ASC02.



Examples:	PC QREAD, PC WRITE, WAIT
See also:	

peek

Syntax:	PEEK (<address expression>)
Description:	Function Returns the contents (converted from ASCII code to decimal) of a specified address.
Remarks:	<address expression> is address to be PEEKed. Valid range: *[&H0E000…&H3FFFF]*. The result is an integer value representing the character code of the byte at the specified address. The returned integer will be in the range *[0…255]*. **Note:** 1. Be sure to address the correct area. The user memory contains areas other than the variable area. The other areas are used for other purposes, such as program execution.

Examples:	`> 10 A = PEEK(&H30000)`
	`> 20 PRINT A`
	`> RUN`
	`32`

See also:	POKE

pgen

Syntax:	PGEN <program expression>
Description:	Command. Selects which of the four BASIC program areas is the current area.
Remarks:	<program expression> is an integer in the range: *[1…4]*. If the program expression is out of the allowable range an "ILLEGAL FUNCTION CALL" error (code B005) will result. The name and the number of bytes currently used by the selected program area will be displayed. All subsequent commands entered at the terminal will operate on the currently active program area.

Examples:	`> PGEN 1`
	`> LIST`
	`10 REM THIS IS THE PROGRAM IN MEMORY AREA #1`
	`20 PRINT "HELLO WORLD"`
	`> PGEN 2`
	`> LIST`
	`10 REM THIS IS THE PROGRAM IN MEMORY AREA #2`
	`20 PRINT "HELLO AGAIN WORLD"`
	`>`

See also:	PNAME, PINF, PMEM

pinf

Syntax:	PINF [(ALL | <program expression>)]
Description:	Command. Displays information about the specified BASIC program area.
Remarks:	<program expression> is an integer in the range: *[1…4]*. If the program expression is outside the allowable range an "ILLEGAL FUNCTION CALL" error (code B005) will result. Alternatively information about all BASIC program areas can be displayed by specifying PINF ALL. If no argument is supplied to the PINF command information about the current program area will be provided. The name of the specified program areas, their read/write status, number of bytes used, number of bytes of intermediate code used, and available free space will be displayed as shown below. P1 : NAME [R/W] NUMBER OF BYTES USED P2 : NAME [R/W] NUMBER OF BYTES USED P3 : NAME [R/W] NUMBER OF BYTES USED P4 : NAME [R/W] NUMBER OF BYTES USED LIB : NUMBER OF BYTES USED I_CODE: NUMBER OF BYTES USED AVAILABLE FREE SPACE P1 through P4 are program numbers 1 through 4, a write-protect status of "R" is read-only and "R/W" is read/write, "LIB" is system-reserved, and "I_CODE" is the intermediate code. The intermediate code of the program that will be executed is created when the ASCII Unit is started (RUN). In some cases there is not enough memory available to create the intermediate code, so it is advisable to execute the RUN command and check the size of the I_CODE and the available free space. The memory available after creation of the intermediate code can be used for applications such as the variable area.

Examples:	`> PINF ALL`
	`P1 : TEST [R/W] 6523 BYTES`
	`P2 : [R/W] 0 BYTES`
	`P3 : [R] 2876 BYTES`
	`P4 : MY_PROG [R] 23231 BYTES`
	`LIB : 0 BYTES`
	`I_CODE : 13046 BYTES`
	`157890 BYTES FREE`
	`>`

See also:	PGEN, PMEM, PNAME

pmem

Syntax:	PMEM (ON | OFF)
Description:	Command. Turns write protection of the current BASIC program area ON or OFF.
Remarks:	The PMEM statement write protects the current BASIC program area. If protection is set using PMEM ON then it is not possible to change or delete the BASIC program in the current BASIC program area. To remove write protection of the BASIC program area, execute PMEM OFF. The disabling of RENUM cannot be set from PMEM.

Examples:	`> 10 REM THIS PROGRAM WILL BE PROTECTED`
	`> PMEM ON`
	`> DEL 10`
	`PROTECTED PROGRAM`
	`> PMEM OFF`
	`> DEL 10`
	`> LIST`
	`>`

See also:	DEL, EDIT, NEW, PGEN, PINF, PNAME.

pname

Syntax:	PNAME <string expression>
Description:	Command. Assigns a name to the current BASIC program area.
Remarks:	<string expression> is any valid BASIC string with a maximum length of 8 characters. If the name is longer than 8 characters it will be truncated. Upper and lower case characters are distinguished in the names. If a program area has been assigned a name using the PNAME statement then it is protected against the effects of the NEW command. In other words it cannot be overwritten. However it may still be edited using the EDIT and the DEL commands. A name assigned to a program area may be removed by assigning an empty string as the name in the PNAME statement. E.g. PNAME “ ”

Examples:	`> 10 REM THIS PROGRAM HAS A NAME`
	`> PNAME “PROG1”`
	`> PINF`
	`P1  : PROG1 [R/W]  30  BYTES`
	`>`

See also:	DEL, EDIT, NEW, PGEN, PINF, LOAD.

poke

Syntax:	POKE <address expression> , <numerical expression>
Description:	Statement. Writes one byte to the specified address.
Remarks:	<address expression> is the memory location that will be POKEd and is an expression returning an integer in the range : *[&H0E000…&H3FFFFF]*. <numerical expression> is the byte of data that will be written to the specified address in the range : *[0…255]*. The user should not POKE data to memory areas reserved for system use. Doing so may cause the system to run out of control. Do not write data to any addresses other than variable addresses using the POKE command.
Examples:	`POKE &H25000, &H65 (&H25000 is a variable address)`
See also:	PEEK

print

Syntax:	PRINT [# <port expression>][USING <print format>;]<expression> {[(,|;|<space>)] <expression>}
Description:	Statement. Outputs data or text to a communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. If the <port expression> is omitted, the terminal port (#1 in the ASC11/21, #3 in the ASC31) is used as the default. <Expression> is any valid numerical or character expression. If the expression does not correspond to the <print format> string then a "TYPE MISMATCH" error (code B013) will occur. If the <expressions> are separated by semicolons or spaces, and no formatting is used, they are output one immediately after the other on the output. If they are separated by commas they are output with a tab character separating them on the output stream. If the terminating comma, or semicolon, is omitted a carriage return is appended at the end of the output. All numerical expressions are output with a space character either side of the expression. The space before the numerical expression can be used for an optional minus sign. If no expressions are specified this statement outputs a carriage return on the output. An optional format string can be used by applying the USING statement. The <print format> string is a string expression containing the following control characters. If multiple expressions are set, make sure to include a space between the expression and the corresponding format to be used. Otherwise the output may not follow the correct format.

Format type	Code	Description
String	!	Prints only the first character of a string.
	&&	Prints the first n characters, where n is the number of blanks enclosed between & plus 2.
	@	Prints the corresponding character string.
Numerical	#	Indicates a digit position.
	.	Inserts a decimal point at any desired place
	+	Specifies the position of the sign of the numeric value.
	-	If specified at the end of the numeric value it specifies that the sign for negative numbers is suffixed to numeric value.
	**	Fills the left-most unused positions of the numeric value with *. Also if - is specified at the end of a number then * will fill the sign position if the value is positive.

\\	Prefixes \ at the beginning of the numeric value. \ allocates space for one digit position.
**\	A combination of ** and \\.
,	Separates the integer portion of a numeric value by a comma every three digits from the right of the integer part.
^^^^	When specified at the end of a numeric field it specifies that exponential form should be used. The exponential format is (E+nn,D+nn).
^^^^^	When specified at the end of a numeric field it specifies that specification format should be used. The specification format is (E+nnn,D+nnn).
_	Allows the printing of the above special characters in the format string. If the special character is preceded by the '_' it is printed as normal.

The number of digits including codes is indicated by the number of "#" symbols. If the number of digits of data is less than the number indicated by the "#" symbols, the number will displayed with more significant digits. If the number of digits of data is greater than the number indicated by the "#" symbols, the number will displayed after a "%" symbol.

Note:

1. The port must be opened for output.
2. Port #3 corresponds to the terminal port of the ASC31.
3. If Port #3 is specified on the ASC11 or ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
4. If the specified number of digits to be output is greater that that specified in the numerical format a "%" will be output before the numerical value.

It is possible to use DMA transfer for printing through port #1 of the ASC31 and port #2 of the ASC11, ASC21 and ASC31. DM m+2 bits 08-15 specify whether or not DMA is used for port #1 and DM m+3 bits 08-15 specify whether or not DMA is used for port #2. A maximum of 255 bytes of data can be sent for each DMA data transmission.

Example 1:

```
> 10 PRINT "HELLO WORLD"
> RUN
HELLO WORLD
```

Example 2:

```
> 10 X=1234.56
> 20 PRINT USING "####.###";X
> 30 PRINT USING "+####.###";X
> 40 PRINT USING "X=#####.##";X
> 50 PRINT USING "+###.#";1234.5
> 60 END
> RUN
1234.560
+1234.560
X= 1234.56
%+1234.5
```

See also: LPRINT, WAIT

pword

Syntax:	PWORD <string expression>
Description:	Statement. Assigns a password to the BASIC program area.
Remarks:	<string expression> is any valid BASIC string with a maximum length of 32 characters. If the name is longer than 32 characters it will be truncated. Passwords are case-sensitive. The PWORD statement must be the first line of the BASIC program. The password will be invalid if the PWORD statement is not at the beginning of the program. Set the password after inputting the whole program. If the password is set at the beginning of the program, the password will be required at the beginning. If a BASIC program is protected with a PWORD statement then the password will be requested for the following commands. If the password is incorrect then a "PASSWORD INCORRECT" error (code 0069) will occur. If the password is correct then the command will be executed as normal. Even if a password is set using the PWORD statement, the program will not be protected from the NEW command.

Examples:	`> 1 PWORD "ABRACADABRA"`
	`> 20 PRINT "PROTECTED BY A PASSWORD"`
	`> RUN`
	`> PROTECTED BY A PASSWORD`
	`> LIST`
	`PASSWORD : **********`
	`1 PWORD "ABRACADABRA"`
	`20 PRINT "PROTECTED BY A PASSWORD"`
	`>`

See also:	PNAME, PMEM

random

Syntax:	RANDOM [<seed>]
Description:	Statement. Re-seeds the random number generator.
Remarks:	<seed> is a numerical expression that is used as the random number generator seed. The valid range is: *[-32768…32767]*. If the RANDOM statement is used without a seed then the user will be prompted for it. If the random number generator is not re-seeded then the RND statement will begin the same pseudo-random sequence each time the power is turned OFF. For better random behavior the random number generator should be re-seeded using the RANDOM statement each time a program is executed.

Examples:	`> 10 RANDOM 654`
	`> 20 PRINT RND`
	`> RUN`
	`.234831`
	`> PRINT RND`
	`.969226`
	`> RUN`
	`.234831`

See also:	RND

read

Syntax:	READ <variable> {, <variable>}
Description:	Statement. Reads values from DATA statements and assigns them to the specified variables.
Remarks:	<variable> is any valid BASIC numeric or character variable name. READ statements must always be used in conjunction with DATA statements. If no corresponding DATA statements exist, or if all of the data has been exhausted with READ statements then an "OUT OF DATA" error (code B004) will occur. The variables specified in the READ statements can be of any type, however the type must match the type of the value in the corresponding DATA statement. If the type of the variable in the READ statement does not match the type of the value in the DATA statement then a "SYNTAX ERROR" will occur. The READ statement can read a numerical constant as either a string or a number depending on the type of the variable used in the READ statement. However, a string constant must always be read with a numerical string variable. When there are two or more DATA statements, they are treated as a single list of data items and the constants in the DATA statements are read in the order that they appear in the program. When the RESTORE statement is executed, data will be read again starting from the first DATA statement. If all data has been read and the RESTORE statement has not been executed then the next READ will result in a "OUT OF DATA" error (code B004).

Examples:	`> 10 DATA 10, "HELLO", 1.6, "WORLD"`
	`> 20 READ A, B$, C, D$`
	`> 30 PRINT B$+" "+D$`
	`> 40 PRINT A+C`
	`> RUN`
	`HELLO WORLD`
	`11.6`

See also:	DATA, RESTORE

rem

Syntax:	REM [<comment string>]
Description:	Statement. Inserts non-executable comments in a program.
Remarks:	<comment string> is a freely definable string that need not be enclosed in quotes. It can be as long as the maximum line length. REM statements should be used to give titles to programs and to make useful comments to make the program easier to understand. All characters on the same line but after the REM statement are ignored. It is not necessary to insert a space between the REM keyword and the comment string. Comments can be added to the end of a line by using a colon to specify the start of a new statement. Alternatively a single quote character can be used instead for the same effect. The REM keyword cannot be used in DATA statements as it will be assumed to be a legal data string.
Examples:	`> 10 REM TEST PROGRAM`
See also:	

renum

Syntax:	RENUM [<new line number>][,[<old line number>][, <increment>]]
Description:	Command. Renumbers program lines.
Remarks:	<new line number> is the first line number to be used in the new sequence. It can be any valid line number in the range : *[1…65535]*. If it is omitted then it has a default value of 10. <old line number> is the first line number in the current program where renumbering should begin from. It can be any valid line number in the range : *[1…65535]*. If it is omitted then it defaults to the first line of the program. <increment> is the increment to be used in the new sequence. It can be any valid integer in the range : *[1…65535]*. If it is omitted then it has a default value of 10. The RENUM statement also changes the line number references used in the program in GOTO, GOSUB, IF...THEN...ELSE and RESTORE statements. If the RENUM statement would cause an invalid program number then it is not executed. An "ILLEGAL FUNCTION CALL" error (code B005) will result.

Examples:	`> 1 REM FIRST LINE`
	`> 2 REM SECOND LINE`
	`> 3 GOTO 3`
	`> RENUM 100, 1`
	`> LIST`
	`100 REM FIRST LINE`
	`110 REM SECOND LINE`
	`120 GOTO 120`
	`>`

See also:	

restore

Syntax:	RESTORE [(<line number> | <label>)]
Description:	Statement. Resets the next data item to first item in the DATA statement specified by the line number.
Remarks:	<line number> is any valid line number in the BASIC program containing a DATA statement. The valid range is : *[1... 65535]*. <label> is a BASIC program label. It references a line number somewhere in the BASIC program. If the <line number> and <label> are omitted then the data pointer will be restored to the first DATA statement in the BASIC program. If the specified <line number> does not exist in the BASIC program, an "UNDEFINED LINE NUMBER" error (code B008) will occur.

Examples:	`> 10 DATA 10, "HELLO", 1.6, "WORLD"`
	`> 20 READ A, B$, C, D$`
	`> 30 PRINT B$+" "+D$`
	`> 40 PRINT A+C`
	`> 50 RESTORE 10`
	`> 60 READ A, B$, C, D$`
	`> 70 PRINT A+C`
	`> 80 PRINT B$+" "+D$`
	`> RUN`
	`HELLO WORLD`
	`11.6`
	`11.6`
	`HELLO WORLD`

See also:	DATA, READ

resume

Syntax:	RESUME [(<line number> | NEXT)]
Description:	Statement. Resume program execution after an error handling operation.
Remarks:	<line number> is any valid line number in the BASIC program from which execution will resume. The valid range is [0…65535]. If <line number> is omitted or set to 0, program execution will be continued at the statement that caused the error. If NEXT is specified execution will resume at the line after the line that caused the error.

Examples:	`> 10 B = 6`
	`> 20 ON ERROR GOTO 80`
	`> 30 FOR A = 1 TO 10`
	`> 40 PRINT A, B, A/B`
	`> 50 B = B-1`
	`> 60 NEXT A`
	`> 70 END`
	`> 80 PRINT "ERROR HAS OCCURRED"`
	`> 90 B = 5`
	`> 100 RESUME`
	`1 6 .166667`
	`2 5 .4`
	`3 4 .75`
	`4 3 1.33333`
	`5 2 2.5`
	`6 1 6`
	`ERROR HAS OCCURRED`
	`7 5 1.4`
	`8 4 2`
	`9 3 3`
	`10 2 5`
	`>`

See also:	ERROR, ON ERROR

right$

Syntax:	RIGHT$ (<string expression>, < numerical expression>)
Description:	Function Returns the specified number of characters starting from the rightmost position in a string expression.
Remarks:	<string expression> is the string to be searched. <numerical expression> is the number of characters to be returned. The valid range is : *[1…255]*. If the number of characters to be returned is 0 then a null string is returned. If the number of characters to be returned is greater than the number of characters in the search string then the entire search string is returned.

Examples:	`> 10 A$ = RIGHT$("HELLO WORLD", 5)`
	`> 20 PRINT A$`
	`> RUN`
	`WORLD`

See also:	LEFT$, MID$

rnd

Syntax:	RND [(<numerical expression>)]
Description:	Function. Returns a random number between 0 and 1.
Remarks:	<numerical expression> has a valid range: *[-2147483648…2147483647]* If the <numerical expression> is negative the first number in the random number sequence is returned. If the <numerical expression> is omitted or positive then the next random number in the sequence is returned. If the <numerical expression> is zero then the last random number in the sequence is repeated. **Note:** 1. The random number sequence may be changed by using the RANDOM function.

Examples:	`> 10 RANDOM 654`
	`> 20 PRINT RND`
	`> RUN`
	`.234831`
	`> PRINT RND`
	`.969226`
	`> RUN`
	`.234831`

See also:	RANDOM

romload

Syntax:	ROMLOAD
Description:	Command. Restores BASIC programs from FlashROM to program areas in RAM.
Remarks:	All four BASIC program areas are restored from FlashROM. Refer to *7-1 Programming Procedure* for precautions on restoring data from FlashROM. If the ROMLOAD command is used while a program is executed, an “ILLEGAL FUNCTION CALL” error will occur. If ROMSAVE has not been performed and the user attempts to execute ROMLOAD from flash ROM, a “DATA TRANSFER CHECK SUM ERROR” will occur.

Examples:	`> 10 REM THIS PROGRAM WILL BE SAVED`
	`> ROMSAVE`
	`> 10 REM THIS WILL BE OVERWRITTEN`
	`> LIST`
	`10 REM THIS WILL BE OVERWRITTEN`
	`> ROMLOAD`
	`> LIST`
	`10 REM THIS PROGRAM WILL BE SAVED`
	`>`

See also:	ROMSAVE, ROMVERIFY

romsave	
Syntax:	ROMSAVE
Description:	Command. Saves BASIC programs in RAM to FlashROM.
Remarks:	All four BASIC program areas are saved from RAM to FlashROM. Refer to *7-1 Programming Procedures* for precautions on saving data to FlashROM. The ASCII Busy Flag will turn ON while the ROMSAVE command is executing (i.e., while data is being written to FlashROM). During this time, data will not be received at ports, and interface processing with the PC (I/O refreshing) will not be performed. If the ROMSAVE command is used while a program is executed, an "ILLEGAL FUNCTION CALL" error will occur.

Examples:	`> 10 REM THIS PROGRAM WILL BE SAVED`
	`> ROMSAVE`
	`> 10 REM THIS WILL BE OVERWRITTEN`
	`> LIST`
	`10 REM THIS WILL BE OVERWRITTEN`
	`> ROMLOAD`
	`> LIST`
	`10 REM THIS PROGRAM WILL BE SAVED`
	`>`

See also:	ROMLOAD, ROMVERIFY

romverify	
Syntax:	ROMVERIFY
Description:	Command. Checks the consistency of the program stored in FlashROM with those currently stored in RAM.
Remarks:	All four BASIC program in RAM are compared with those stored in FlashROM. If the contents do not match a "VERIFY ERROR" (code 0066) will occur. If the contents are the same then there will be no output and the ASCII Unit will return to the command prompt. If the ROMVERIFY command is used while a program is executed, an "ILLEGAL FUNCTION CALL" error will occur.

Examples:	`> 10 REM THIS PROGRAM WILL BE SAVED`
	`> ROMSAVE`
	`> VERIFY`
	`> 10 REM THIS WILL BE OVERWRITTEN`
	`> VERIFY`
	`VERIFY ERROR`
	`> ROMLOAD`
	`> VERIFY`
	`> LIST`
	`10 REM THIS PROGRAM WILL BE SAVED`

See also:	ROMLAOD, ROMSAVE

rts

Syntax:	RTS #<port expression> (SET | RESET)
Description:	Statement. Sets or resets the RTS line of the specified communications port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1…3]*. If accompanied with the SET command the RTS line is made active, if the RESET command is used then the RTS becomes inactive. The RTS statement can only be used with open ports. **Note:** 1. Port #3 corresponds to the terminal port of the ASC31. 2. If Port #3 is specified on the ASC11 or if port #2 or port #3 is specified on ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`RTS #2 SET`
See also:	CTS, DSR, DTR

run

Syntax:	RUN [<start line number>][TO <end line number>]
Description:	Command. Starts program execution.
Remarks:	<start line number> is any valid line number in the range : *[1…65535]*. If it is omitted then it defaults to the first line in the BASIC program. If the <start line number> is used then program execution will begin at the line number specified. If the RUN command is used with the TO modifier then a breakpoint will be set at the specified <end line number> and execution will terminate just before this point. If a line number that does not exist is specified in a program, an "UNDEFINED LINE NUMBER" error will occur. The RUN command clears all variables and closes all open ports before beginning execution of the program. Program execution can be aborted by pressing CTRL+X, or by setting the START/STOP toggle switch to STOP. If the switch is already at STOP when the RUN command is given the program will not run, but the "CANNOT RUN WITH SWITCH IN STOP POSITION" message is displayed. Program execution can be halted from within the BASIC program using the END or the STOP statements. If the RUN command is executed immediately after programming is completed, a message saying that the program is being compiled will be displayed. The time required from when the RUN command is executed to when the program actually starts to be executed may be delayed depending on the time taken to compile the BASIC program and to perform garbage collection. If a <start line number> or <end line number> that does not exist is specified in a program, an "UNDEFINED LINE NUMBER" error will occur. The RUN compile time for a 53-Kbyte (approx. 1,300 column) program is approximately 10 seconds. The time however, may differ depending on the commands that are used in the program. **Note:** If RUN is executed after making more than one change to the program, several minutes may be required to compile the program (due to complex memory status in the ASCII Unit).
Examples:	RUN
See also:	END, STOP

save

Syntax:	SAVE # <port expression> [,"TERM:[<communication definition string expression>]"]
Description:	Command. Writes the BASIC program in the current program area to a communication port.
Remarks:	<port expression> is an expression returning an integer in the range: *[1… 3]*. See the OPEN statement for a description of the <communication definition string expression>. The communications conditions specified by the SAVE command's <communication definition string expression> are valid while the SAVE command is being executed. The original conditions are valid again after execution of the SAVE command is completed. **Note:** 1. The procedure for reading a program stored in the ASCII Unit to a terminal depends on whether the terminal mode of the connected port is VT-100 or FIT-10. In either case, input from the terminal is disabled during the transfer. Refer to *7-1 Programming Procedure* for details on saving BASIC programs. 2. Port #3 corresponds to the terminal port of the ASC31. 3. If Port #3 is specified on the ASC11 or ASC21 an "ILLEGAL FUNCTION CALL" error (code B005) will result.
Examples:	`> SAVE #1, "TERM:9600,8,N,2,CS_OFF,RS_OFF,DS_OFF,XN_ON"`
See also:	LOAD

search

Syntax:	SEARCH (<integer array>, <numerical expression 1> [, <numerical expression 2>] [, <numerical expression 3>])
Description:	Function. Searches an specified array variable (one-dimensional array variables only) for the occurrence of a specified value and returns the index of the first occurrence of that value.
Remarks:	<integer array> is a one-dimensional array of integers to be searched. <numerical expression 1> is the expression to be searched for in the array. If this is not an integer it is first rounded. <numerical expression 2> this optional expression gives the starting index of the search in the array. If omitted the starting index is assumed to be the first element in the array. If OPTION BASE has been executed specifying that the first index is 1 and this expression is 0 then the search will start from the first element in the array, index 1. <numerical expression 3> this optional integer expression gives the search increment. For instance a value of 2 would suggest that every second element in the array is searched. If omitted a value of 1 is assumed. If an occurrence of the search string is not found then -1 is returned.

Examples:	`> 10 DIM ARRINT%(100)`
	`> 20 FOR I = 1 TO 100`
	`> 30 ARRINT%(I) = I MOD 10`
	`> 40 NEXT I`
	`> 50 A = SEARCH(ARRINT%, 7, 12, 1)`
	`> 60 PRINT "7 IS FIRST FOUND AT POSITION "; A`
	`> RUN`
	`7 IS FIRST FOUND AT POSITION 17`

See also:	

sgn

Syntax:	SGN(<numerical expression>)
Description:	Function. Returns the sign of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. SGN returns 1 if the numerical expression is positive and -1 if the numerical expression is negative. If the numerical expression is zero SGN returns 0.

Examples:	`> 10 INPUT A%`
	`> 20 IF SGN(A%) = 1 THEN PRINT "POSITIVE"`
	`> 30 IF SGN(A%) = -1 THEN PRINT "NEGATIVE"`
	`> 40 IF SGN(A%) = 0 THEN PRINT "ZERO"`
	`> 50 GOTO 10`
	`> RUN`
	`? 1 <CR>`
	`POSITIVE`
	`? -34 <CR>`
	`NEGATIVE`
	`? 0 <CR>`
	`ZERO`

See also:	

sin

Syntax:	SIN(<numerical expression>)
Description:	Function. Calculates the sine of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double-precision floating point. **Note:** 1. The argument should be specified in radians.

Examples:	`> 10 A = 0.4`
	`> 20 PRINT SIN(A)`
	`> RUN`
	`.389418`

See also:	COS, TAN

space$

Syntax:	SPACE$ (<numerical expression>)
Description:	Function. Returns a character string containing the specified number of spaces.
Remarks:	<numerical expression> may be any integer expression with a valid range: *[0…255]*. Non-integer expressions will be rounded. If the numerical expression is 0 then a null string is returned.

Examples:	`> 10 A$ = "HELLO" + SPACE$(5) + "WORLD"`
	`> 20 PRINT A$`
	`> RUN`
	`HELLO  WORLD`

See also:	SPC

spc

Syntax:	SPC (<numerical expression>)
Description:	Function. Outputs blanks in a PRINT or LPRINT statement
Remarks:	<numerical expression> may be any integer expression with a valid range: *[0…255]*. Non-integer expressions will be rounded. If the expression is negative, 0 will be assumed. This function is used to output blanks in a PRINT or LPRINT statement. If the number of blanks exceeds the terminal's line width, the output will wrap around to the next line. **Note:** 1. SPC must be used in conjunction with a PRINT or LPRINT statement. It cannot be used on its own. 2. If the numerical expression is greater than 255 then an "ILLEGAL FUNCTION CALL" error (code B005) will result.

Examples:	`> 10 PRINT "HELLO";"WORLD"`
	`> 20 PRINT "HELLO";SPC(5);"WORLD"`
	`> RUN`
	`HELLOWORLD`
	`HELLO  WORLD`

See also:	TAB

sqr

Syntax:	SQR (<numerical expression>)
Description:	Function. Calculates the square root of numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point expression. If <numerical expression> is negative then an "ILLEGAL FUNCTION CALL" error (code B005) will result. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double-precision floating point.

Examples:	`> 10 A = 4`
	`> 20 PRINT SQR(A)`
	`> RUN`
	`2`

See also:	

step	
Syntax:	STEP [<numerical expression>]
Description:	Command. Executes the specified number of lines of BASIC program.
Remarks:	<numerical expression> specifies the number of program lines to execute, it may be any valid integer expression in the range *[0…255]*. If it is omitted a value of 1 is assumed. The STEP command resumes execution at the line number after the last executed line. It executes the number of lines specified before terminating execution. Execution can be continued by issuing another STEP command or by using the CONT command. If STEP is executed when there is no information about the last executed line, for instance after an END, LOAD or EDIT command, then STEP will execute the first line of the program.

Examples:	`> 10 PRINT "STARTED"`
	`> 20 STOP`
	`> 30 PRINT "CONTINUING"`
	`> 40 PRINT "CONTINUING"`
	`> 50 PRINT "FINISHED"`
	`> RUN`
	`STARTED`
	`BREAK EXECUTED IN LINE 20`
	`> STEP 2`
	`CONTINUING`
	`CONTINUING`
	`BREAK IN LINE 40`
	`> CONT`
	`FINISHED`
	`>`

See also:	CONT, RUN

stop	
Syntax:	STOP
Description:	Statement. Terminates execution of the BASIC program and returns to command mode.
Remarks:	The execution of the BASIC program will terminate and the message "BREAK EXECUTED IN LINE <line number>" will be displayed on the terminal. Execution can be continued from the current line by using the CONT or the STEP commands. The BASIC variables will not be cleared and open ports will not be closed.
Examples:	`STOP`
See also:	CONT, RUN, STEP

str$	
Syntax:	STR$ (<numerical expression>)
Description:	Function. Converts a numerical expression into a character string representation.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point expression. If the expression is positive a space is prefixed to the character string representation. If the numerical expression is negative a minus sign (-) is prefixed to the character string representation. The VAL function is the opposite of the STR function; it converts a character string into its equivalent numerical expression.

Examples:	`> 10 FOR A = 1 TO 5`
	`> 20 A$ = STR$(A)`
	`> 30 PRINT A$`
	`> 40 NEXT A`
	`> RUN`
	`1`
	`2`
	`3`
	`4`
	`5`

See also:	VAL

string$	
Syntax:	STRING$ (<numerical expression 1>, (<numerical expression 2> | <string expression>))
Description:	Function. Outputs a specified number of a certain character.
Remarks:	<numerical expression 1> is the number of the specified character that should be in the output string. It may be any integer, single-precision floating point or double-precision floating point expression. Non-integer expressions will be rounded. Valid range: *[0…255]*. If this value is zero an empty string will be returned. The second argument of STRING$ gives the character that should be repeated the specified number of times in the output string. This can be specified as a character code or as a character string. If specified as a character code the valid range is: *[0…255]*. If specified as a character string the first character in the string will be assumed to be the desired character. Use the STRING$ function when you need to make a character string with a series of the same character.

Examples:	`> 10 FOR A = 1 TO 5`
	`> 20 A$ = STRING$(A, "*")`
	`> 30 PRINT A$`
	`> 40 NEXT A`
	`> RUN`
	`*`
	`**`
	`***`
	`****`
	`*****`

See also:	SPACE$

tab

Syntax:	TAB (<numerical expression>)
Description:	Function. Moves the cursor to a specified position on the terminal display in a PRINT or LPRINT statement
Remarks:	<numerical expression> may be any integer expression with the valid range: *[1…255]*. Non-integer expressions will be rounded. If the value of the numerical expression exceeds the number of characters per line, then the value of the numerical expression will be divided by the number of characters per line and the remainder will be used. If the current cursor position is already beyond the specified tab position then the cursor will move to the specified position on the next line.

Examples:	`> 10 PRINT "H";TAB(0);"W"`
	`> 20 PRINT "H";TAB(1);"W"`
	`> 30 PRINT "H";TAB(2);"W"`
	`> 40 PRINT "H";TAB(3);"W"`
	`> 50 PRINT "HELL";TAB(4);"O"`
	`> RUN`
	`H`
	`W`
	`HW`
	`H W`
	`H  W`
	`HELLO`

See also:	SPC

tan

Syntax:	TAN (<numerical expression>)
Description:	Function. Calculates the tangent of a numerical expression.
Remarks:	<numerical expression> may be any integer, single-precision floating point or double-precision floating point. The return type is single-precision floating point if the argument is of type integer or single-precision floating point. If the argument is of type double-precision floating point then the return type is also double-precision floating point. **Note:** 1. The argument should be specified in radians.

Examples:	`> 10 A = 0.4`
	`> 20 PRINT TAN(A)`
	`> RUN`
	`.422793`

See also:	SIN, COS

time$

Syntax:	TIME$ [= <time string expression>]
Description:	System variable. Returns the time of the internal clock or sets the time.
Remarks:	<time string expression> is an absolute time in the format “hh:mm:ss”. Where hh has a range *[00…23]*, mm has a range *[00…59]* and ss has a range *[00…59]*. If the <time string expression> is not a valid time then an “ILLEGAL FUNCTION CALL” error (code B005) will occur. If used on its own the TIME$ statement returns the current time of the system clock.
Examples:	`TIME$ = “12:10:05”`
See also:	DATE$, DAY

time$ on/off/stop

Syntax:	TIME$ (ON | OFF | STOP)
Description:	Statement. Enables, disables or stops the time$ interrupt defined by the ON TIME$ statement.
Remarks:	ON enables (unmasks) the time$ interrupt. When a time$ interrupt occurs, program execution will be branched to the specified interrupt subroutine for interrupt processing. The time$ interrupts are enabled until they are disabled with the TIME$ OFF statement. OFF disables the interrupt. All subsequent time$ interrupts are ignored. STOP masks and enables the interrupt. If a time$ interrupt is received, it is stored in memory but program execution is not branched to the interrupt subroutine. Program execution will be branched to the subroutine when the interrupt is unmasked with the TIME$ ON statement. The TIME$ ON/OFF/STOP statements can be executed only after the ON TIME$ statement has been executed.

Examples:	`> 10 ON TIME$ = "08:00:00" GOTO 100`
	`> 20 TIME$ ON`
	`> 30 GOTO 30`
	`> 100 PRINT "WAKE-UP!!!"`
	`> 110 RETURN`

See also:	ON TIME$

timer on/off/stop

Syntax:	TIMER (ON | OFF | STOP)
Description:	Statement. Enables, disables or stops the timer interrupt.
Remarks:	ON enables (unmasks) the timer interrupt. When a timer interrupt occurs, program execution will be branched to the specified interrupt subroutine for interrupt processing. The timer interrupts are enabled until they are disabled with the TIMER OFF statement. OFF disables the interrupt. All subsequent timer interrupts are ignored. STOP masks and enables the interrupt. If a timer interrupt is received, it is stored in memory but program execution is not branched to the interrupt subroutine. Program execution will be branched to the subroutine when the interrupt is unmasked with the TIMER ON statement. The TIMER ON/OFF/STOP statements can be executed only after the ON TIMER statement has been executed.

Examples:	`> 10 T = 0`
	`> 20 ON TIMER 30 GOSUB 100`
	`> 30 TIMER ON`
	`> 40 GOTO 40`
	`> 100 T = T+3`
	`> 110 PRINT T; "SECONDS HAVE ELAPSED SINCE THE START OF EXECUTION."`
	`> 120 RETURN`

See also:	ON TIMER

trace

Syntax:	TRACE
Description:	Command. Displays the contents of the trace buffer.
Remarks:	When the TRON M command is executed, the program's line numbers are stored in the trace buffer as the program is executed. The trace buffer can contain up to 255 line numbers. The trace buffer has a battery backup, so its contents are retained even if the power is turned OFF. When necessary, the TRACE command can be used to output the contents of the trace buffer to the terminal. The contents of the trace buffer will be cleared when the NEW or TROFF command is executed.

Examples:	`> TRON M`
	`> 10 FOR A = 1 TO 5`
	`> 20 PRINT A`
	`> 30 NEXT A`
	`> RUN`
	`1`
	`2`
	`3`
	`4`
	`5`
	`> TRACE`
	`[10] [20] [30] [10] [20] [30] [10] [20] [30] [10] [20] [30] [10] [20] [30]`
	`[10] [30]`

See also:	TROFF, TRON

troff

Syntax:	TROFF
Description:	Command. Turns OFF the line number trace
Remarks:	Stops the output of line numbers during the execution of a program. If the ASCII Unit is reset, powered OFF then ON, or if the NEW statement is executed then the line number trace will automatically be turned OFF. When the line number trace is turned OFF the trace buffer is reset.

Examples:	`> 10 TRON`
	`> 20 FOR A = 1 TO 5`
	`> 30 NEXT A`
	`> 40 TROFF`
	`> 50 FOR A = 1 TO 5`
	`> 60 NEXT A`
	`RUN`
	`[20] [30] [20] [30] [20] [30] [20] [30] [20] [30] [20] [40]`
	`>`

See also:	TRACE, TRON

tron

Syntax:	TRON [M]
Description:	Command. Turns ON the line number trace. If the M option is specified the line numbers will be output to a trace buffer.
Remarks:	The TRON command activates a debugging tool that displays program line numbers in the order they are executed by the BASIC interpreter and in the time required to change line numbers. If the M option is specified output will be redirected to a 255 element circular trace buffer. The trace buffer has a battery backup, so its contents are retained even if the power is turned OFF. When necessary, the TRACE command can be used to output the contents of the trace buffer to the terminal. The line number trace facility can be turned OFF using the TROFF statement. If the ASCII Unit is reset or if the NEW statement is executed then the line number trace will automatically be turned OFF.

Examples:	`> 10 TRON`
	`> 20 FOR A = 1 TO 5`
	`> 30 NEXT A`
	`> 40 TROFF`
	`> 50 FOR A = 1 TO 5`
	`> 60 NEXT A`
	`RUN`
	`[20] [30] [20] [30] [20] [30] [20] [30] [20] [30] [20] [40]`
	`>`

See also:	TRACE, TROFF

val

Syntax:	VAL (<string expression>)
Description:	Function. Converts a character string to a numerical expression.
Remarks:	<string expression> may be any character representation of a numeric value. If the character representation does not represent a numeric value 0 will be returned. All valid numeric expressions are acceptable, including hexadecimal, octal, single and double precision floating point and exponential formatted numbers. Exponential formatted numbers must be specified with an upper-case E or D. Spaces in the numerical expression are ignored. The conversion will be terminated at the first non-numerical character. A value of 0 will be returned if the character string does not begin with a numeric character (+, -, &, ., or a number). All numeric characters following a non-numeric character will be ignored. Leading white space is stripped from the character representation before conversion.

Examples:	`> 10 A$ = "2"`
	`> 20 B$ = "3"`
	`> 30 PRINT A$+B$`
	`> 40 PRINT VAL(A$)+VAL(B$)`
	`> RUN`
	`23`
	` 5`
	`>`

See also:	STR$

varptr	
Syntax:	VARPTR (<variable>)
Description:	Function. Returns the memory address of a variable argument.
Remarks:	<variable name> must represent a valid variable in the BASIC program that has already had a value assigned to it. The address of the value field in the identifier table is returned. If the variable is an array variable, the memory address where the leftmost bit of the memory block address of the array is stored will be returned as decimal data. Refer to *Appendix D Formats for Storing Variables in Memory* for details on variable storage formats.

Examples:	`> 10 A = 5`
	`> 20 PRINT VARPTR(A)`
	`> RUN`
	`12321`

See also:	

wait	
Syntax:	WAIT <wait string> [, <line number>]
Description:	Statement. Specifies a time-out for the statement following the WAIT statement.
Remarks:	<wait string> specifies the waiting time, after which a time-out will occur. It has the format " [<mm>:]<ss>.<t>" where : <mm> specifies minutes, valid range is *[00…59]*. This field is optional. <ss> specifies seconds, valid range is *[00…59]*. This field is mandatory. <t> specifies tenths of seconds, valid range is *[0…9]*. This field is mandatory. <line number> specifies the program line to continue execution after a time-out has occurred. It can be any valid BASIC line number in the range : *[1…65535]*. If the line number specified does not exist then an "UNDEFINED LINE NUMBER" error (code B008) will occur. If an illegal wait time is specified an "ILLEGAL FUNCTION CALL" error (code B005) will occur. The WAIT statement monitors the statement that immediately follows it. If it finishes execution within the specified time then program execution will continue as normal. If not then execution of the monitored statement will terminate and execution will continue. If <line number> is specified then, in the event of a time-out, execution will continue at the specified line number. If not then execution will continue with the statement immediately following the monitored statement. The WAIT statement can be used to monitor the following statements: INPUT, INPUT$, LINE INPUT, PC READ, PC WRITE, PC QREAD, PC QWRITE, PRINT and LPRINT, FOR, WHILE. If another BASIC interrupt occurs while the WAIT statement is active, that BASIC interrupt subroutine will be executed. The WAIT timer will be paused while the interrupt subroutine is being executed.

Examples:	`> 10 WAIT "00:1.5", 100`
	`> 20 INPUT A$`
	`> 30 PRINT "HELLO ";A$`
	`> 40 END`
	`> 100 PRINT "TOO LATE..."`
	`> 110 END`
	`> RUN`
	`? TOO LATE...`

See also:	

watch

Syntax:	WATCH [(SET | DEL) <variable name>]
Description:	Command. Sets, deletes or lists watched variables in the BASIC debugger. This command can be used to identify the values of watched variables when the program stopped.
Remarks:	<variable name> must represent a valid variable in the BASIC program that has already had a value assigned to it. Up to 16 variables can be watched simultaneously. The <variable name> cannot be specified as an array variable. If an array variable is specified as the <variable name>, a "NO SUPPORT" error (code 0064) will occur. To monitor array variables, use PRINT array variables. WATCH SET is used to add variables to the watch list. WATCH DEL is used to remove variables from the watch list. If neither SET nor DEL are specified, the watch list will be listed. Once the watch variables have been set, the value of the variables will be displayed every time that the BASIC program is stopped. After running RUN compile, specify the variables again using the WATCH command.

Examples:	`> WATCH SET A$`
	`> WATCH SET B`
	`> 10 A$ = ""`
	`> 20 FOR B = 1 TO 9`
	`> 30 A$ = A$ + STR$(B)`
	`> 40 STOP`
	`> 50 NEXT B`
	`> RUN`
	`BREAK IN LINE 40`
	`A$ = 1`
	`B = 1`
	`> CONT`
	`BREAK IN LINE 40`
	`A$ = 1 2`
	`B = 2`
	`>`

See also:	BRKPT, CONT, STEP, STOP

while / wend	
Syntax:	WHILE <numerical expression> <program> WEND
Description:	Statement. Conditionally repeats the series of statements enclosed between WHILE and WEND.
Remarks:	The results of <numerical expression> determines the termination of the loop. If the result of the <numerical expression> is zero (or FALSE) then the loop will terminate and execution will resume at the line number following the WEND statement. If the <numerical expression> is non-zero (or TRUE) then the loop will continue. When the WEND statement is reached execution jumps back to the WHILE statement containing the condition. If the condition is never true then the statements in the WHILE/WEND loop will never be executed. Unlike FOR/NEXT loops that have a specified number of repetitions, the WHILE/WEND loops have a repetition condition. For example, use a WHILE/WEND loop if a particular process is needed to repeat until X=0 and the number of repetitions that are required is unknown. WHILE/WEND must always be used in pairs. If a wend statement is encountered without a corresponding while then a “MISMATCH: WHILE/WEND” error (code B024) will occur. WHILE/WEND statements may be nested and the only limit on nesting is the amount of available memory.

Examples:	`10 A = 0`
	`20 WHILE A<10`
	`30 PRINT A`
	`40 A = A+1`
	`50 WEND`

See also:	@IF, FOR

7-7 User-defined BASIC Functions

User-defined BASIC functions are expressions created by the user in BASIC and saved as user-defined functions. Other BASIC functions can be called up and executed by their specified function name in the same way. Function names, input variables, and expressions are declared using BASIC's DEF FN command.

User-defined BASIC Function Procedure

1,2,3...

1. Declare the following details using the DEF FN command.
   - BASIC function call name
   - Variable name input to the BASIC function
   - Expression
2. Execute the program using the BASIC function name.

User-defined BASIC Function Definition (DEF FN)

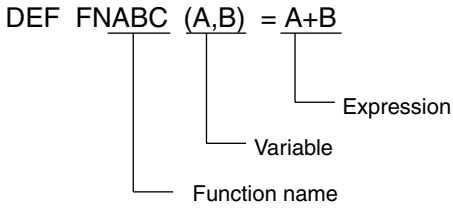
Syntax

DEF FN *function_name* (*variable_name*[, *variable_name*]) = *expression*

Meaning

DEF FN declares a user-defined BASIC function. It sets the function name (same restrictions as for variable names) and the variable names for input to functions (more than one name is possible), and the expression.

Example:



BASIC Function Calls

As for other functions, used-defined functions are called by specifying the function name and then the variable(s) within parentheses.

Example: C = FNABC (A,B)

Note Text strings starting with reserved words cannot be used for function names. The following characters cannot be used at the end of function names: $ % & ! #

7-8 Debugging

7-8-1 Overview

The debugging function allows you to stop the program at a specified line by various methods so that the contents of variables can be monitored. It is also possible to determine the position of the current line number.

Note The breakpoint function can be used to set a breakpoint in an interrupt subroutine, but it is not possible to continue executing the interrupt subroutine after the breakpoint.

Breaks

Break at a specified line	By command input from the terminal	Break at multiple lines (255 settings max.)	Input BRKPT command
		Break after a single line or specified number of lines	Input STEP command
		Execute up to the specified line	Input the RUN TO line no.
	By internal program command	Break at the specified line	STOP command

Program error

Break condition

Break condition

Variable Monitor

Variable content monitor (16 variables max.)	By command input from the terminal	Input WATCH command

Tracing the Current Line Number

By internal program command	TRON command
Save in the trace buffer, then monitor	TRON[M] command to TRACE command input

Each of the following debugging functions are described below.

- Break point function
- STEP function
- Variable monitor function
- Trace function

7-8-2 Breakpoint Function (BRKPT)

When a breakpoint is set at a specified line using the BRKPT command, the program will stop at that point. Once it has stopped, a message indicating a break in execution and the current line number will be displayed at the terminal. The ASCII Unit will switch to Command Mode. Up to 255 breakpoints can be set. To restart the program from the breakpoint, enter the CONT command from the terminal.

Syntax

BRKPT (SET | DEL) *line_No.*

Meaning

BRKPT sets and deletes breakpoints. When no line is specified, the current breakpoint settings are displayed.

Example

BRKPT SET 200:	Sets breakpoint at line 200.
BRKPT DEL 100:	Deletes breakpoint at line 100.
BRKPT:	Displays the current breakpoint settings.
BRKPT DEL ALL:	Deletes all the current breakpoint settings.

7-8-3 Step Function (STEP)

Each time the STEP command is executed, the program executes a single line or a specified number of lines.

Syntax

STEP [*numeric_expression*]

Meaning

The specified number of lines are executed. If no number is specified, 1 line is executed.

Example

STEP 5:	5 program lines are executed.
STEP:	1 program line is executed.

7-8-4 Variable Monitor Function (WATCH)

It is possible to monitor the contents of specified variables (except array variables) using the WATCH command. If the WATCH command is input, then whenever the BASIC program breaks due to a STOP command, a STEP command, a breakpoint, or a program error, the display will indicate that a break has occurred and the current line number, and then the value of the variable to be monitored will be shown at the terminal. The variables are displayed in order in which the WATCH commands were input. A maximum of 16 variables can be monitored at one time.

Syntax

WATCH (SET | DEL) *variable_name*

Meaning

WATCH sets or deletes a specified variable to be monitored. When no variable is specified, the list of currently monitored variables is displayed. A maximum of 16 variables can be monitored at one time. An array variable cannot be specified for the variable name.

Example

WATCH SET A$:	Sets A$ to be monitored.
WATCH DEL B:	Stops monitoring B.
WATCH:	Displays the current list of variables being monitored.

Examples of Variable Monitoring

WATCH SET B$:	Sets B$ to be monitored
WATCH SET A:	Sets A to be monitored

When there is a break at line number xxx after the RUN command,
BREAK IN LINE xxx
B$ = “ABCD”
A = 3

Note Set the variables using the WATCH command after the program has been compiled for the RUN command.

7-8-5 Trace Function (TRON and TRACE)

When the TRON command is executed, the executed program lines are displayed at the terminal in order of execution. This allows you to see execution for programming such as for FOR to NEXT repetitions, and IF GOTO and GOSUB subroutine branching. When using the TRON command with the M option, trace data is stored in the trace buffer, and can be displayed at the terminal later using the TRACE command.

The trace buffer can store a maximum of 255 lines (when 255 lines are exceeded, lines are overwritten starting from the first line executed.)

Note Trace details stored in the trace buffer are backed up by a battery even after the power supply is turned off. This means that they can be read afterwards using the TRACE command. The trace buffer is cleared using the NEW or TROFF command.

TRON

Syntax TRON[M]

Meaning

TRON starts a trace of the lines that follow. After RUN, the executed lines are displayed at the terminal enclosed in brackets [ ]. If the option [M] is used, the trace is output to the trace buffer. A maximum of 255 lines can be traced.

Example

TRON: Starts a trace of the lines that follow

TRON[M]: Starts a trace of the lines that follow, and stores the trace in the trace buffer.

Examples of the Trace Function

RUN
[10][20][30][20][40]....: Lines were executed in the order [10][20][30][20][40].

TRACE

Syntax TRACE

Meaning

TRACE displays the contents of the trace buffer at the terminal. Executed lines are displayed sequentially inside square brackets [ ] in the order in which they were executed.

Examples of the Trace Function

TRON M

TRACE
[10][20][30][20][40].....: Lines were executed in the order [10][20][30][20][40].

Note Only when the execution line number is changed, the line number is added to the trace buffer. Therefore, if a line such as 100 GOTO 100 is executed, TRACE will display [100] one time only.

A DEVICE UNAVAILABLE IN PORT X error will occur if line numbers displayed for traces are duplicated when displaying program trace results on the terminal while executing an interrupt subroutine. If this occurs, use the M option for the TRON command to store trace results in the trace buffer.

SECTION 8
Data Exchange Application Programs

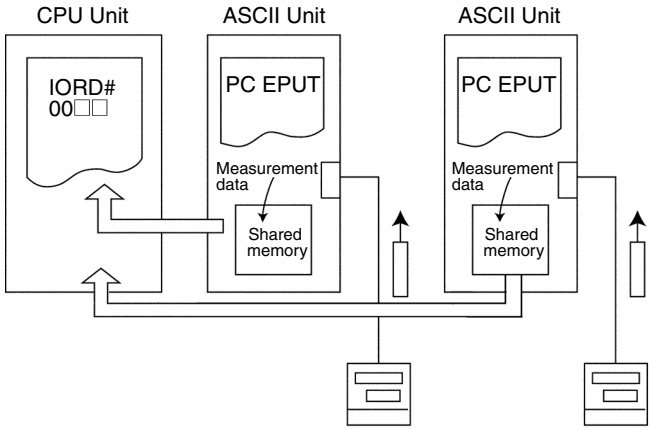
This section provides examples of data exchange applications.

8-1 Asynchronous Processing 236
8-2 High-speed Data Exchanges 237
8-3 High-volume Data Exchanges. 239
8-4 Data Transfer: ASCII Unit to CPU Unit 239
8-5 Bit Data Exchanges 240

8-1 Asynchronous Processing

Example 1

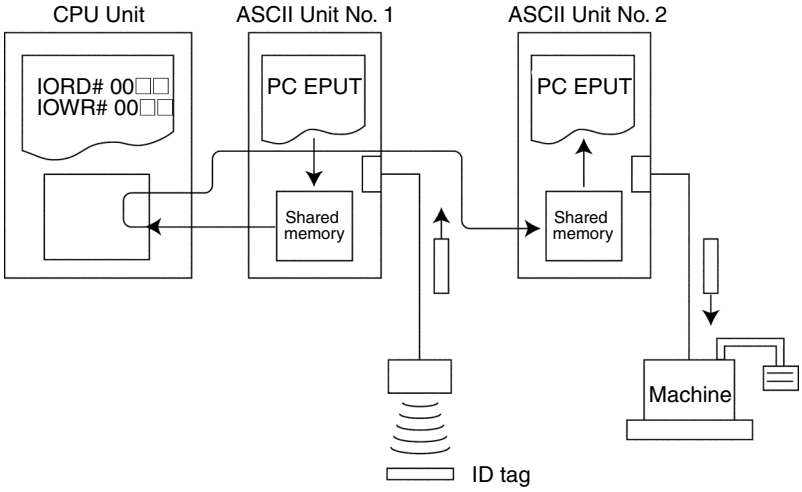
The monitoring system allows measurement data (PV) from a controller to be set whenever required in the shared memory of the ASCII Unit. The CPU Unit reads this data from the ASCII Unit's shared memory as required to monitor it.

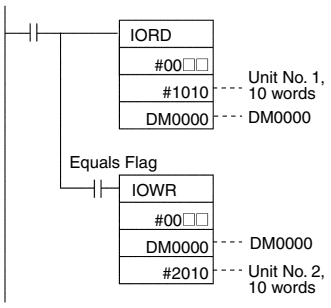


ASCII Unit		CPU Unit
Refreshes at high-speed the measurement data from the controller to the ASCII Unit's shared memory using the PC EPUT command.	→	Reads measurement data from the ASCII Unit as required,using the IORD (#00□□) instruction.

Example 2

Here, various data is transferred between ASCII Units. The CPU Unit acts as an intermediary.



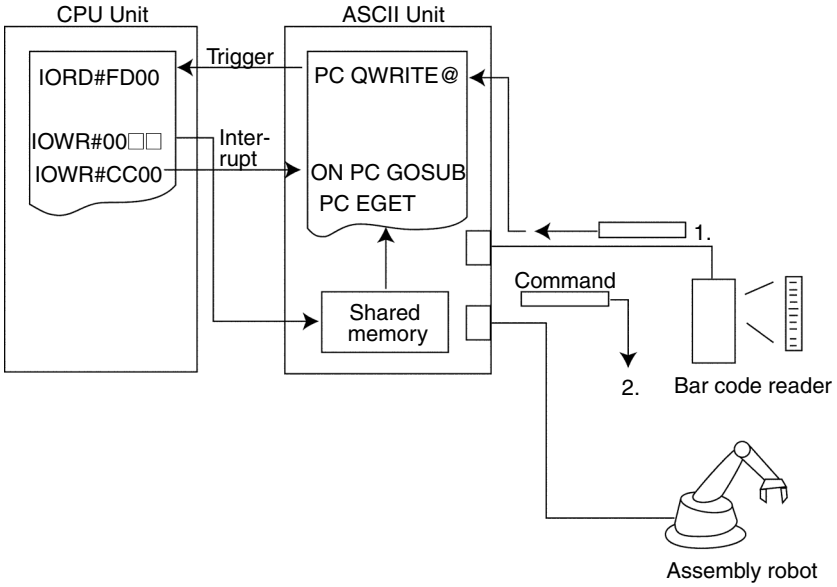
CPU Unit	ASCII Unit
 Reads from the shared memory of ASCII Unit No. 1 to DM 0000 to DM 0009 Writes to the shared memory of ASCII Unit No. 2 from DM 0000 to DM 0009 IORD (#00□□) and IOWR (#00□□) executed continuously.	Writes to shared memory using the PC EPUT command (ASCII Unit No. 1). Reads from shared memory using the PC EGET command (ASCII Unit No. 2).

8-2 High-speed Data Exchanges

Example 1

When the ASCII Unit functions as a communications interface, the trigger from the ASCII Unit enables CPU Unit processing to activate a trigger from the CPU Unit.

In response to data from a bar code reader, an assembly robot is controlled via CPU Unit processing.

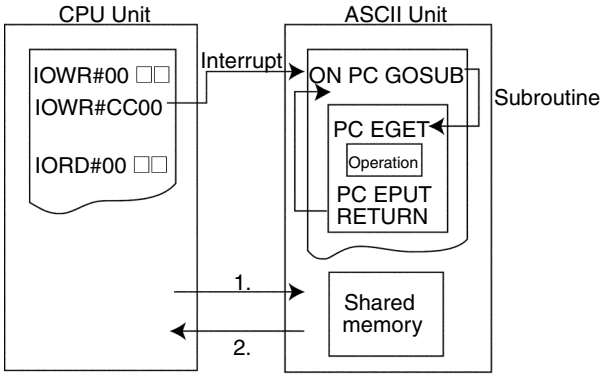


	ASCII Unit		CPU Unit
1	Sends an IORD Request Flag when the bar code reader data is input.	→	Receives the IORD Request Flag, and receives data via the IORD (#FD00) instruction.
			↓
2	After receiving the interrupt, the ASCII Unit reads the data written to the shared memory via the PC EGET command in the interrupt subroutine, and then issues commands to the assembly robot.	←	After processing, writes data to the ASCII Unit shared memory via the IOWR (#00□□) instruction, then sends an interrupt to the ASCII Unit via the IOWR (#CC00) instruction.

Examples 2

When the ASCII Unit operates as a special BASIC processing unit, a trigger is sent from the CPU Unit to the ASCII Unit, which after processing, sends a trigger back to the CPU Unit.

The CPU Unit assigns arithmetic processing outside its capability to the ASCII Unit, enabling high-speed processing for both Units by sharing the workload.

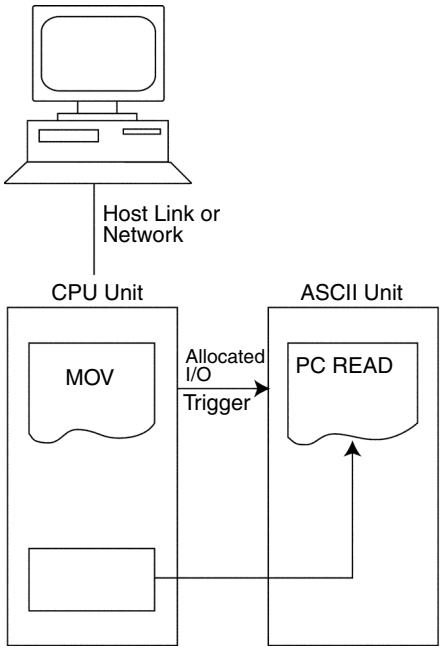


	CPU Unit		ASCII Unit
1	Writes data to the shared memory via the IOWR (#00□□) instruction, then sends an interrupt to the ASCII Unit via the IOWR (#CC00) instruction.	→	Receives the interrupt, reads the data written to the shared memory via the PC EGET command in the interrupt subroutine, then carries out special arithmetic operations.
			↓
2	At the completion of the interrupt subroutine (determined by checking the Interrupt Subroutine Completed Flag), reads from the shared memory via the IORD (#00□□) instruction.	←	Writes the result of the processing to the shared memory via the PC EPUT command; end of the interrupt subroutine.

8-3 High-volume Data Exchanges

Example

Large volumes of data can be transferred when high speed is not required by sending production directions as recipe data (settings/parameters) to various machines from the host computer.

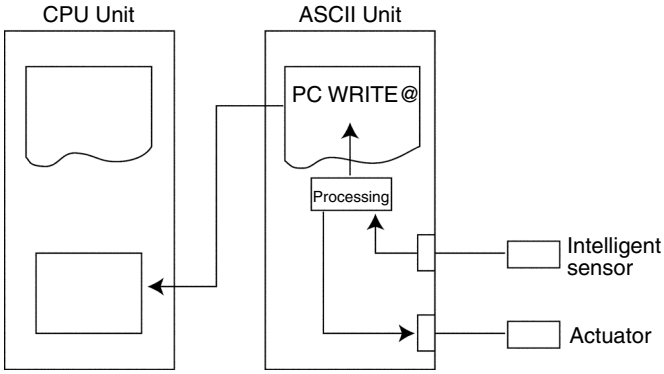


CPU Unit		ASCII Unit
Sends a Write Bit to the ASCII Unit via allocated IR/CIO words.	→	Reads the CPU Unit memory via the PC READ command to a maximum of 255 words. During the I/O refresh period (1 cycle), the maximum is127 words.

8-4 Data Transfer: ASCII Unit to CPU Unit

Example 1

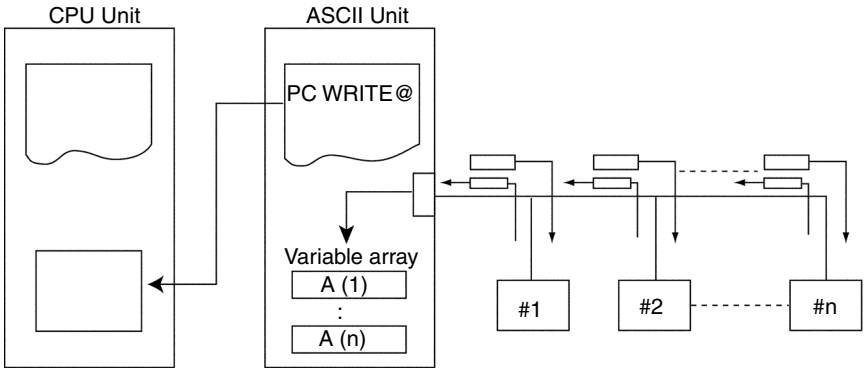
The ASCII Unit is used as a control unit to upload status monitoring data.



ASCII Unit		CPU Unit
Processes data from the sensor and outputs it to the actuator. The ASCII Unit uploads status monitoring data as required via the PC WRITE@ command.	→	Reads status monitoring data.

Example 2

Multiple external devices are connected via the RS-422/485 ports, and polling is used to send commands. The response data is stored in variable arrays and uploaded to the CPU Unit as required.

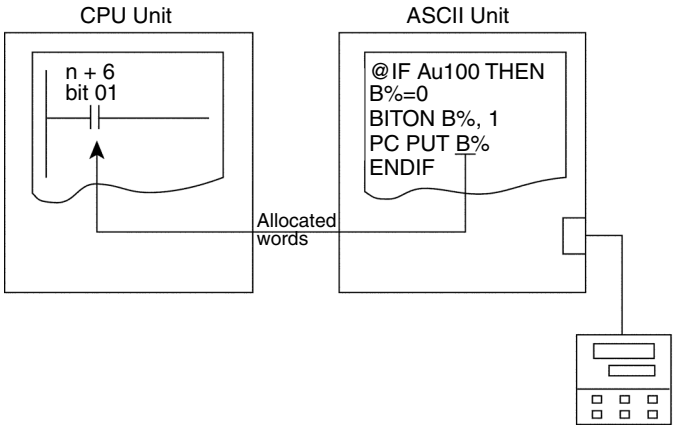


ASCII Unit		CPU Unit
Response data is stored in the variable arrays. The ASCII Unit uploads as required using the PC WRITE@ command.	→	Reads the response data.

8-5 Bit Data Exchanges

Example

A measured temperature from a temperature controller sends an ON signal to the CPU Unit once a certain temperature is reached.



ASCII Unit	CPU Unit
After bit 01 of the numeric variable B% is turned ON, B is set in word n + 6 via the PC PUT command.	Reads word n + 6 bit 01.

SECTION 9
Clearing Error Messages

This section provides a list of error messages, probable causes and possible corrections.

9-1 List of Error Messages 242
9-2 Troubleshooting 248
9-2-1 Procedure for Dealing with Damage to the Program Memory. 251
9-3 CPU Unit Error Indicators 252
9-3-1 Special I/O Unit Error List 252
9-3-2 Useful Flags and Control Bits. 253
9-4 Reading and Clearing the Error Log 254
9-4-1 Overview of Error Log Table 254
9-4-2 Reading the Error Log Table 255
9-4-3 Clearing Errors 255
9-5 Maintenance 255
9-5-1 Mounting Precautions. 255
9-5-2 Replacing Batteries. 256

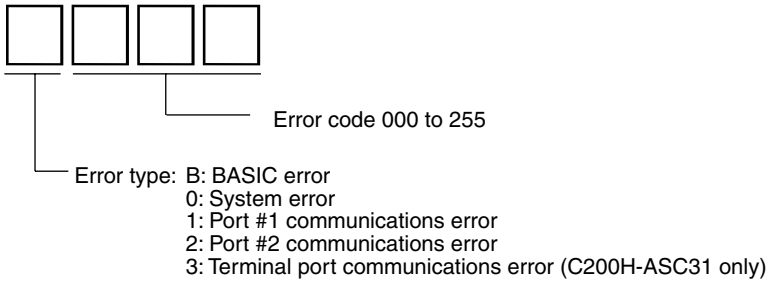
9-1 List of Error Messages

If an error (BASIC error) is generated during the execution of a BASIC command, or a system error is generated, the error code is stored in bits 00 to 11, and the error type is stored in bits 12 to 15 of the IR/CIO area (words n + 7) in the CPU Unit. At the same time, the relevant error message is displayed on the terminal. (Only an error code for errors 0080 to 0099 generated by the CPU Unit interface is stored in words n + 7, and no error message is displayed.)

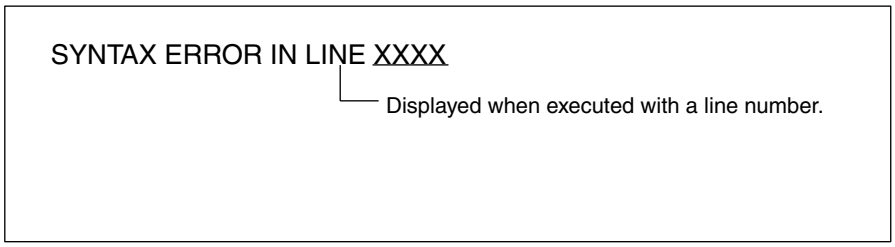
IR/CIO Area Allocation

Word	Bit	Name	Value
n +7	0 to 11	Error Code	000 to 255 BCD
	12 to 15	Error Type	B: BASIC error 0: System error 1: Port #1 communications error 2: Port #2 communications error 3: Terminal port communications error (C200H-ASC31 only)

Error Type and Code



Example Terminal Display



Note If the execution of a BASIC program is stopped when the terminal port is configured as a non-terminal device, the ASCII Unit will wait for a message to be displayed. In this case, connect the terminal to the terminal port and press Ctrl+X Keys, and the message will be displayed.

The various error types, codes, error messages and actions are as follows:

BASIC Errors

Error code	Error message	Probable cause	Corrective measures
B001	NEXT WITHOUT FOR	FOR command is missing.	Confirm the FOR....NEXT combination.
B002	SYNTAX ERROR	The program syntax is not correct. A variable name or label name starts with the same characters as a reserved word (command name).	Correct the program line. Change the variable or label name.
B003	RETURN WITHOUT GOSUB	Encountered RETURN command before the GOSUB command was executed.	Confirm the GOSUB....RETURN syntax.
B004	OUT OF DATA	The data to be read by the READ command is missing. The number and type of READ command variables and the number and type of the DATA command constants do not match or RESTORE command is not used.	Confirm the number and type of READ command variables, and the number and type of DATA command constants.
B005	ILLEGAL FUNCTION CALL	Illegal statement or function call.	Confirm the callout method for statements and functions (syntax).
B006	OVERFLOW	The numerical value exceeds the permissible range.	Check the numerical value range and the variable type.
B007	OUT OF MEMORY	Insufficient memory capacity.	There is not enough user memory. Check the status of memory use by using PINF or other command, and review the program, or free some memory.
B008	UNDEFINED LINE NUMBER	Input line number is incorrect.	Check the destination line number.
B009	BAD SUBSCRIPT ERROR	The array variable subscript exceeds the maximum value specified using the DIM command.	Use a value within the maximum subscript value range specified using the DIM command.
B010	DUPLICATE DEFINITION	The definitions of array variables or user-defined BASIC functions have been duplicated.	Change one of the duplicated array variables or user-defined BASIC functions.
B011	DIVISION BY ZERO	A division by zero has occurred.	Check the division and correct the program.
B012	ILLEGAL DIRECT	Execution of a statement has been attempted that is illegal in the Command Mode.	Can only be executed in a BASIC program (Program Mode).
B013	TYPE MISMATCH	The variable type does not match.	Check the variable type.
B014	---	Not used	---
B015	---	Not used	---
B016	---	Not used	---
B017	CANNOT CONTINUE	The program execution cannot be continued.	Restart the program using the RUN command.
B018	UNDEFINED USER FUNCTION	The user function is not defined.	Define the user function using the DEF FN command.
B019	NO RESUME	There is no RESUME command in the error interrupt subroutine within the ON ERROR GOTO command.	Add a RESUME command.
B020	RESUME WITHOUT ERROR	Although there is no error, the RESUME command has been executed.	Check the program.
0021	---	Not used	---
B022	MISSING OPERAND	The required parameters (operands) are missing.	Check the program syntax.

Error code	Error message	Probable cause	Corrective measures
B023	FOR WITHOUT NEXT	The NEXT command or the WHILE command is missing.	Check the FOR...NEXT or WHILE...WEND combinations.
B024	MISMATCH: WHILE/WEND	The WEND or WHILE command is missing.	Add a WEND or WHILE command.
B025	UNDEFINED SYMBOL	The character is outside the permissible range or a label is missing.	Use a character within the permissible range or check the program label.
B026	INVALID EXPRESSION	An invalid expression has been used.	Correct the expression.
B027	INVALID STATEMENT	An invalid statement has been used.	Correct the statement.
B028	---	Not used.	---
0029	MISMATCH: @IF/ENDIF	@IF and ENDIF do not match.	Check that @IF and ENDIF match.

System Errors

Error code	Error message	Probable cause	Corrective measures
0030	FATAL ERROR: BUS ERROR	System error. (The ASCII Unit is not operating properly.)	Turn OFF the power, and then turn ON again. If the error persists, replace the Unit.
0031	FATAL ERROR: ADDRESS ERROR		
0032	FATAL ERROR: PRIVILEGE VIOLATION		
0033	FATAL ERROR: UNINTIALISED INTERRUPT		
0034	FATAL ERROR: SPURIOUS INTERRUPT		
0035	FATAL ERROR: ILLEGAL INSTRUCTION		
0036	RTC ACCESS ERROR		Check battery and RTC. Otherwise, exchange Unit.
0037	FATAL ERROR: MPU IN LIMP MODE ERROR		Turn OFF the power, and then turn ON again. If the error persists, exchange the Unit.
0038	FATAL DEBUG ERROR: SYSTEM DEBUG ERROR IN PORT XXXX		
0039	---	Not used.	---

Operational Errors

Error code	Error message	Probable cause	Corrective measures
0040	BREAK EXECUTED IN LINE XXXX	---	---
0041	CANNOT RUN WITH SWITCH IN STOP POSITION	Attempted to execute the RUN command with the switch in the STOP position.	Set the switch to START.
0042	USER BREAK	Program stopped by user.	---
0043	FILE TRANSFER INTERRUPTED	File transfer stopped while executing.	---
0044	PROGRAM MEMORY ERROR	Corrupted memory.	Toggle the START/STOP switch to initialize the memory and restart.
0045	---	Not used.	---
0046	---	Not used.	---
0047	---	Not used.	---
0048	---	Not used.	---
0049	---	Not used.	---

Error code	Error message	Probable cause	Corrective measures
0050	BAD PORT NUMBER	The port number is incorrect.	Check the port number.
0051	CANNOT READ FROM WRITE-ONLY DEVICE	Inputting to a write-only external device.	Check the port and external device specifications.
0052	PORT ALREADY OPEN	The port is already open. The same port has been opened twice using the OPEN command.	Delete one of the OPEN commands.
□053	PARITY ERROR COMMUNICATING WITH DEVICE	An error has been generated while communicating with an external device.	Check the communications network.
□054	---	Not used.	---
0055	PORT CONFIGURATION ERROR	Incorrect port communications parameters. If the ON COM interrupt is being used, the port communications parameters will not be set to proper values.	Check the communications parameters for the port or check that the restrictions on baud rate and communications interrupt conditions for port 1 and 2 have not been exceeded.
□056	DIRECT STATEMENT	When the LOAD command was executed, a line with no line number was discovered in the program being transferred to the ASCII Unit.	Check the program being transferred to the ASCII Unit.
0057	PORT IS NOT OPEN	An unopened port number has been used.	Open the port using an appropriate OPEN command for the device symbol.
□058	FRAMING ERROR COMMUNICATING WITH DEVICE	Incorrect framing (framing for start bit + data + parity + stop bit) for received data read from the port.	Check whether or not the communications conditions specified using the OPEN command match the framing of the external device.
0059	CANNOT WRITE TO READ ONLY DEVICE	Attempted to write to a read-only external device.	Check whether or not the device symbol specified using the OPEN command matches the external device specifications.
0060	DEVICE UNAVAILABLE	An invalid device symbol has been used in the OPEN command.	Check the OPEN command device symbol.
□060	DEVICE UNAVAILABLE IN PORT□	I/O commands such as a PRINT command may be in conflict with each other due to an interrupt.	Correct the program so that I/O commands do not compete.
0061	---	Not used	---

Note The □ indicates the port number generating the error.

Execution Errors

Error code	Error message	Probable cause	Corrective measures
0062	PROTECTED PROGRAM	Program is protected from the NEW command.	When creating a new program using the NEW command, delete the program name using the PNAME“ ” command. If PMEM ON has been specified, execute the NEW command after executing PMEM OFF.
□063	INCOMING DATA HAS BEEN LOST	The data to be received has been lost.	Reduce the baud rate or use flow control.
0064	NOT SUPPORT	There is no support. Automatic transfer was specified even though the program is not saved in flash ROM	Check the positions of the OPTION LENGTH command and other commands for defining variables. Check the positions of commands.
0065	FLASHROM ERROR	There is an error with the flash ROM, or nothing has been written to the flash ROM.	Try again or replace the Unit.
0066	VERIFY ERROR	Error generated when comparing and verifying flash ROM	If there is no problem with the program in the user memory (RAM), rewrite the correct data to the flash ROM using the ROMSAVE command.
B067	FORMAT ERROR	The format or variable specifications for the PC READ or PC WRITE command are incorrect, or the number of transfer words or the specified first transfer word of the source in the IR/CIO area of the CPU Unit are incorrect.	Check the contents of the PC READ/ PC WRITE command at the ASCII Unit and words n + 3 and n + 4 in the IR/CIO area.
0068	---	Not used.	---
0069	PASSWORD INCORRECT	The password is incorrect.	Input the correct password.
0070	DATA TRANSFER CHECK SUM ERROR	Data sum error generated during data transfer or nothing written in flash ROM.	Perform the data transfer again.
0071	---	Not used.	---
0072	---	Not used.	---
0073	---	Not used.	---
0074	---	Not used.	---
0075	---	Not used.	---
0076	---	Not used.	---
0077	---	Not used.	---
0078	---	Not used.	---
0079	---	Not used.	---

Note The □ indicates the port number generating the error.

PC Interface Errors

Error code	Error message	Probable cause	Corrective measures
0080	No message.	The IOWR/IORD instruction is incorrect.	Check the operands for the IOWR/IORD instructions.
0081	No message.	Incorrect number of transfer (read/write) words in the IOWR/IORD instruction.	Set the IOWR/IORD (#00□□) instruction to a maximum of 90 words and the IOWR/IORD (#FD00) instruction to a maximum of 128 words.
0082	No message.	Interrupt to the ASCII Unit using the IOWR (#CC00) instruction is incorrect.	Check whether or not the IOWR(#CC00) instruction interrupt number matches the ON PC (interrupt number) command.
0083	No message.	When reading and writing to the shared memory using the IOWR/IORD (#00□□) instruction, the first read/write word (offset word □□ from the beginning of the IOWR/IORD Area) is incorrect.	Check the offset word □□ in the IOWR/IORD (#00□□) instruction and the first words of the IOWR Area and IORD Area (m + 6 and m + 7 in the DM Area allocations).
0084	---	Not used.	---
0085	No message.	ON PC interrupt number is out of range.	Use 01 to 99.
0086	No message.	The PC interrupt number is an illegal BCD value.	Use a valid BCD value.
0087	---	Not used.	---
0088	---	Not used.	---
0089	---	Not used.	---

DM Setting Errors

Error code	Error message	Probable cause	Corrective measures
0090	No message.	Incorrect setting for program number at startup (DM Area allocations).	Set any between 00 and 04 (BCD).
0091	No message.	Incorrect automatic transfer settings from flash ROM (DM Area allocations).	Set either 00 or $5A_{hex}$.
0092	No message.	Incorrect Start Mode settings (DM Area allocations).	Set either 00 or $5A_{hex}$.
0093	No message.	Incorrect initial value transfer settings (DM Area allocations)	Set either 0 or 1 (BCD).
0094	No message.	Not used (DM Area allocations).	Set 0 (BCD).
0095	No message.	Incorrect baud rate parameters for port 1, 2, or terminal (DM Area allocations).	Set any of 00 to 08 (BCD).
0096	No message.	Incorrect number of transfer words per cycle (DM Area allocations).	After checking that the PC is a C200HX/HG/HE or CS1-series PC, set either 00 or $5A_{hex}$.
0097	No message.	Invalid screen size or terminal mode.	Set the specified value within the proper range.
0098	No message.	The first transfer word setting for the IORD or IOWR Area is incorrect (DM Area allocations).	Set either 00, or between 10 and 99 (BCD).
0099	No message.	Invalid DMA option (port 1 or 2)	Set either 00 or $5A_{hex}$.

When an error is generated, the following commands are used to include an error processing program within a BASIC program.

Situation	Command	Remarks
To branch to an interrupt subroutine when an error has been generated.	ON ERROR GOTO command	Include an error processing program in the interrupt subroutine.
To obtain the error code that has been generated.	ERR function	Use this function in the interrupt subroutine at the ON ERROR GOTO destination.
To obtain the line number where the error occurred.	ERL function	Use this function in the interrupt subroutine at the ON ERROR GOTO destination.
To simulate error generation.	ERROR command	The action that is considered to have generated the error for a specified error code is performed.

9-2 Troubleshooting

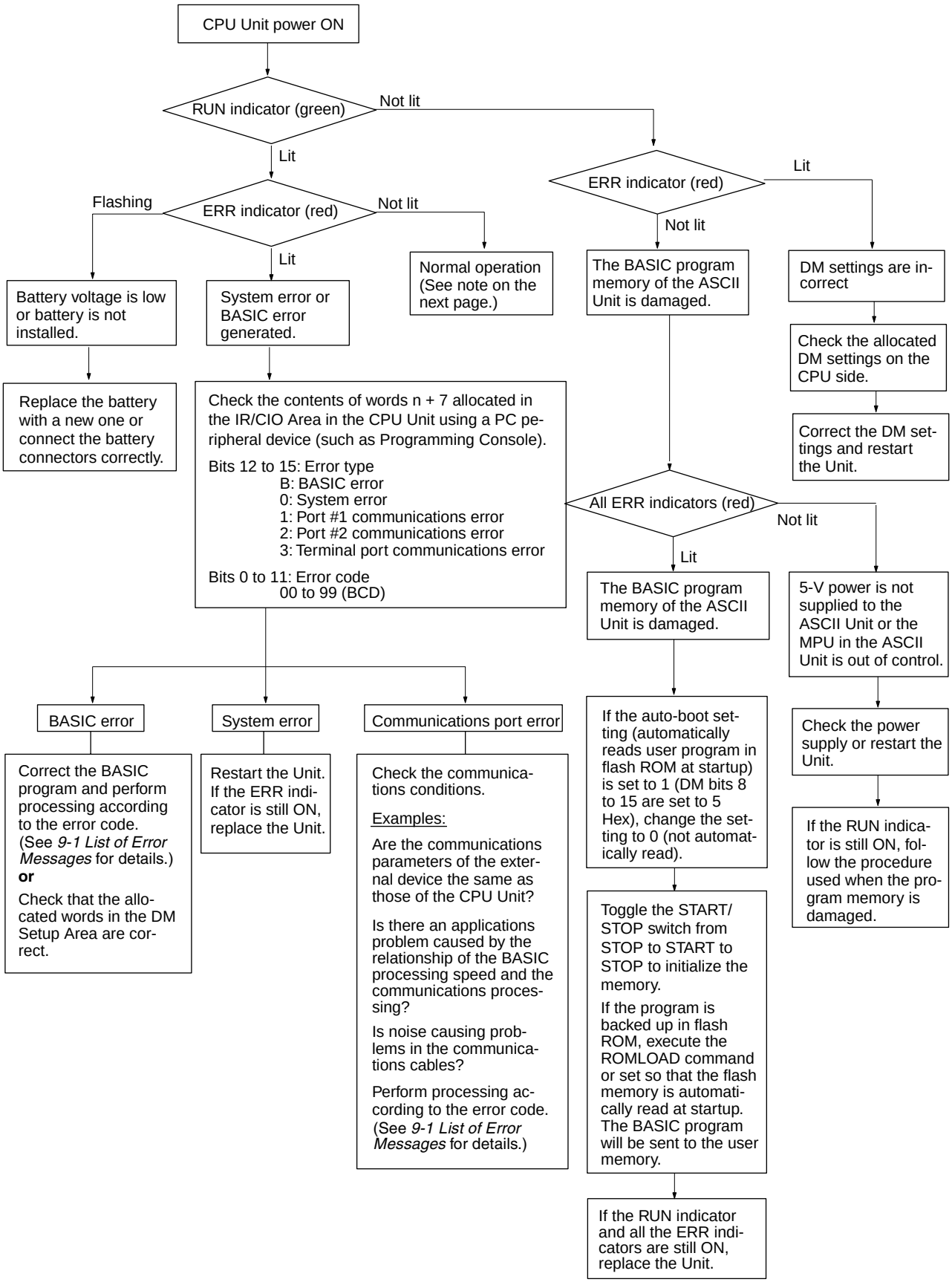
Error List

Error	Probable cause	Possible correction
None of the indicators are lit.	Power to the PC itself is turned OFF.	Turn ON the PC power.
	The ASCII Unit is not mounted securely.	Mount the ASCII Unit securely.
	One of the Special I/O Units is faulty. (Waiting for the Special I/O Unit to start up) CPU Unit does not start operation.	Exchange the faulty Special I/O Unit. The faulty Unit has “$” only displayed in the Read I/O Table operation. Before replacing the Unit, perform *Corrective Measure 1*, given below.
	The Special I/O Unit number occurs twice. (I/O Unit over) In such cases, the CPU Unit will not start operation.	Adjust the settings to ensure that the unit numbers do not occur more than once. The unit number can be viewed using Read I/O Table.
	The refresh between the CPU Unit and the ASCII Unit has not occurred properly. (Special Unit error) In such cases, only the ASCII Unit stops operating.	After eliminating the cause of the fault, Turn ON the corresponding Restart Bit (OFF→ON→OFF). If not restored after restarting, press the Ctrl+X Keys. Perform *Corrective Measure 1*, given below.
	An error has occurred in the Special I/O Unit or the CPU Unit as a result of damage to the BASIC program memory.	**Corrective Measure 1:** Set the user memory default switch on the back panel of the ASCII Unit to the right (ON) and turn the power OFF then ON again. Set the default switch back to the left and turn the power OFF then ON again. If the ASCII Unit still does not reset after this process, replace the Unit.
ERR1/2/T indicator is lit. (Port #1/port #2/terminal port communications error.)	The connecting cable is not connected to the device properly.	Connect the connecting cable correctly and secure with screws.
	The connecting cable is burnt out or the connection at the connector is faulty.	Repair or exchange the connecting cable.
	The baud rate or communications parameters of the connected device and the ASCII Unit do not match.	Match the baud rate and communications parameters of the connected device and the ASCII Unit.
ERR (error) indicator is lit. (System/BASIC error)	Incorrect data has been set for the DM Area in the PC. Alternatively, a BASIC program execution problem has been generated.	Check the error code and error message and take the appropriate action for that error.

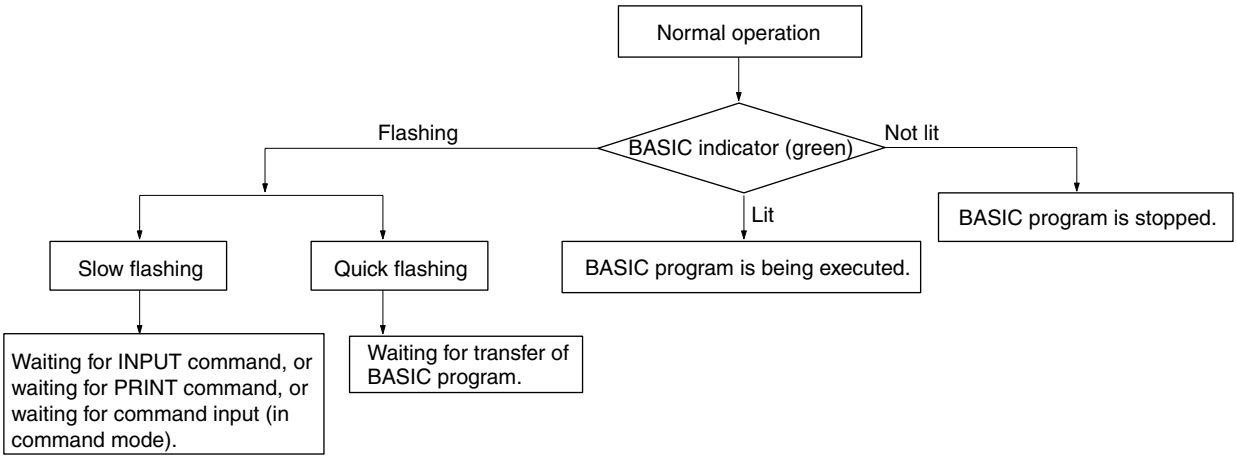
Error	Probable cause	Possible correction
ERR (error) indicator is flashing. (Battery error)	The battery connector is disconnected.	Connect the battery connector properly.
	The battery voltage has dropped.	Replace with new battery.
All ERR indicators are flashing and PROGRAM MEMORY ERROR appears on the initial screen, and pressing Ctrl+X Keys is ineffective.	The BASIC program memory is destroyed.	Change the DM setting so that the flash ROM is not read automatically at startup and then toggle the START/ STOP switch on the front panel of the Unit to START, STOP, and START again. The BASIC program memory will be cleared. If the program is backed up in flash ROM, the program memory can be recovered afterwards using the ROMLOAD command.

Note “Ctrl+X Keys” means to press the X Key while holding down the Ctrl Key.

Troubleshooting Flowchart



Note The following flowchart shows the BASIC indicator during normal operation.



9-2-1 Procedure for Dealing with Damage to the Program Memory

1,2,3...

1. Turn OFF the power supply to the PC and remove the ASCII Unit from the Backplane.
2. Set the user memory default switch on the back panel of the ASCII Unit to the right (ON) and attach the ASCII Unit to the Backplane.
3. Turn ON the power supply to the PC and check that all the indicators on the ASCII Unit are flashing (except T/R). The CPU Unit is now ready for operation. if the DM setting is set to automatically read the flash ROM at startup (DM bits 8 to 15: 5A), set to 0 (flash ROM not automatically read). (If the indicators are not flashing, replace the Unit.)
4. After checking that all the indicators are flashing, turn OFF the power supply to the PC and remove the ASCII Unit from the Backplane.
5. Set the user memory default switch on the back panel of the ASCII Unit to the left (OFF), and attach the ASCII Unit to the Backplane.
6. Turn the power to the PC ON, and check that the RUN indicator is lit. (If the RUN indicator is not lit, replace the Unit.)
7. Since the program memory is now set to default (all clear), programs need to be transferred once more to RAM (by setting DM bits 8 to 15 to 5A, so that flash ROM is read automatically at startup, etc.).

Note When performing the above procedure, make sure that the DM setting for automatically reading the flash ROM at startup is not set to read (bits 8 to 15: 00). If the setting is already set to automatic (bits 8 to 15: 5A), set to 0 and then perform the procedure.

9-3 CPU Unit Error Indicators

The CPU Unit of the C200H/HS, C200HX/HG/HE, or CS1-series PC monitors errors that occur in the ASCII Unit as described in this section. The errors are indicated for the ASCII Unit as a Special I/O Unit.

9-3-1 Special I/O Unit Error List

Error	Error cause and operation	Remedy
Waiting for Special I/O Unit startup	The program memory of the ASCII Unit is damaged or there is a hardware fault at a Special I/O Unit. The PC will not begin operation in this condition.	Perform the procedure for damage to the program memory. If the Unit still will not restart, replace the Special I/O Unit where the error occurred. The faulty Unit will be indicated by a "$**" character in the I/O Table Read operation.
I/O Unit over	Two or more Special I/O Units have the same unit number setting. The PC will not begin operation in this condition. The Special I/O Unit Error Flag (SR 25415) will be ON.	Change the unit number settings to eliminate duplications. The unit numbers can be listed with the I/O Table Read operation.
Special I/O Unit error	Refreshing between the CPU Unit and Special I/O Unit did not proceed normally. In this case, only the faulty Unit won't operate. The Special I/O Unit Error Flag (SR 25415 for C200H-family PCs; A40206 for CS1-series PCs) will be ON.	With a C200H/HS PC, determine the unit number of the faulty Unit from Error Flags AR 0000 through AR 0009. Restart the Unit by toggling the corresponding Restart Bit (AR 0100 to AR 0109) after eliminating the cause of the error. With a C200HX/HG/HE PC, determine the unit number of the faulty Unit from Error Flags SR 28200 through SR 28215. Restart the Unit by toggling the corresponding Restart Bit (SR 28100 through SR 28115) after eliminating the cause of the error. With a CS1-series PC, determine the unit number of the faulty Unit from Error Flags A41800 through A41815. Restart the Unit by toggling the corresponding Restart Bit (A50200 to A50215) after eliminating the cause of the error. If the Unit doesn't restart after toggling the Restart Bit, perform the procedure for damage to the program memory. If the Unit still will not restart, replace the Unit.

9-3-2 Useful Flags and Control Bits

Special I/O Unit Error Flags

The PC error flags in the following tables will indicate the following errors.

- Duplicated unit numbers on Special I/O Units
- Refreshing between the CPU Unit and Special I/O Unit didn't proceed normally.

Flag address				Function
C200H/HS	C200HX/HG/HE	CS1		
		Errors	Unit Number Duplication	
AR 0000	SR 28200	A41800	A41100	ON when an error occurred in Unit 0.
AR 0001	SR 28201	A41801	A41101	ON when an error occurred in Unit 1.
AR 0002	SR 28202	A41802	A41102	ON when an error occurred in Unit 2.
AR 0003	SR 28203	A41803	A41103	ON when an error occurred in Unit 3.
AR 0004	SR 28204	A41804	A41104	ON when an error occurred in Unit 4.
AR 0005	SR 28205	A41805	A41105	ON when an error occurred in Unit 5.
AR 0006	SR 28206	A41806	A41106	ON when an error occurred in Unit 6.
AR 0007	SR 28207	A41807	A41107	ON when an error occurred in Unit 7.
AR 0008	SR 28208	A41808	A41108	ON when an error occurred in Unit 8.
AR 0009	SR 28209	A41809	A41109	ON when an error occurred in Unit 9.
---	SR 28210	A41810	A41110	ON when an error occurred in Unit A (10).*
---	SR 28211	A41811	A41111	ON when an error occurred in Unit B (11).*
---	SR 28212	A41812	A41112	ON when an error occurred in Unit C (12).*
---	SR 28213	A41813	A41113	ON when an error occurred in Unit D (13).*
---	SR 28214	A41814	A41114	ON when an error occurred in Unit E (14).*
---	SR 28215	A41815	A41115	ON when an error occurred in Unit F (15).*

Note *Of the C200HX/HG/HE CPU Units, used only in the C200HX/HG-CPU5□-(Z)E C200HX/HG-CPU6□-(Z)E.

Model	Flag address	Function
C200HX/HG/HE(-Z) C200H/HS	SR 25415	ON when an error occurred in a Special I/O Unit.
CS1	A40206	
	A40113	ON with the same unit number has been assigned to more than one Special I/O Unit.

Special I/O Unit Restart Bits

To restart a Special I/O Unit, toggle (OFF → ON → OFF) the corresponding Restart Bit shown in the following table. These bits can be used to restart the Unit without turning off the power supply.

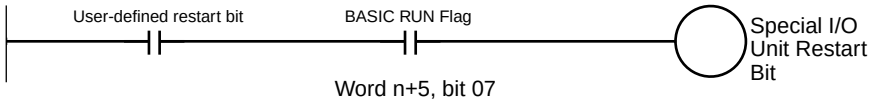
Bit address			Function
C200H/HS	C200HX/HG/HE	CS1	
AR 0100	SR 28100	A50200	Restarts Unit 0.
AR 0101	SR 28101	A50201	Restarts Unit 1.
AR 0102	SR 28102	A50202	Restarts Unit 2.
AR 0103	SR 28103	A50203	Restarts Unit 3.
AR 0104	SR 28104	A50204	Restarts Unit 4.
AR 0105	SR 28105	A50205	Restarts Unit 5.
AR 0106	SR 28106	A50206	Restarts Unit 6.
AR 0107	SR 28107	A50207	Restarts Unit 7.
AR 0108	SR 28108	A50208	Restarts Unit 8.

Bit address			Function
C200H/HS	C200HX/HG/HE	CS1	
AR 0109	SR 28109	A50209	Restarts Unit 9.
---	SR 28110	A50210	Restarts Unit A (10).*
---	SR 28111	A50211	Restarts Unit B (11).*
---	SR 28112	A50212	Restarts Unit C (12).*
---	SR 28113	A50213	Restarts Unit D (13).*
---	SR 28114	A50214	Restarts Unit E (14).*
---	SR 28115	A50215	Restarts Unit F (15).*

Note *Of the C200HX/HG/HE CPU Units, used only in the C200HX/HG-CPU5□-(Z)E C200HX/HG-CPU6□-(Z)E.

Program memory in the ASCII Unit may be destroyed if the ASCII Unit is restarted while it is executing startup processing after the power supply is turned ON. Program restarts so that they will not occur during startup processing.

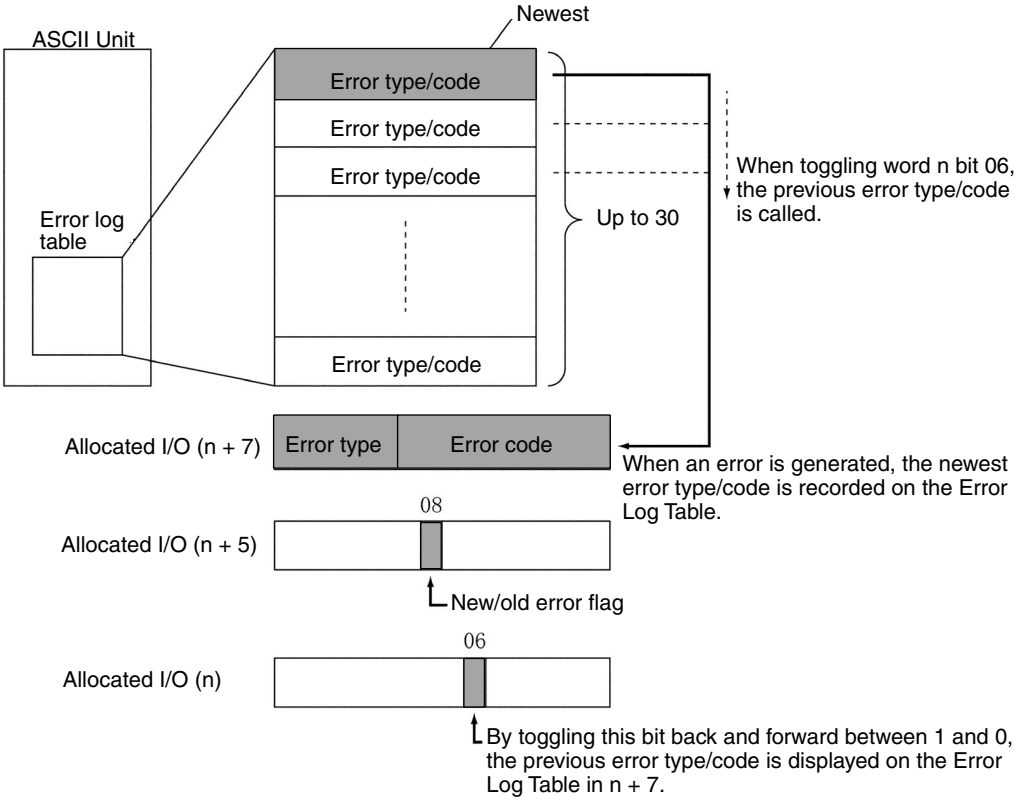
Example:



9-4 Reading and Clearing the Error Log

9-4-1 Overview of Error Log Table

The ASCII Unit has an error log table which contains up to of 30 errors. When an error occurs, the error code and type are stored in the Error Log Table.



9-4-2 Reading the Error Log Table

The Error Log Table is reflected in the Allocated I/O as shown below. Therefore, the table can be read from the CPU Unit via the Allocated I/O.

- When an error is generated, the newest error code and error type is set to n + 7 words in the IR/CIO Area Allocations.
- To trace past error codes and error types, refer to the Error Log Table when toggling the Allocated I/O word n bit 06 from 1 to 0 and 0 to 1. The previous error code and error type will be set to n+7 words. If there are no older errors the n + 7 words will become 00_{hex} once and then return to the newest error.

  Note The ERR function is used when an error is generated and that error code is to be read in the BASIC program.
- The error that has been read can be monitored to determine whether or not it originated after the program was run (i.e., old or new error information) by using word n + 5 bit 08 in the IR/CIO Area. This bit matches each error type/code for the error log.

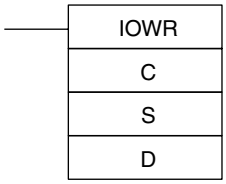
9-4-3 Clearing Errors

Errors are cleared using the ERC command as follows:

By executing the ERC command while in Command Mode, the error status at that point will be deleted and the error types and codes in the Error Log Table will also be cleared.

Clearing Errors from the ASCII Unit

The IOWR (SPECIAL I/O UNIT WRITE) instruction can be executed with a control code of #EC00 to delete error status (such as with ERC in program mode). The generated error type and code will remain in the Error Log Table.



C: #EC00
S: Any (data must be 0000)
C200HX/HG/GE
D: Bits 12 to 15: Unit number (0 to F)
Bits 00 to 11: 001_{hex}
CS1-series PCs
D: Bits 00 to 03: Unit number (0 to F)
D+1: 0001_{hex}

Clearing Error Codes Using BASIC Commands

The ERC command can be executed in Program Mode to delete the current error status. The deleted error type/codes will remain in the Error Log Table. These errors will all be given the status “old.”

9-5 Maintenance

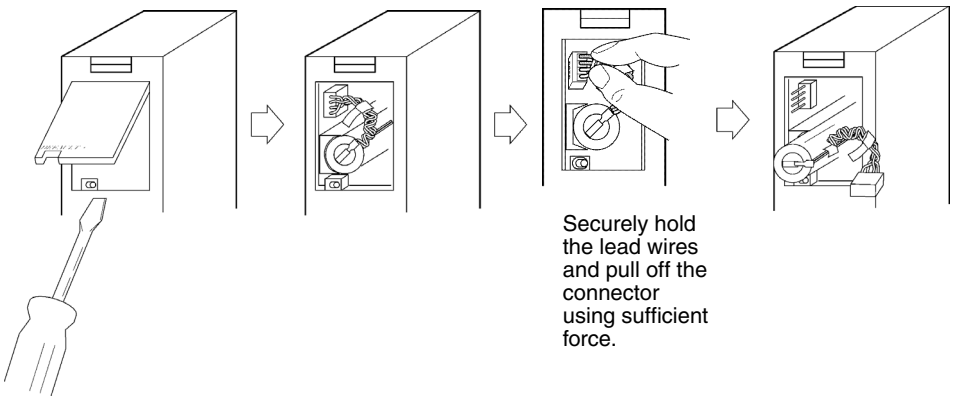
9-5-1 Mounting Precautions

- Turn OFF the power before exchanging ASCII Units.
- After exchanging ASCII Units, confirm again that the new Unit is not faulty.
- If the ASCII Unit has a fault, the user memory area (RAM) in the ASCII Unit and the BASIC program or library functions in the flash ROM cannot be read in some cases. Save the compiled BASIC program and library functions on an external device (e.g., personal computer).

9-5-2 Replacing Batteries

1,2,3...

1. Turn OFF the power. (If the power is already OFF, turn ON the power for at least one minute and then turn it OFF again.)
2. Press down on the lock lever of the Backplane with a screwdriver, and remove the ASCII Unit.
3. Remove the cover of the battery case on the back panel of the ASCII Unit using a flat screwdriver.
4. Unclip the battery connectors, and replace the batteries. This operation must be completed within five minutes.



5. Replace the battery case cover.
6. Mount the ASCII Unit to the Backplane.

Replacement Batteries

Name: Battery Set
Model: C200H-BAT 09

Battery Life and Replacement Time

- The effective period (maximum life) of a battery is five years at an ambient temperature of 25°C, whether or not the ASCII Unit is connected to electricity. If the ambient temperature is too high, the life of the batteries will be shortened.
- Replace the ASCII Unit within one week of the ERROR indicator flashing (red) on the front panel of the unit. Once the ERROR indicator starts to flash (red) on the front panel of the unit, the Battery Error Flag (word n+5 bit 06) will turn ON. Use this flag for battery maintenance.

Note Having a spare ASCII Unit and battery available will enable quick and smooth repairs on faulty Units.

Appendix A
Operating Precautions

Changing from C200H-ASC02

Execution Time

The time required by the C200H-ASC11/ASC12/ASC31 to execute commands is different from the C200H-ASC02. Therefore, the execution time of BASIC programs will depend on the ASCII Unit being used.

Machine Language Programs

Programs created on the C200H-ASC02 using machine language cannot be used with the C200H-ASC11, C200H-ASC21, or C200H-ASC31 ASCII Unit.

Integer Format

In the C200H-ASC11/ASC21/ASC31, real numbers (single-precision and double-precision) are in floating-point constant format. Since the standard has changed to conform to IEEE-754, the results of operations using real numbers may change. If precise operations are required, use double-precision variables for the arguments.

Example:
```
10 A# = 0.9 + 0.1
20 A = ACOS (A#)
```

Command Competition in Interrupt Processing

In the C200H-ASC02, an interrupt cannot be generated while a BASIC program command is being executed. This is possible, however, for the C200H-ASC11/ASC21/ASC31. Therefore, if an interrupt is generated while a command is being executed and the same command is included in the interrupt subroutine, there will be competition between the two commands and operations will not run smoothly (e.g., if an interrupt is generated while the PRINT command is executing and the interrupt subroutine contains the PRINT, INPUT, LINE INPUT command for the same port). To stop competition between commands from occurring, exclusively control interrupts by using control commands to enable, disable, and stop execution.

Handling Communications Errors

In the C200H-ASC02, the BASIC program will not stop executing due to a communications error. In the same way, if a communications error is generated with the C200H-ASC11/ASC21/ASC31 and the ON ERROR command is not used, the BASIC program will not stop executing. The BASIC program will stop executing, however, if the ON ERROR command is used and a new error (not a communications error) is generated while an error interrupt subroutine is being executed.

Internal Bit Allocations

The functions of the words and bits allocated in memory have changed. Make sure to check the allocations when using ladder programs in the CPU Unit. Functions and operations for the ASCII Busy Flag have changed, so exercise caution when using this flag.

ASCII Unit System Setup

The method of setting up the ASCII Unit system has changed from using the DIP switch on the back of the Unit to using 10 words at the beginning of the DM Allocation Area. Use a PC peripheral device to set the allocated DM words in the CPU Unit.

Communications Transmission Code and Transfer Control Signal

Depending on the device code specifications for the OPEN command, the limitations of the codes output when executing the PRINT command have changed (all transmission codes have been changed to outputs except for those specified using the TERM command for port 2). In the same way, depending on the device code specifications for the OPEN command, the operations of the transfer control signals (timing chart) have been changed. Check the interface and external devices (See *SECTION 4 Data Exchange with General-purpose External Devices*).

Terminal Emulation Mode Selection

If the same terminal used with the C200H-ASC02 is to be used with the C200H-ASC11/ASC21/ASC31 and it supports VT100 mode, use VT100 mode. If the terminal does not support VT100 mode, use FIT10 mode.

Using PC READ/PC WRITE (Including @) Character-string Variables

In the C200H-ASC02, when character-string variables are used for the variable with the PC READ/PC WRITE (including @) commands, the first four characters in the character string are written, and only these four characters are read to the variable. For the C200H-ASC11/ASC21/ASC31, all of the characters stored in the variable are written and word data up to 255 characters can be read.

Example: C200H-ASC02

PC READ “@D,0,5,5H4” ; A$,B$,C$,D$,E$

Example: C200H-ASC11/ASC21/ASC31

PC READ “@D,0,5,5H4” ; A$(Five words of data will be read from A$.)

The following example show the difference in operations between the C200H-ASC02 and the C200H-ASC11/ASC21/ASC31 when the following format is used.

Example: PC READ “@D,0,5,5H4” ; A$(0)

C200H-ASC02: The first word will be stored in A$(0).

C200H-ASC11/ASC21/ASC31: Five words of data will be stored in A$(0).

If changing from the C200H-ASC02 to the C200H-ASC11/ASC21/ASC31, the program will also need to be changed as the operations are different.

Example 1

Before changing:

10 PC READ “@D,0,10,S10H4” : A$(0)

Executing this program with a C200H-ASC11/ASC21/ASC31 will generate a FORMAT ERROR.

After changing:

```
10 PC READ “@D,0,10,S10H4” : A (0)
20 FOR I=0 TO 9 : A$(I)=RIGHT$ (“0000” + HEX$(A(I)),4) : NEXT I
```

Example 2

Before changing:

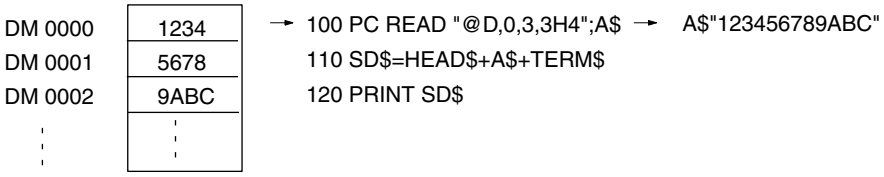
10 PC READ “@D,0,5,5H4” ; A0$,A1$,A2$,A3$,A4$

After changing:

```
10 PC READ “@D,0,5,5H4” ; A$
20 A0$=MID$(A$,1,4)
30 A1$=MID$(A$,5,4)
40 A2$=MID$(A$,9,4)
50 A3$=MID$(A$,13,4)
60 A4$=MID$(A$,17,4)
```

When using a C200H-ASC11/ASC21/ASC31, PC and DM data can be easily sent to and from peripheral devices.

Example



Example 3

Before changing:

10 PC READ “@D,0,255,100A3,100A3,55A3” ; A$,B$,C$

If the above syntax is used for a C200H-ASC11/21/31, a FORMAT ERROR will occur.

After changing:

10 PC READ “@D,0,100,100A3” ; A$
20 PC READ “@D,0,100,100,100A3” ; B$
30 PC READ “@D,0,200,55,55A3” ; C$

A maximum of 127 transfer words can be placed into multiple variables by separate the transfer values using the MID$ function in the same way as shown in Example 2.

Before changing:

10 PC READ “@D,0,4,2A3,2A3” ; A$,B$

After changing:

10 PC READ “@D,0,4,4A3” ; A$
20 A$=MID$(A$,5,4)
30 A$=MID$(A$,1,4)

For the C200H-ASC11/21/31, only one character-string variable can be used for each variable. The multiple settings that could be performed for the C200H-ASC02 are not possible. For details on the formats for other PCs, refer to *Appendix C PC Format*.

Terminal Operations

Terminal Software Limitations

VT100 mode can be used as a terminal emulation mode for the C200H-ASC11/ASC21/ASC31. Nevertheless, even if the terminal software used supports VT100 mode, the limitations of the actual terminal software functions may result in some ASCII Unit functions being unavailable. (For example, the Cursor Keys on the Hyperterminal for the Japanese version of WINDOWS 95 cannot be used.)

EDIT Command Restrictions

Do not continuously press the Up and Down Cursor Keys when using them to move lines being edited with the EDIT command. Doing so will add the Up or Down Cursor Key code to the edited lines.

Executing Programs

In the C200H-ASC02, the BASIC source program is executed as it is input. To improve the execution speed of the BASIC program in the C200H-ASC11/ASC21/ASC31, once a BASIC source program is input, it is compiled to an intermediate code (a simpler code that identifies the command contents) and then it is executed. Therefore, if the RUN command is executed using the C200H-ASC11/ASC21/ASC31, there is a delay while compiling is being performed before the actual program is executed. The RUN compiling time is approximately 10 seconds in the 53-KB (approx. 1,300 lines) program. This time will vary according to the commands used in the program. The intermediate code is regenerated the first time the program is executed after the program (area) is changed.

Saving Programs

Never turn OFF the power to the PC while writing the program to flash ROM using the ROMSAVE command (until the command prompt is displayed at the terminal). Doing so will damage the flash ROM and saving data may be no longer possible. Therefore, be sure not to disconnect power to the PC while the ROMSAVE command is being executed.

RS-422A/485 Ports

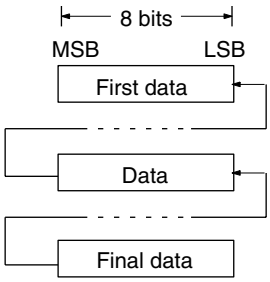
Port 2 (RS-422A/485) of C200H-ASC21 can be used in 1:N communications, however, full-duplex communications are not possible with this port.

FCS Command Error Check Code Arithmetic Operation

The calculations for CRC-16 and CRC-CCITT used with the FCS command to determine the error check code of the character expressions is described below. Check that the equations are the same as those for the external devices.

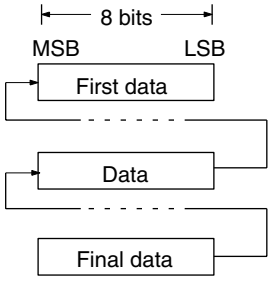
CRC-CCITT: Using XMODEM

All 8-bit data is placed in series, it is then divided by a specific 17-bit binary value (1, 0001, 0000, 0010, 0001), and the remainder (16-bit) is used as the error check code as shown below. The initial value is 0 (16 bits all 0).



CRC-16: Using MODBUS

All 8-bit data is placed in series, it is then divided by a specific 17-bit binary value (1, 0001, 0000, 0010, 0001), and the remainder (16-bit) is used as the error check code as shown below. The initial value is –1 (16 bits all 1).



Baud Rate for Receive Data Interrupt

When an ON COM interrupt is used, the baud rate of all ports will be as follows:

Port 1 + port 2 ≤ 38.4 Kbps

If the baud rate is set to exceed this condition, a PORT CONFIGURATION ERROR will be generated.

Baud Rate for PC Data Transfer

A maximum of 127 words of data can be transmitted in one cycle. If executing PC READ/PC WRITE with data of a comparatively large size (such as 127 words) and the data is being received using a high baud rate (19.2 kbps or higher), the overlap of data being received and sent will extend the time required to complete the transmission and the cycle time at the PC will also be extended. The effect on the PC cycle time can be minimized by transmitting a maximum of 20 words per cycle.

Operations after Aborting Program Execution

If a BASIC program that is being executed with an open communications port is aborted by pressing the Ctrl + X Keys, and data has already been received at the communications port, the return of the prompt after the LIST display command is executed may be delayed. To stop the delay, after the program execution has been aborted input the CLOSE command, and then execute the LIST display command.

Executing BASIC Interrupts during Command Execution

When a BASIC program is stopped due to the wait status of a command such as WAIT, a BASIC interrupt will occur (ON COM, ON PC, etc.) but only one interrupt subroutine will be executed for all BASIC interrupts. All other interrupts will held and a record made of each.

Program Startup Time

If a program is saved to ROM using ROMSAVE immediately after being debugged, the program memory may contain garbage. This may cause delays in starting program execution after turn ON the power supply. If this occurs, save the program at a terminal, load the program from the terminal, and then use ROMSAVE.

Power Interruptions or Restarts during Program Memory Operations

Do not allow PC power to be interrupted or restart the CPU Unit while program memory operations are being performed, including those for LOAD, SAVE, ROMLOAD, ROMSAVE, autoboot, and compiling for RUN. Program memory may be destroyed, preventing further operation.

Consumption of User Memory through Repetition of Subroutine Branching

Observe the following precautions when using the following two items in combination.

- 1: Subroutine using the GOSUB or interrupt command (ON COM, ON KEY, ON PC, ON TIMER, ON ALARM, ON TIME$, or ON ERROR)
- 2: GOTO, ON ERROR GOTO/RESUME, ON KEY GOTO, or WAIT command

When the subroutine using the GOSUB or interrupt command (1 above) is forced to branch using the GOTO command (2 above), data will remain in the stack area, consuming the user memory area inside the ASCII Unit. When this branching is repeated, an OUT OF MEMORY error (B007), one of the BASIC errors, will occur or the stack area will overflow, which may result in a malfunction of the system operation.

Be sure to design the program so that it always returns to the subroutine after branching using the GOTO command and exits from the subroutine using the RETURN command. Refer to the following example.

Example:

```
10   ON ERROR GOTO 100
20   PRINT FRE
30   GOSUB 1000
90   END
100  PRINT ERR,ERL;
110  RESUME 20
1000     A=1/0 'Divide by zero
1010     RETURN
```

In the above example, the subroutine will be branched from the 1,000th line to the 100th line using the ON ERROR GOTO command if an error occurs.

In this case, exit from the subroutine by changing the 110th line from RESUME 20 (to resume from the 20th line after error processing) to RESUME NEXT (to resume from the line following the line that caused an error.)

ASCII Unit Operation when Program Memory is Damaged

When program memory is damaged a PROGRAM MEMORY ERROR or a Special Unit error on the PC side will occur and CPU Unit operation will be held. The PC can be restarted by using the default switch on the back panel of the ASCII Unit. In this situation, the program memory will be forcibly initialized, so always be sure to use ROMSAVE to save programs to flash ROM memory after inputting them. Refer to *Reading the BASIC Program from Flash ROM* in *7-1 Programming Procedure*, for details on how to load the saved program to flash ROM.

Be careful when overwriting data in the user memory area using the POKE command, as damage to the program memory may result. When damage to the program memory is caused by the use of the POKE command, a memory error may not occur immediately.

Appendix B

Comparison with C200H-ASC02

The following table provides details of the differences between the C200H-ASC11/ASC21/ASC31 and the C200H-ASC02 ASCII Units. For an overview of the comparison, see *1-5 Comparison with Previous ASCII Units*.

Item		C200H-ASC02	C200H-ASC11/21/31
Executing programs		Executes the source program as it has been input as soon as RUN is executed.	Generates intermediate code for the source program before executing it, so there is a short delay between when RUN is executed and the program is executed.
Memory	Allocation in the character variable area	Dynamic method.	The user can select between a static (fixed) method and a dynamic (free) method, depending on whether the OPTION LENGTH command is used.
	Defragmentation (reorganization of unused areas)	Executed while program is executing.	Executed when the RUN command is executed.
	Setting memory areas	Memory areas specified using the FRE command.	None
Communications	Communications buffers	Receive buffer: 256 bytes Transmit buffer: 256 bytes	Receive buffer: 512 bytes Transmit buffer: 512 bytes
	ER (DTR) code	Controlled by the system.	Also can be controlled by the DTR command in the program instead of the system.
	Indicator showing that data is waiting to transmit.	The ERR indicator is lit.	BASIC indicator flashes slowly.
	Limiting the output code when executing the PRINT command, depending on the device code specifications for the OPEN command.	Depending on the code, some codes are not output, and some are output with LF added.	The TERM command cannot be used for communications port #2. Otherwise, there is no limitation on outputs.
	DMA (Direct Memory Access) data transmission when sending data with the PRINT command.	Not available. (DMA transmission is not available. Depending on the software, send data and stop the BASIC program execution.)	Available. (DMA transmission can be enabled or disabled. While software data transmission is being performed, parallel hardware data transmission is possible by using a trigger from the software.
Exchanging data with the CPU Unit	Number of data words that can be exchanged using PC READ/PC WRITE (including @) in one cycle of the CPU Unit	20 words max.	20 words or 127 words max. (available for C200HX/HG/HE or CS1-series PCs only)
	Amount of data that can be exchanged using I/O allocation.	CPU-to-ASCII Unit: 8 bits ASCII-to-CPU Unit: 8 bits	CPU-to-ASCII Unit: 32 bits ASCII-to-CPU Unit: 16 bits
	Monitoring errors generated in the ASCII Unit from the CPU Unit.	Not available	Available
	Storing data in character-string variables for such commands as PC READ and PC WRITE.	Only the first 4 characters are read or written for each character variable.	Up to 255 characters can be read or written for each character variable.

Item		C200H-ASC02	C200H-ASC11/21/31
Flags/Control Bits	Software Restart Bit	Yes	No
	Read Bit and PC Interrupt Bit	Uses the Write Bit.	The Read Bit and the PC Interrupt Bit are different.
	ASCII Busy Flag	Changes status when the PC interrupt is generated.	Does not change when the PC interrupt is generated. Instead, the PC Interrupt Fail Flag changes if the PC interrupt fails.
Interrupts	Multiple interrupts	Yes. If an XXX ON command occurs, or another interrupt is generated while executing an interrupt subroutine, the executing interrupt subroutine is stopped, and the next interrupt subroutine is branched to.	No. If another interrupt is generated while executing an interrupt subroutine, the next subroutine is executed after the first interrupt subroutine execution is completed. The interrupt will, however, stop executing immediately when an ON ERROR interrupt is generated.
		Multiple interrupts are executed in the order that they were generated.	The next interrupt subroutine is executed according to order of priority.
	Executing interrupts	Even if a BASIC interrupt occurs while a command is being executed, it will not branch into the interrupt subroutine. Branching will only occur after the command has been executed.	When a BASIC interrupt occurs while a command is being executed, the interrupt subroutine will be executed, but only once.
Processing speed	Execution time depending on destination of GOSUB and GOTO commands.	Changes according to the distance from the current line to the destination.	Same time for all destinations.
Commands	RND command	Random number generator logic is different.	
	WAIT command	Timer does not stop while interrupt subroutine is being executed.	Timer stops while interrupt subroutine is being executed.
	PRINT USING command	Display will change depending on the USING format.	
Number range	Integer range	–32768 to 32767	–2147483648 to 2147483647
	Single-precision real number range	–1.70141E + 38 to 1.70141E + 38	$\pm 1.2 \times 10^{-38}$ to 3.4×10^{38}
	Double-precision real number range	–1.701411834604692D + 38 to 1.701411834604692D + 38	$\pm 2.2 \times 10^{-308}$ to 1.79×10^{308}
	Precision of arithmetic function results.	Single-precision	Single or double-precision
Saving to/reading from ROM		Uses the LOAD command to read from EEPROM.	Reads from /writes to the flash ROM and user memory (RAM) using the ROMLOAD and ROMSAVE commands.

Appendix C
PC Format

PC Format

When the ASCII Unit reads from and writes to the I/O memory data of the CPU Unit or the shared memory of the ASCII Unit itself using the following commands, it recognizes the data format of the I/O memory or shared memory. The ASCII Unit uses this data format when reading data from the I/O memory or the shared memory into the ASCII Unit variable or writing data from the ASCII Unit variable to the I/O memory or the shared memory. To allow such read-write operation, the data format must be specified. This specified data format is called the PC format.

Target Commands

For CPU Unit I/O Memory Data: PC READ/PC WRITE, PC READ@/ PC WRITE@, PC QREAD@/ PC QWRITE@ commands

For ASCII Unit Shared Memory Data: PC EPUT/ PC EGET commands

- In the A format, multiple PC words can correspond to one ASCII Unit variables, with a maximum of 127 words for one variable.
  For formats other than A, S, and where the variable is a character variable, there must be one variable corresponding to each word.
  In the A format, one variable corresponds to one format.
  In the S format, one array variable corresponds to one format.
  For formats other than A, S, and where the variable is a character variable, multiple variables can be allocated to one format.
- When “m” is omitted, one word is assumed.
  Words from 1 to 64 (1 to 127 for A format) and numerals from 1 to 255 can be handled at one time.
  Formats and subcommands can be specified as character variables.

Reading Data from CPU Unit I/O Memory or ASCII Unit Shared Memory to ASCII Unit Variable

Name	Format		Meaning	Example
I (BCD) format	mIn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 255 (see note 1) n = 1 to 4 The following must be true when variables are characters: m x n < 256	Data of m words is regarded as BCD data, and the data from the rightmost 1 to n digits of each word is stored in each variable.	2I3: Data of 2 words is regarded as BCD, and the data from the rightmost 1 to 3 digits of each word (bits 00 to 11) is stored in 2 variables.

Name	Format		Meaning	Example
H (hexadecimal) format	mHn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 255 (see note 1) n = 1 to 4 The following must be true when variables are characters: m x n < 256	Data of m words is regarded as hexadecimal data, and the data from the rightmost 1 to n digits of each word is stored in each variable.	2H3: Data of 2 words is regarded as hexadecimal, and the data from the rightmost 1 to 3 digits of each word (bits 00 to 11) is stored in 2 variables.
O (octal) format	mOn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 255 (see note 1) n = 1 to 4 The following must be true when variables are characters: m x n < 256	Data of m words is regarded as octal data, and the data from the rightmost 1 to n digits of each word is stored in each variable.	2O3: Data of 2 words is regarded as octal, and the data from the rightmost 1 to 3 digits of each word (bits 00 to 11) is stored in 2 variables.
B (bit) format	mBn	When variables are numerals: m = 1 to 255 n = 0 to 15 When variables are characters: m = 1 to 255 (see note 1) n = 1 to 15 The following must be true when variables are characters: No. of characters < 256	Data of m words is regarded as binary data, and the nth bit of data from each word only is converted to decimal data (other bits are regarded as 0) and stored in each numeric variable.	2B3: Data of 2 words is regarded as binary, and the 3rd bit of data from each word only is converted to decimal data (other bits are regarded as 0) and stored in each numeric variable.
A (ASCII) format	mAn	m = 1 to 255 (see note 2) m = 127 or less if n = 1 to 2	Data of m words is regarded as ASCII data, and the data specified as leftmost or rightmost in each n word stored in 1 character variable. n = 1: Rightmost byte n = 2: Leftmost byte n = 3: Leftmost and rightmost bytes	2A3: Data of 2 words is regarded as ASCII data, and the leftmost and rightmost bytes of each word (4 characters total) is stored in 1 character variable.

Name	Format		Meaning	Example
S (array variable) format	SmXn Note: X is any of I,H,O or B.	m = 1 to 255 n = same as specified format	Data of m words is regarded as data in the specified format (BCD, hexadecimal, octal, or bit), and the data from each word is stored in n array variables. Character string arrays cannot be used as variables.	S2I2: Data of 2 words is regarded as BCD data and data from each word is stored in 2 array variables.

Note 1. For versions up to V1.02, m = 1 to 64. The version is shown on the terminal startup display.
2. For versions up to V1.02, m = 1 to 127. The version is shown on the terminal startup display.

Reading Data from ASCII Unit Variable to CPU Unit I/O Memory or ASCII Unit Shared Memory

Name	Format		Meaning	Example
I (BCD) format	mIn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 64 n = 1 to 4	Data from the rightmost 1 to n digits of m variables is converted to BCD data, and saved to the rightmost 1 to n digits of each word for m words.	2I3: Data from the rightmost 1 to 3 digits of 2 words is converted to BCD data, and saved to the rightmost 1 to 3 digits of each word for 2 words.
H (hexadecimal) format	mHn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 64 n = 1 to 4	Data from the rightmost 1 to n digits of m variables is converted to hexadecimal data, and saved to rightmost 1 to n digits of each word for m words.	2H3: Data of the rightmost 1 to 3 digits of 2 words is converted to hexadecimal data, and stored in the rightmost 1 to 3 digits of each word for 2 words.
O (octal) format	mOn	When variables are numerals: m = 1 to 255 n = 1 to 4 When variables are characters: m = 1 to 64 n = 1 to 4	Data from the rightmost 1 to n digits of m variables is converted to octal data, and saved to the rightmost 1 to n digits of each word for m words.	2O3: Data of the rightmost 1 to 3 digits of 2 words is converted to octal data, and stored in the rightmost 1 to 3 digits of each word for 2 words.
B (bit) format	mBn	When variables are numerals: m = 1 to 255 n = 0 to 15	Each of m numeric variables is regarded as decimal, and converted to binary data, and the nth bit of data from each word only (other bits are regarded as 0) is stored to each word for m words.	2B3: Each of 2 numeric variables is regarded as decimal and converted to binary data, and the 3rd bit of data from each word (other bits are regarded as 0) only is stored in each of 2 words.

Name	Format		Meaning	Example
A (ASCII) format	mAn	m = 1 to 127 n = 1 to 3	Two ×m characters from the beginning of 1 character variable are converted into ASCII data (2 characters correspond to 1 word), and is stored in the leftmost/rightmost byte position in each word specified by n. n = 1: Rightmost byte (leftmost byte is 00_{hex}) n = 2: Leftmost byte (rightmost byte is 00_{hex}) n = 3: Leftmost and rightmost bytes	2A3: The first 4 characters in 1 character variable is converted to ASCII data, and stored to the leftmost and rightmost bytes of each word for 2 words.
S (array variable) format	SmXn Note: X is any of I,H,O or B.	m = 1 to 255 n = similar to each specified format.	Data of n array variables is converted to a data format specified by X (any of BCD, hexadecimal, octal, or bit) and stored to each of m words. Character string arrays cannot be used as variables.	S2I2: Data from 2 array variables is converted to hexadecimal and stored in each of 2 words.

Up to 255 words can be transferred at one time. For integer variables, however, only a maximum of 50 to 60 words can be transferred at one time. The following restrictions apply:

- A maximum of 255 characters can be written in one line.
- One variable must correspond to each word for integer variables.

Example: H format

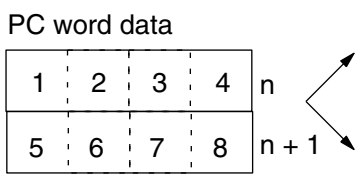
1000 PC READ "59H4"; A0, A1, A2, . . . , B0, B1, . . . ,

19 characters 236 characters max.

Up to 79 variables using 3 characters per variable (236 ÷ 3 = 79) can be written. Therefore, a maximum of 59 words can be transferred at one time.

PC READ Format Conversion Examples

I Type Format



Integer variables

```
PC READ "I1"; J    → J = 4
PC READ "I2"; J    → J = 34
PC READ "I3"; J    → J = 234
PC READ "I4"; J    → J = 1234
```

Character variables

```
PC READ "I1"; A$   → A$ = "4"
PC READ "I2"; A$   → A$ = "34"
PC READ "I3"; A$   → A$ = "234"
PC READ "I4"; A$   → A$ = "1234"
PC READ "2I4"; A$  → A$ = "12345678"
```

H Type Format

PC word data

8	9	A	B	n
C	D	E	F	n + 1

Integer variables

```
PC READ "H1"; J     → J = &HB       = 11
PC READ "H2"; J     → J = &HAB      = 171
PC READ "H3"; J     → J = &H9AB     = 2475
PC READ "H4"; J     → J = &H89AB    = –30293
```

Character variables

```
PC READ "H1"; A$   → A$ = "B"
PC READ "H2"; A$   → A$ = "AB"
PC READ "H3"; A$   → A$ = "9AB"
PC READ "H4"; A$   → A$ = "89AB"
PC READ "2H4"; A$ → A$ = "89ABCDEF"
```

O Type Format

PC word data

1	2	3	4	n
1	2	3	4	n + 1

Integer variables

```
PC READ  "O1"; J  → J = &4      = 4
PC READ  "O2"; J  → J = &34     = 28
PC READ  "O3"; J  → J = &234    = 156
PC READ  "O4"; J  → J = &1234   = 668
```

Character variables

```
PC READ  "O1"; A$   → A$ = "4"
PC READ  "O2"; A$   → A$ = "34"
PC READ  "O3"; A$   → A$ = "234"
PC READ  "O4"; A$   → A$ = "1234"
PC READ  "2O4"; A$  → A$ = "12341234"
```

B Type Format

PC word data

C	1	2	2
C	1	2	2

Integer variables

```
PC READ "B1"; J    → J = 2
PC READ "B2"; J    → J = 0
PC READ "B5"; J    → J = 32
PC READ "B14"; J → J = 16384
PC READ "B15"; J → J = -32768
```

Character variables

```
PC READ "B1"; A$    → A$ = "2"
PC READ "B2"; A$    → A$ = "0"
PC READ "B5"; A$    → A$ = "32"
PC READ "B14"; A$  → A$ = "16384"
PC READ "B15"; A$  → A$ = "–32768"
PC READ "2B15";A$ → A$ = "–32768-32768"
```

A Type Format

If the integer variable type does not match, an error will be generated.

PC word data

5	1	5	2
5	3	5	4

Character variables

PC READ "2A1"; A$ → A$ = "RT"
PC READ "2A2"; A$ → A$ = "QS"
PC READ "2A3"; A$ → A$ = "QRST"

Q: &H51
R: &H52
S: &H53
T: &H54

S Type Format

PC word data

0	1	2	3
4	5	6	7
8	9	0	1
2	3	4	5

Integer variables (In I type format)

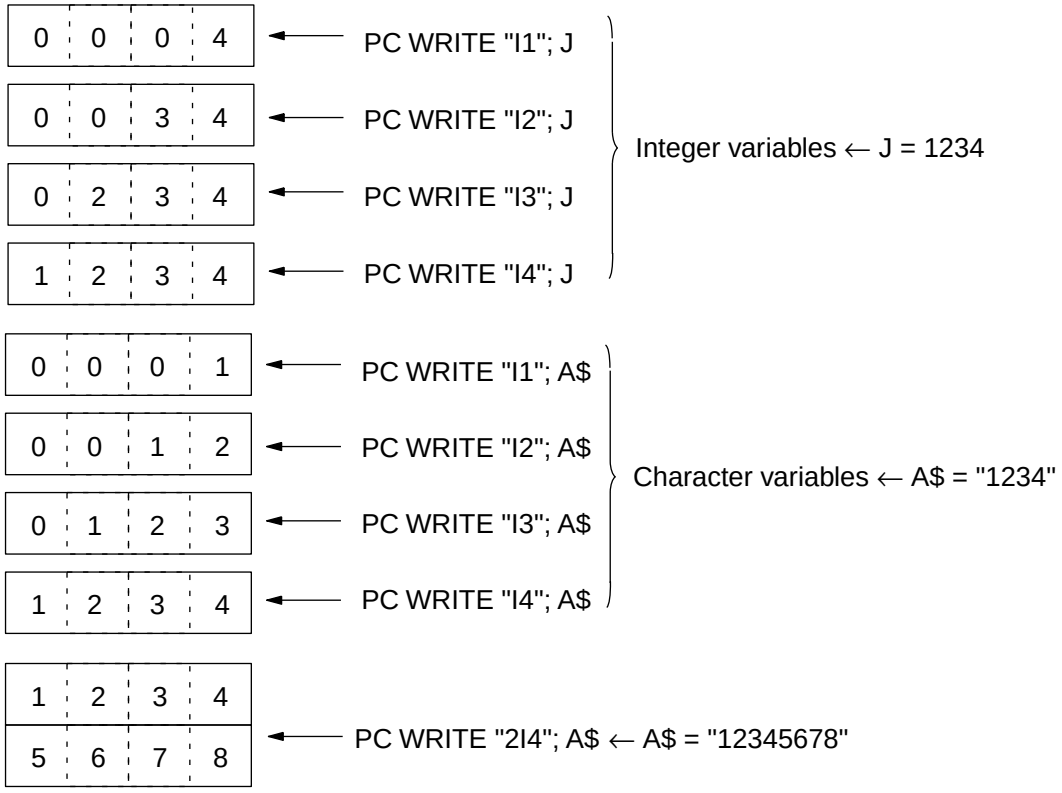
PC READ "S4I4"; A (1) → A (1) = 123
→ A (2) = 4567
→ A (3) = 8901
→ A (4) = 2345

Note: Character-string array variables cannot be used.

Examples of PC WRITE Format Conversions

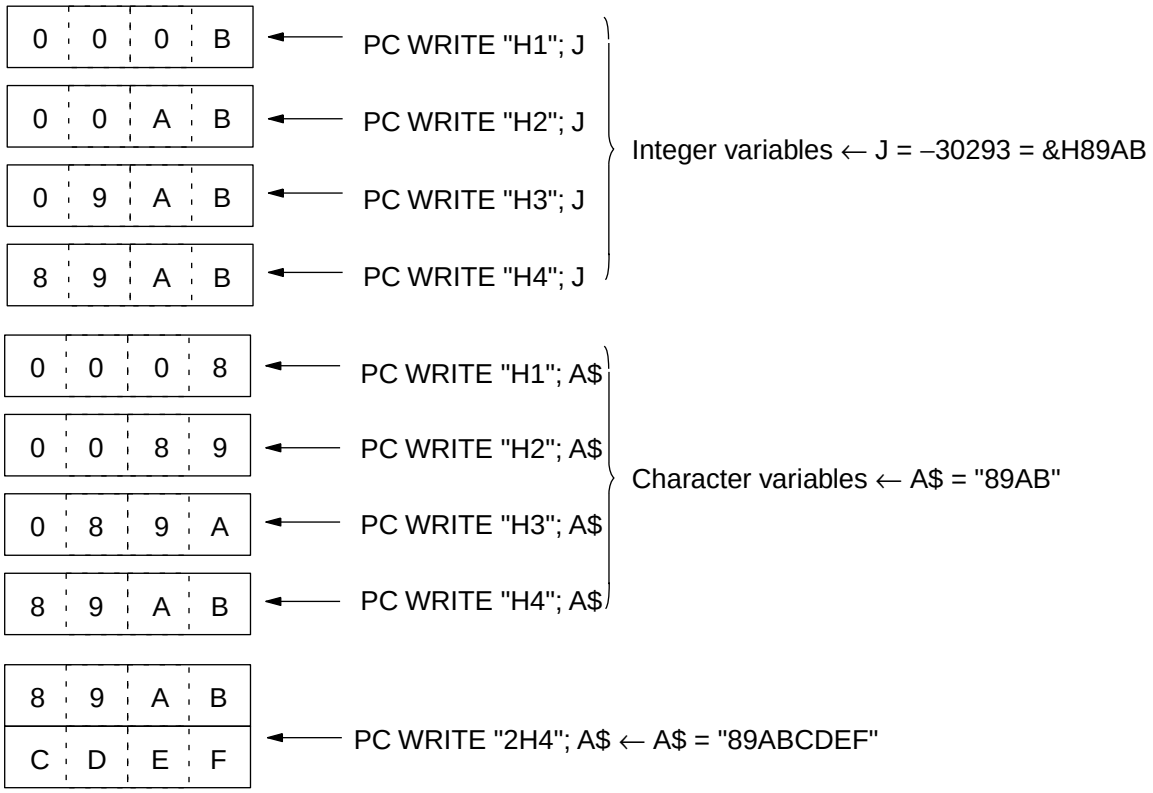
I Type Format

PC word data



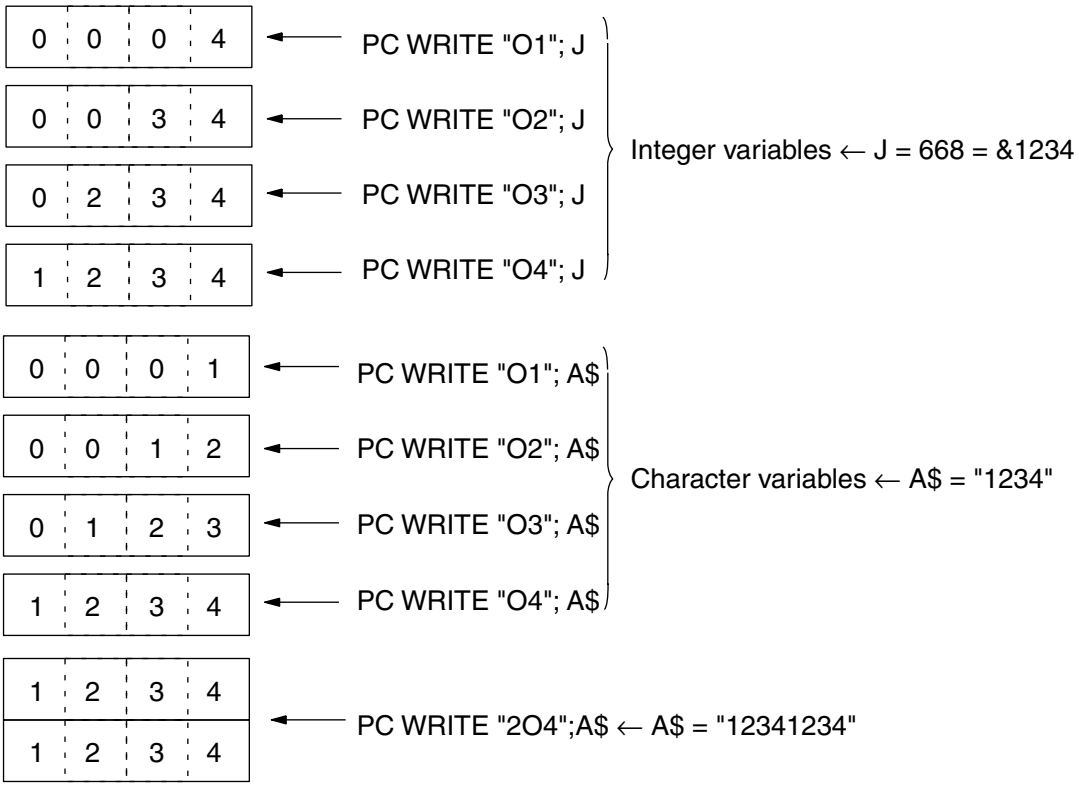
H Type Format

PC word data



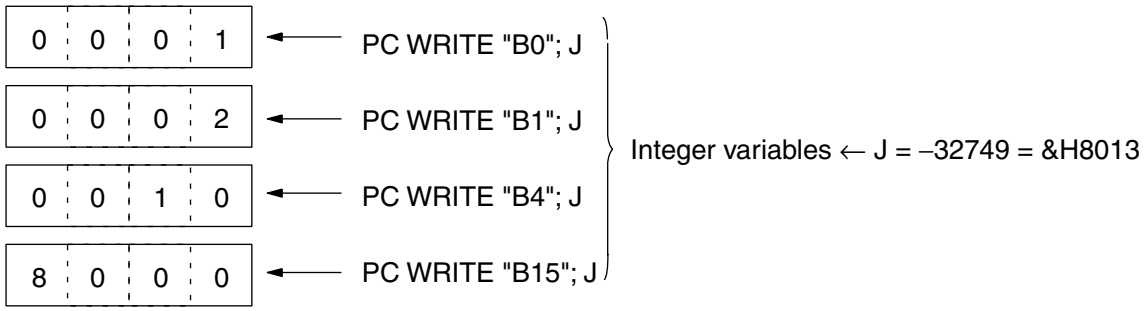
O Type Format

PC word data



B Type Format

PC word data



If the character variable type does not match an error will be generated.

Relationship between Format and Variables

For Bn, if n bit is ON when the variables will be displayed in binary numbers, turn ON PC bit n and turn OFF other bits. If the variable n bit is turned OFF, turn OFF all the PC bits.

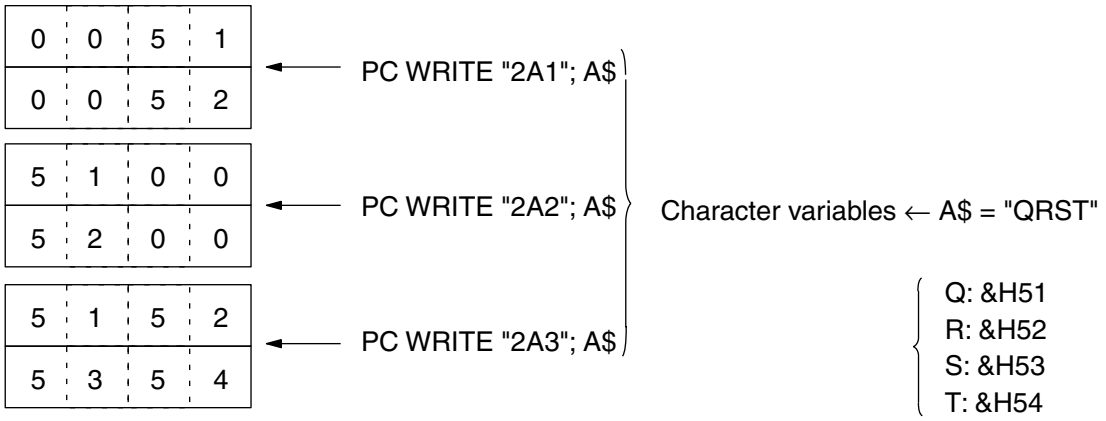
Example: –32749 = &H8013 = 1000 0000 0001 0011

B0, B1, B4, B15:	ON
B2, B3, B5 to B14:	OFF

A Type Format

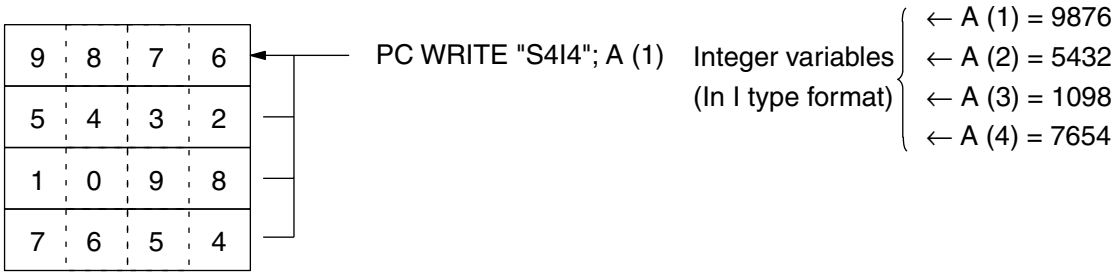
If the integer variable type does not match, an error will be generated.

PC word data



S Type Format

PC word data



Note 1. Numeric variables and character variables can be used together for PC READ and PC WRITE commands if the character variable is at the end of the format as follows:
Example: PC READ “@D,0,2,H4,H4”; B, A$
If the command format is written as PC READ “@D,0,2,H4,H4”; A$, B, a FORMAT ERROR will be generated. The above combination cannot be used if array variables have been used as variables for PC WRITE.

2. Character-string array variables cannot be used with the S-type format. Use the following format to read or write character data.
Example: PC READ “@D,0,2,10,10H4”; A$

3. Only one character-string variable can be used in the variable list, so multiple strings are not possible.

4. If using “mA1, mA2” when writing ASCII variables in A–type format to I/O memory in the CPU Unit, provide 2 x m characters in the character variable.

Example 1: PC WRITE "@D,0,4,4A,4A1";("1234"+"1234")
Dummy data

Example 2: A$= 1234"
PC WRITE "@D,0,4A,4A1";(A$+A$)

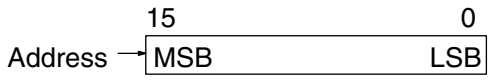
5. Negative values cannot be written using the I–type format for PC WRITE. A format error will occur.

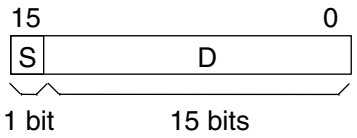
Appendix D

Formats for Storing Variables in Memory

Variables are stored to memory in the following ways, depending on the variable type.

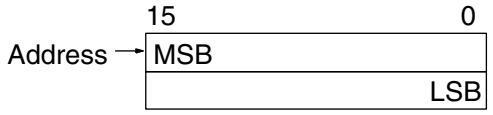
- **Short Integer Variables**

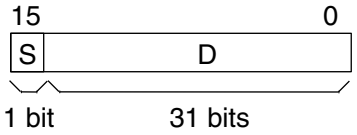




S: Sign (0: positive, 1: negative)
D: Numeric value

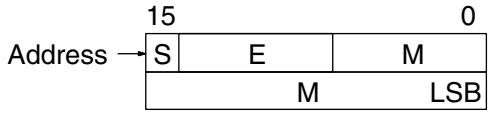
- **Long Integer Variables**

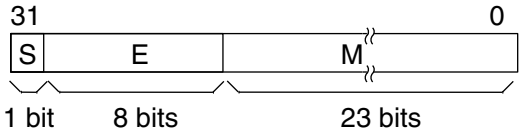




S: Sign (0: positive, 1: negative)
D: Numeric value

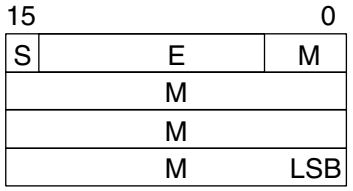
- **Single-precision Real Number Variables**

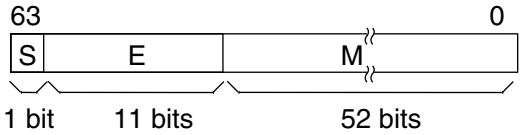




S: Sign (0: positive, 1: negative)
E: Exponent (offset: 127)
M: Mantissa

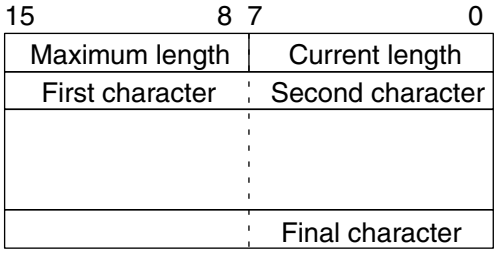
- **Double-precision Real Number Variable**





S: Sign (0: positive, 1: negative)
E: Exponent (offset: 1023)
M: Mantissa

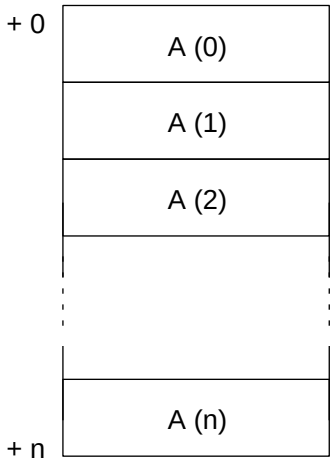
- **Character String Variable**



- Add empty bytes and adjust so that there will be an even number of memory bytes for each variable.
- The character data section not being used beyond the current length contains undefined data.

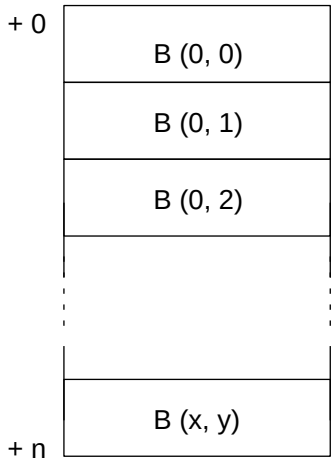
- **Array Variables**

For one-dimensional arrays



- As an element of the array, each variable is stored in memory in order.

For two-dimensional arrays



- For multi-dimensional arrays, the rightmost subscript is stored in memory, in the order of alteration.

VARPTR (Variable Name)

Function

Assigns the memory address where the variable value is stored.

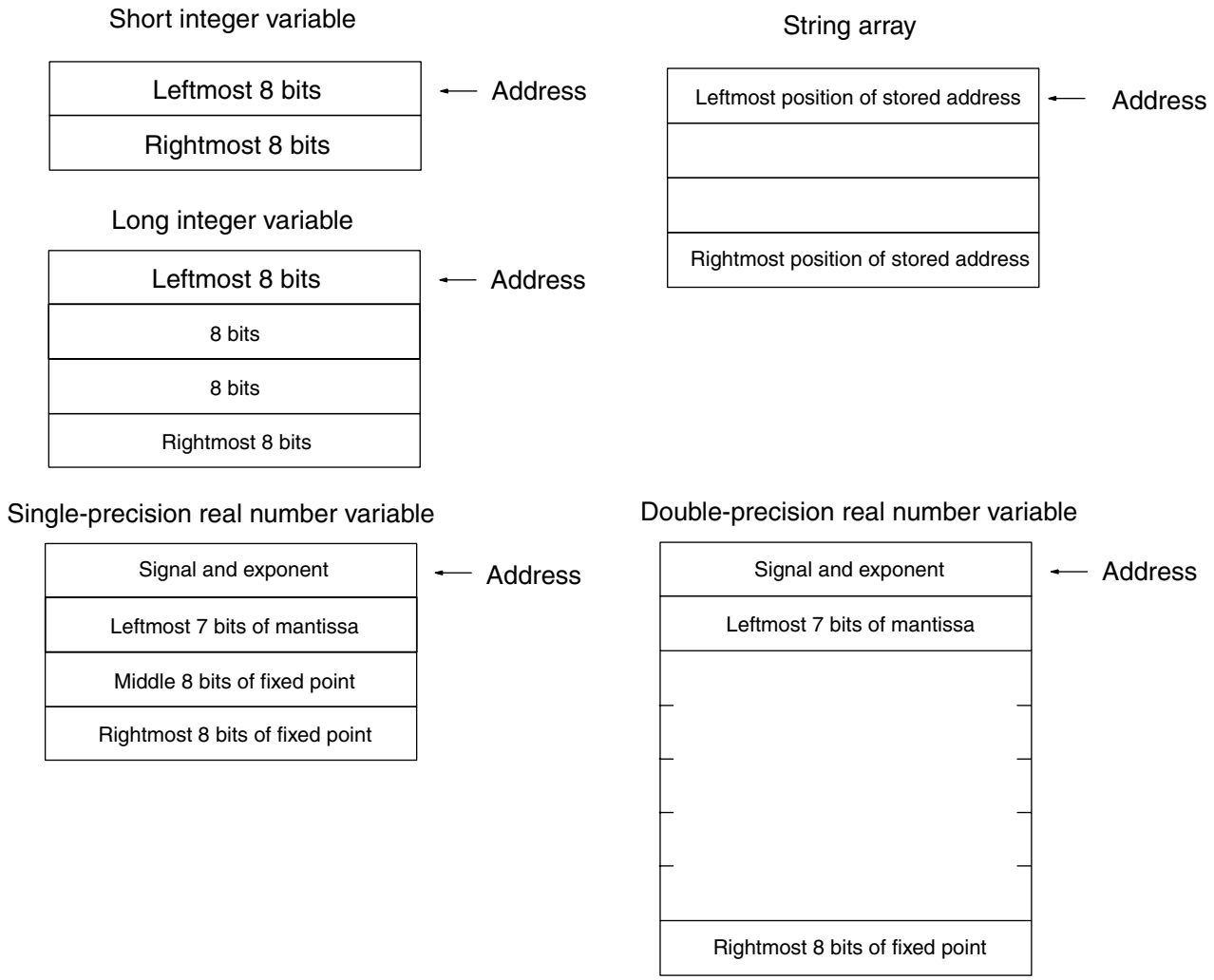
Notation Example

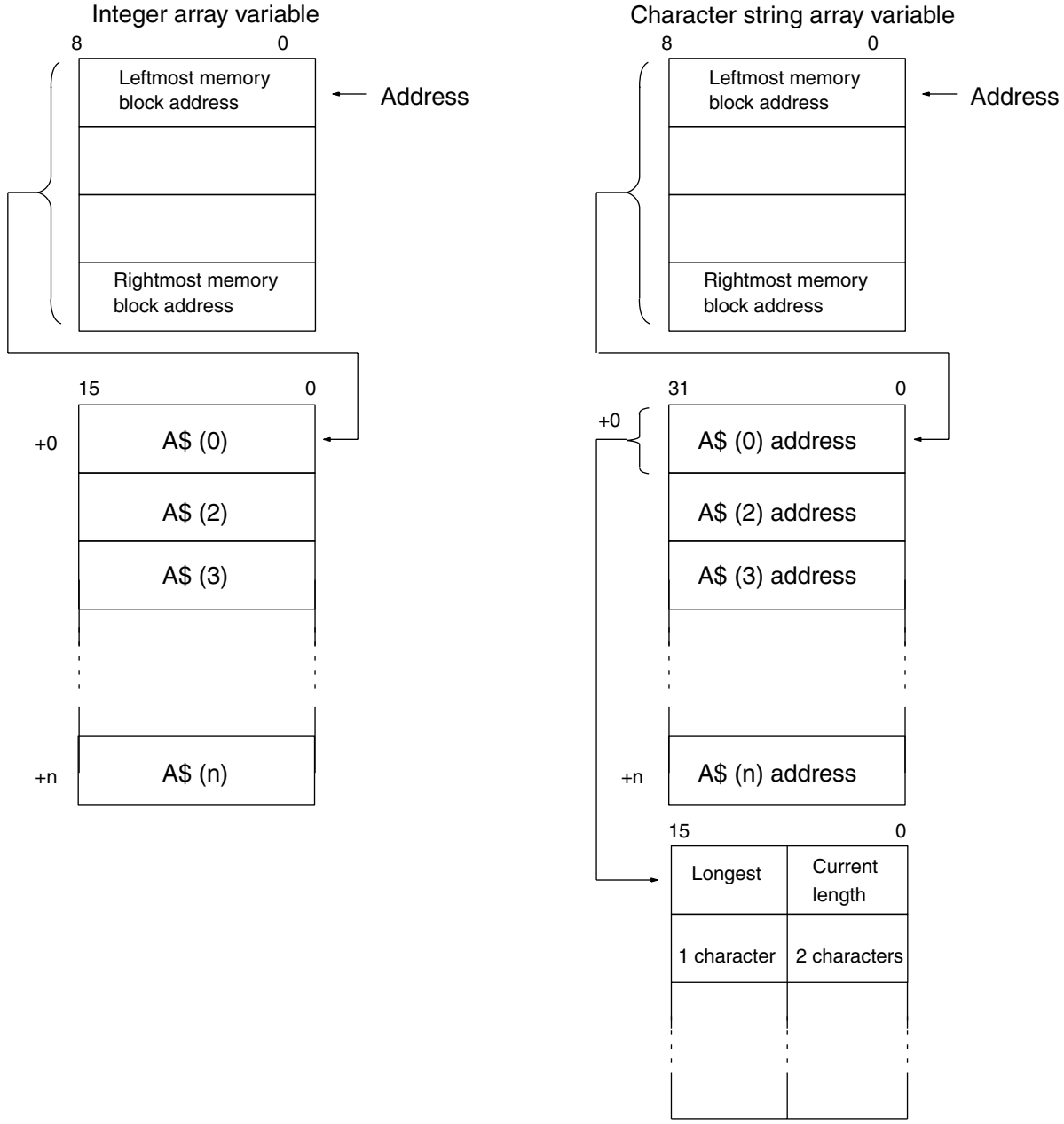
B& = VARPTR (A)

Details

- Assigns the memory address to the area where variable data specified for the variable name is stored.
- Shows the address of the pointer to where character data is stored for character variables.

Relationship between Variable Type and VARPTR Address




Integer array variable
8
0
Leftmost memory block address
Address
Rightmost memory block address
15
0
+0
A$ (0)
A$ (2)
A$ (3)
+n
A$ (n)
Character string array variable
8
0
Leftmost memory block address
Address
Rightmost memory block address
31
0
+0
A$ (0) address
A$ (2) address
A$ (3) address
+n
A$ (n) address
15
0
Longest
Current length
1 character
2 characters

Appendix E

Command Execution Time Samples

Assignments

Assignment	Execution time (ms)
A=1	0.19
A%=1	0.14
A=A + 1	0.41
A=A – 1	0.41
A=A*1	0.39
A=A/1	0.41
A=A\1	0.36
A=A MOD 2	0.35
A=A∧ 2	0.52
A%=NOT A%	0.20
A%=A% AND 1	0.28
A%=A% OR 1	0.28
A%=A% XOR 1	0.28
A%=A% EQV 1	0.28
A%=A% IMP 1	0.28

Statements

Statement	Execution time (ms)
@IF A=0 THEN B=0 ELSE B=1 ENDIF (true and false)	0.45
BITON C%, 0 / BITOFF C%, 0	0.12
CLOSE	0.60
FOR I=1 TO 10000: NEXT I (for one loop)	0.14
FOR I%=1 TO 10000: NEXT I% (for one loop)	0.07
GOSUB *TEST* TEST: RETURN	0.13
GOTO	0.03
IF A=0 THEN B=0 ELSE B=1 (true and false)	0.45
INPUT #2, A$ 200 characters	Approx. 20.0
LET	0.19
LINE INPUT #2, A$ 200 characters	Approx. 30.0
OPEN #2, “COMU:”	1.06
PC EGET #0, 90, “S90I4”; A (0)	7.8
PC EPUT #0, 90, “S90I4”; A (0)	8.4
PC GET I, J	0.34
PC PUT I	0.17
PC QREAD “@D,0,10,10I4”; A,B,C,D,E,F,G,H,I,J (See note 2.)	11.5
PC QWRITE “@D,0,10,10I4”; A,B,C,D,E,F,G,H,I,J (See note 2.)	8.8
PC READ “I4”; A(See note 1.)	7.5
PC READ “5I4”; A,B,C,D,E(See note 1.)	7.6
PC READ “10I4”; A,B,C,D,E,F,G,H,I,J (See note 1.)	7.7
PC READ “63H4”; A$ (See note 1.)	41.9

Statement	Execution time (ms)
PC WRITE “I4”; A (See note 1.)	3.9
PC WRITE “5I4”; A,B,C,D,E (See note 1.)	4.2
PC WRITE “10I4”; A,B,C,D,E,F,G,H,I,J (See note 1.)	7.7
PC WRITE “63H4”; A$ (See note 1.)	20.7
PC READ “@D,0,1,I4” A (See note 1.)	5.8
PC READ “@D,0,5,5I4”; A,B,C,D,E (See note 1.)	5.9
PC READ “@D,0,10,10I4”; A,B,C,D,E,F,G,H,I,J (See note 1.)	7.8
PC READ “@D,0,63,63H4”; A$ (See note 1.)	32.7
PC READ “@D,0,63,63H4”; A$ (α mode) (See note 1.)	30.8
PC WRITE “@D,0,1,I4” A (See note 1.)	4.0
PC WRITE “@D,0,5,5I4”; A,B,C,D,E (See note 1.)	4.0
PC WRITE “@D,0,10,10I4”; A,B,C,D,E,F,G,H,I,J (See note 1.)	7.7
PC WRITE “@D,0,63,63H4”; A$ (See note 1.)	20.7
PC WRITE “@D,0,63,63H4”; A$ (α mode) (See note 1.)	9.0
PRINT A$ (9,600 bps, 255 characters)	Normal mode: Approx. 300 DMA mode: Approx. 5
REM	0.03
WHILE A < 100 : A = A + 1 : WEND (for one loop)	0.67

Note 1. The execution time includes the time required for handshaking with the PC. Scan time is usually 2 ms (approx.).
Transfer of less than 20 words:
2 × PC scan times max.
@: 1 × PC scan time max.
Transfer of more than 20 words (C200H Mode):
INT ((No. of transfer words – 1)/20) + 2 × PC scan times max.
@: INT ((No. of transfer words – 1)/20) + 1 × PC scan time max.
Transfer of more than 127 words (C200H-HX/HG/HEPC Mode):
INT ((No. of transfer words – 1)/127) + 2 × PC scan times max.
@: INT ((No. of transfer words – 1)/127) + 1 × PC scan time max.

2. The execution time includes the time required for handshaking (IORD or IOWR execution) with the PC.

Functions

Function	Execution time (ms)
A=ABS(–1.5)	0.22
A=ACOS(0.5)	1.72
A%=ASC(”A”)	0.16
A=ASIN(0.5)	1.7
A=ATN(0.5)	0.74
B#=CDBL(1)	4.7
D$=CHR$(&H41)	0.45
C%=CINT(0.5)	0.24
A=COS(0.5)	0.92
A=CSNG(0.1#)	0.22
A=EXP(5)	1.05

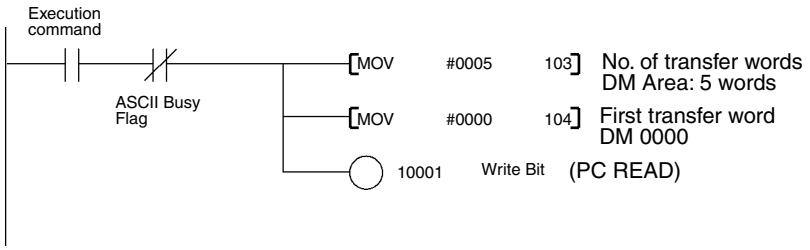
Function	Execution time (ms)
A$=STRING$(250,”A”) B$=FCS(A$,1) Horizontal parity	1.0
C%=FIX(1.1)	0.26
D$=HEX$(1)	0.5
INPUT$(100,#2)	3.0
C%=INT(1.1)	0.25
D$=LEFT$(”ASCII”,2)	0.51
A%=LEN(”ASCII”)	0.17
A%=LOC(2)	0.22
A=LOG(5)	1.0
D$=MID$(”ASCII”,1,1)	0.62
D$=OCT$(&051)	0.54
D$=RIGHT$(”ASCII”,2)	0.5
A=SIN(0.5)	0.73
A=SQR(2)	0.66
D$=STR$(1)	0.64
A=TAN(3)	1.01
A%=VAL(”A”)	0.2

Appendix F

Sample Programs

1) Transferring PC Data from the ASCII Unit

Ladder Program



Basic Program

100 PC WRITE “5I4”; A,B,C,D,E

The 5 words of data starting with the first transfer word DM 0000 are read as BCD data and stored (read) in the 5 variables A, B, C, D, E.

2) Transferring PC Data from the PC to the ASCII Unit

Ladder Program

A ladder program is not required.

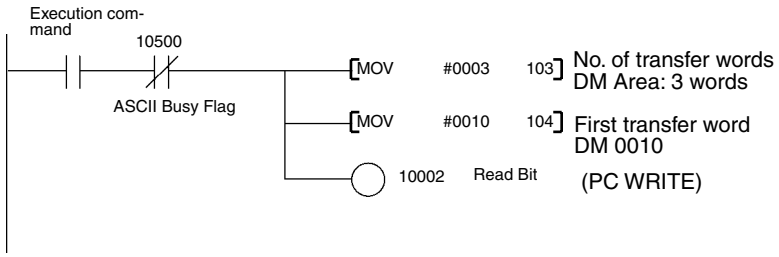
Basic Program

100 PC READ “@D,0,5,5I4”; A,B,C,D,E

Five words of data starting with DM 0000 from the CPU Unit are read as BCD data and each fourth digit (16-bit) of data are stored (read) in the 5 variables A, B, C, D, E.

3) Transferring ASCII Unit Data from the ASCII Unit to the PC

Ladder Program



Basic Program

100 PC WRITE “3I4”; A,B,C

The 3 variables A, B, and C (4-digit data) is converted to BCD data and transferred (written) to the corresponding 4 digits (16 bits) of 3 words in the CPU Unit, beginning with the above first transfer word (DM 0010).

4) Transferring ASCII Unit Data from the ASCII Unit to the PC

Ladder Program

A ladder program is not required.

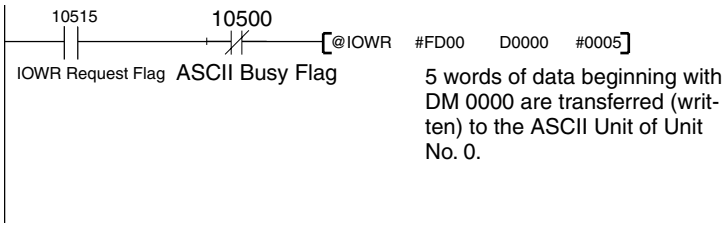
Basic Program

100 PC WRITE “@D,10,3,3I4”; A,B,C

The 3 variables A, B, and C (4-digit data) is converted to BCD data and transferred (written) to the corresponding 4 digits (16 bits) of 3 words in the CPU Unit, beginning with the first transfer word (DM 0010).

5) Transferring Data Asynchronously from PC to ASCII Unit: C200HX/HG/HE PCs Only

Ladder Program



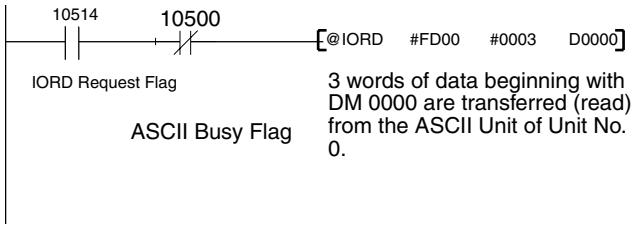
Basic Program

100 PC QREAD “@D,0,5,5I4”; A,B,C,D,E

The 5 words of data that were transferred using the IOWR(FD00) command are converted to BCD and stored in the 5 variables A, B, C, D, E as 4-digit (16-bit) data.

6) Transferring Data Asynchronously from ASCII Unit to PC: C200HX/HG/HE PCs Only

Ladder Program



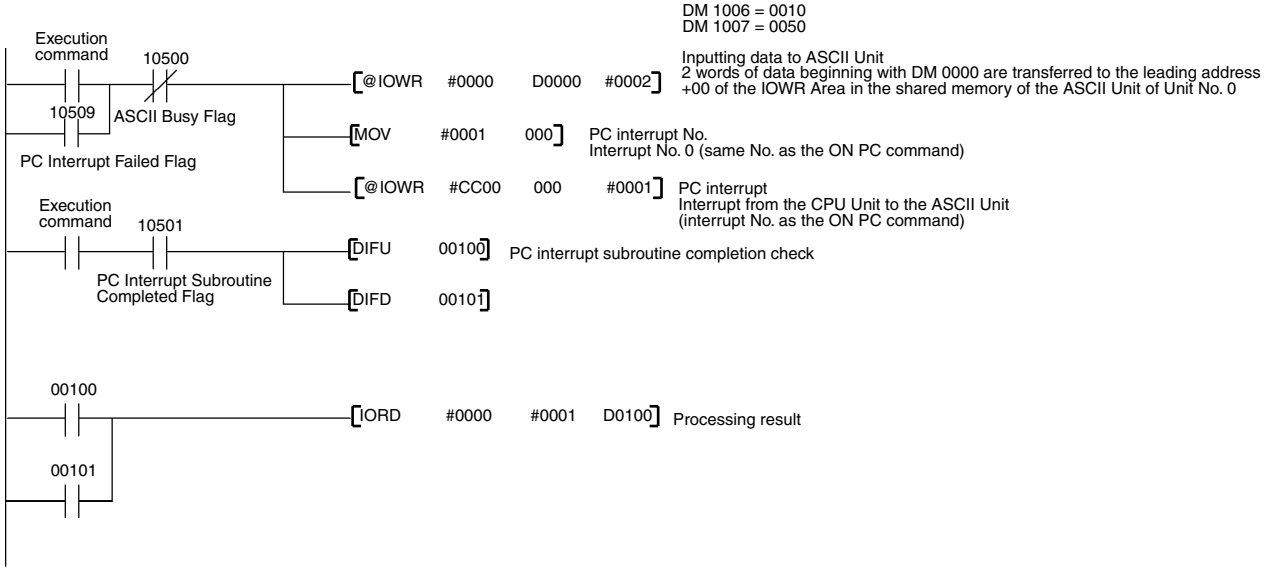
Basic Program

100 PC QWRITE “@D,0,3,3I4”; A,B,C

The 3 variables A, B, and C (4-digit data) is converted to BCD data and is ready for transferring to the 3 words in the CPU Unit, beginning with DM 0000.

ASCII Unit Processing as PC Coprocessor

Ladder Program



Basic Program

```
100 ON PC 1 GOSUB *PINT
110 PC ON
:
:
200 *PINT
210 PC EGET #0,2,"2I4"; A,B
220 C%=A+B
230 PC EPUT #0,1,"I4"; C%
240 RETURN
```

Appendix G

Wiring RS-232C or RS-422A/485 Cable Connectors

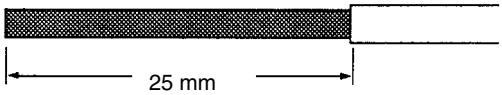
Cable Processing (End connected to FG)

See the diagrams for the lengths required in each step.

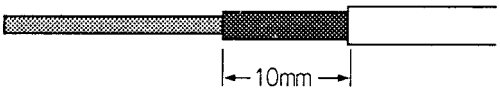
1,2,3...

1. Cut the cable to the required length.

2. Peel the sheath using a razor blade without damaging the shield weaving.



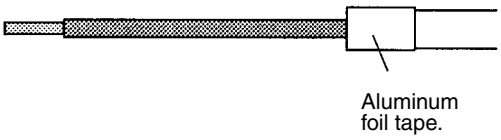
3. Remove the shield using scissors.



4. Peel the core wire of each wire using a stripper.

5. Fold back the shield wire.

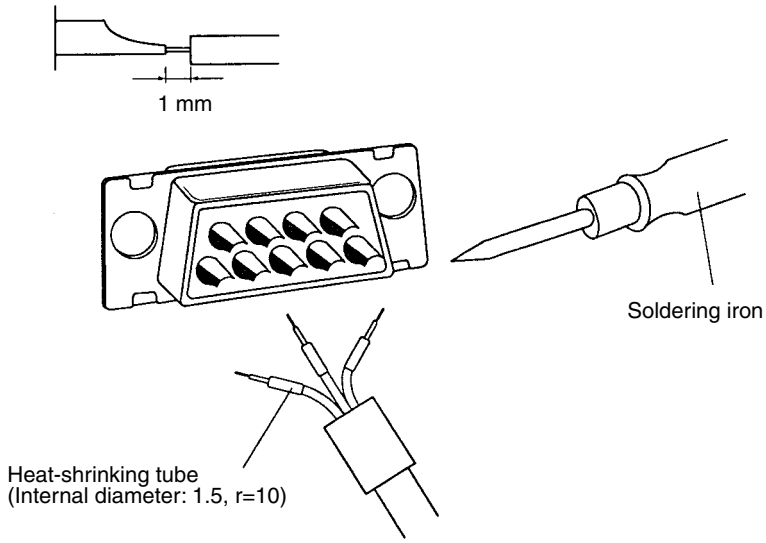
6. Wrap aluminum foil tape on top of the folded shield.



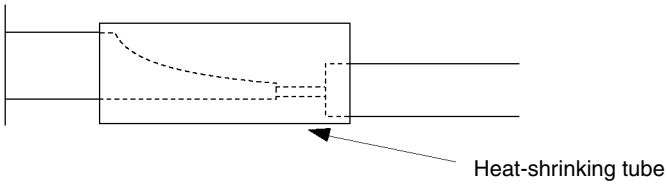
Soldering

Solder as described next.

1. Place a heat-shrinking tube around each wire.
2. Presolder each wire and to its connector pin.
3. Solder each wire firmly in place.

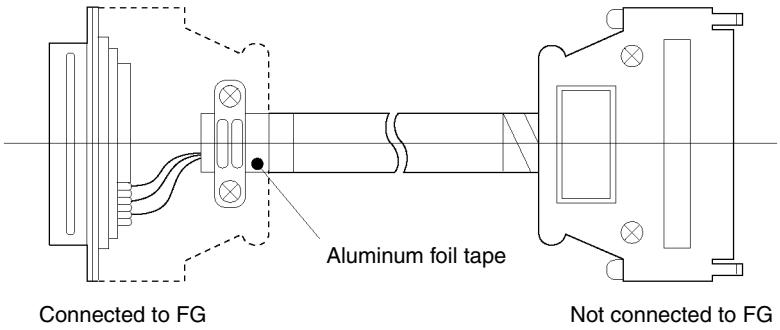


4. Move the heat-shrinking tube to the soldered section and shrink the tube by heating it.



Assembling Hood

Assemble the connector hood as shown below.



Appendix H

Reserved Words

Reserved word				
@IF	GO	RENUM	EQV	INTRS
ALARM	HELP	RESET	IMP	INTRR
ALL	IF	RESTORE	MOD	LEFT$
AUTO	INPUT	RESUME	NOT	LEN
BASE	KEY	RETURN	OR	LOC
BITOFF	LENGTH	ROMLOAD	XOR	LOG
BITON	LET		ABS	MID$
BRKPT	LIB	ROMSAVE	ACOS	MODEL
CLEAR	LINE	ROMVERIFY	ASC	OCT$
CLOSE	LIST	RTS	ASIN	PEEK
CLS	LLIST	RUN	ATN	RIGHT$
COM	LOAD	SAVE	CDBL	RND
CONT	LPRINT	SET	CHR$	SEARCH
DATA	MAP	STEP	CINT	SGN
DEF	NEW	STOP	CLNG	SIN
DEFDBL	NEXT	SUB	COS	SPACE$
DEFINT	OFF	SYSBACK	CSNG	SPC
DEFLNG	ON	SYSLOAD	CTS	SQR
DEFSNG	OPEN	SYSMOVE	DATE$	SSTACK
DEFSTR	OPTION	THEN	DAY	STR$
DEL	PC	TIMER	DMA	STRING$
DIM	PGEN	TO	DSR	TAB
DTR	PINF	TROFF	EOF	TAN
EDIT	PMEM	TRON	ERL	TIME$
EGET	PNAME	TRACE	ERR	TIME
ELSE	POKE	USING	EXP	STRING
END	PRG	VAL	FCS	STACK
ENDIF	PRINT	VARPTR	FIX	WDT
EPUT	PUT	WAIT	FRE	BYTE
ERASE	PWORD	WATCH	HEX$	CODE
ERC	QREAD	WEND	INKEY$	HEAD
ERROR	QWRITE	WHILE	INPUT$	TERM
FN	RANDOM	WRITE	INSTR	
FOR	READ		INT	
GET	REM	AND	INTRB	

Index

Numerics

2-wire/4-wire switch
(WIRE), 21

A

arithmetic expressions, 127
ASCII communications
with Peripheral Devices, 2
asynchronous data exchanges, 236
auto start (5A)
and START/STOP switch, 20
Automatic transfer setting, 40

B

Bar Code Reader
connecting, 26
BASIC
commands, 121
functions, 121
statements, 121
BASIC commands
execution times, 279
listed alphabetically, 129
BASIC errors, 243
BASIC functions
user-defined functions, 231
BASIC programs
continuing, 120
debugging, 232
editing, 12, 112
inputting programs, 114
processing examples, 61
programming, 14
reading from flash ROM, 116
sample programs, 283
saving to flash ROM, 116
starting, 120
starting and stopping, 20
stopping, 120
transferring, 115, 117
battery
replacement, 256
bit processing, 62
breakpoint function (BRKPT), 232
BRKPT command, 232

C

C200H-ASC02
comparison of functions and operation, 263
cable wiring, 287
cables
processing, 287
soldering, 288
character constant
definition, 123
character expressions, 127
character strings
processing, 62
character variables
allocating space, 118
clearing error messages, 241
clearing errors
with the IOWR instruction, 255
Command Mode, 121
commands
BASIC commands listed alphabetically, 129
execution times for BASIC commands, 279
comments
inputting in BASIC programs, 122
communications
communications interrupts, 64
connections, 24
multiport communications, 2
with Peripheral Devices, 2, 4
communications cables
wiring, 287
communications errors, 59
Communications interrupts, 64
communications port
closing, 53
opening, 44
connections, 24
connectors
hood, 288
constants
in BASIC programs, 123
control bits
Error Code Read Bit, 33
PC Interrupt Bit, 33, 90
Read Bit, 33
Special I/O Unit Restart Bits, 253
Sync Time Set Bit, 33
Write Bit, 33
CPU Unit
exchanging data, 69
CPU Units
compatible CPU Units, 5

D

data exchange

applications, 235
ASCII Unit to CPU Unit, 239
asynchronous processing, 236
bit data exchanges, 240
from CPU Unit through shared memory, 85
high-speed, 237
high-volume, 239
receiving data, 51
selecting data exchange method, 74
sending data, 47
timing charts, 87
to CPU Unit through shared memory, 86
with external devices, 43
with PC EGET/PC EPUT commands, 80
with PC PUT/PC GET commands, 79
with PC QREAD/PC QWRITE commands, 82
with PC READ/PC WRITE commands, 75
with PC READ@/PC WRITE@ commands, 78
with the CPU Unit, 69
debugging BASIC programs, 232
device symbol, 46
Digital Operator
connecting, 27
DM Area allocations, 38
DM setting errors, 247
DMA data transmission, 48

E

EDIT command
precautions, 259
ERC command, 59
ERR function, 60
Error Code and Type, 37
error codes
errors listed by error code, 243
error indicators
CPU error indicators, 252
error log
reading and clearing, 254
errors
BASIC errors, 243
clearing error messages, 241
clearing with IOWR instruction, 101
clearing with the IOWR instruction, 255
communications errors, 59
DM setting errors, 247
execution errors, 246
operational errors, 244
PC Interface errors, 247
execution errors, 246
execution times
for BASIC commands, 279
expressions
definition, 127
external devices
connecting, 14
exchanging data, 43

F–H

flags
ASCII Busy Flag, 36
BASIC RUN Flag, 36
Battery Error Flag, 36
Interrupt Fail Flag, 95
IORD Request Flag, 36
IOWR Request Flag, 36
PC Interrupt Fail Flag, 36
PC Interrupt Subroutine Completed Flag, 36
Port 1 Error Flag, 36
Port 2 Error Flag, 36
Special I/O Unit error flags, 253
System BASIC Error Flag, 36
Terminal Port Error Flag, 36
floating point constants
definition, 124
high-speed data exchanges, 237
high-volume data exchanges, 239

I

ID Controller
connecting, 27
indicators, 19
troubleshooting errors, 248
INPUT# command, 51
INPUT$ function, 52
inputs
from the ASCII Unit to the CPU Unit, 35
Input Data, 37
installation, 22
integer constants
definition, 123
internal configuration, 5
internal layout, 7
interrupt functions, 64
interrupt failure, 95
interrupt priority, 65
interrupts
sending interrupts from the CPU Unit, 72
IORD Area
first word setting, 41
IORD instruction
reading ASCII Unit variables, 102
reading from shared memory, 104

IORD/IOWR Area transfer setting, 40
IORD/IOWR Areas, 39
IOWR Area
  first word setting, 41
IOWR instruction
  clearing errors in ASCII Unit, 101
  sending interrupts, 100
  writing ASCII Unit variables, 96
  writing to shared memory, 97
IOWR/IORD instructions
  specifications, 95
IR Area allocations, 30

L

labels
  definition, 122
LED indicators, 19
  troubleshooting errors, 248
line number
  definition, 121
logical expressions, 128
loop processing, 67

M–N

machine language programs, 257
maintenance, 255
manual start (00)
  and START/STOP switch, 20
modes
  command execution modes, 121
multiport communications, 2
nomenclature, 18

O

ON ERROR command, 59
OPEN# command, 44
operation
  basic operating procedures, 12
operational errors, 244
operators, 127
OPTION LENGTH command, 118
outputs
  from the CPU Unit to the ASCII Unit, 32
  Output Data, 33

P

PC format, 265
PC Interface errors, 247
Peripheral Device
  general connections, 26
  specifying with device symbol, 46
Peripheral Devices
  ASCII communications with, 2
personal computer
  use as a terminal, 26
Port #1 baud rate setting, 41
Port #1 DMA transmission setting, 41
Port #2 baud rate setting, 41
Port #2 DMA transmission setting, 41
precautions
  for Terminal operations, 259
  operating precautions, 257
  when changing from C200H-ASC02, 257
  with PC READ/PC WRITE, 258
PRINT# command, 47
program memory
  dealing with lost/damaged memory, 251
Program Mode, 121
program number setting, 40

R

receive buffer processing, 62
receiving data, 51
relational expressions, 128
Restart Bits
  Special I/O Unit Restart Bits, 253
RS-232C connections, 24
RS-422A/485 connections, 25

S

sample programs, 283
sending data, 47
settings
  in the Setup Area, 40
Setup Area, 39
soldering, 288
Special I/O Unit Area
  word allocation, 30, 31
specifications, 7
  for IORD and IOWR, 95
  functional and performance, 7
  general, 7
START/STOP switch, 20
Startup mode setting, 40
statements
  definition, 122

STEP command, 233
step function (STEP), 233
SYSMAC BUS Slave Racks
  restrictions on, 5
system configuration, 3

T

Terminal
  transferring BASIC programs, 117
terminal
  using personal computer as a terminal, 26
Terminal emulation setting, 41
Terminal port #3 baud rate setting, 41
Terminating Resistance switch
  (TERM), 21
time processing, 63
timing charts
  for data exchanges, 87
  for transmission control signals, 53
TRACE command, 234
trace function (TRON and TRACE), 234
transmission control signals
  controlling, 46
  timing charts, 53
  when receiving data, 51
  when sending data, 48
TRON command, 234
troubleshooting, 248
troubleshooting BASIC programs, 232
troubleshooting error messages, 241
troubleshooting flowchart, 250

U

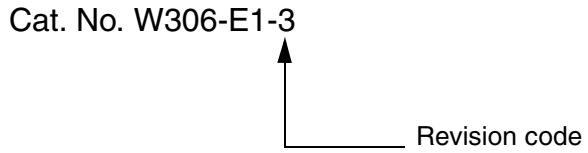
unit number switch
  (MACHINE No.), 21
user memory
  DM Area allocations, 38
  formatting, 21
  IR Area allocations, 30
User Memory Default Switch, 21

V–W

variable arrays, 125
variable monitor function (WATCH), 233
variables, 125
  storage format, 275
WATCH command, 233
wiring communications cables, 287

Revision History

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.



The following table outlines the changes made to the manual during each revision. Page numbers refer to the previous version.

Revision code	Date	Revised content
1	September 1998	Original production
2	July 1999	Additions made for CS1 Series.
3	June 2000	**Pages 5 and 23:** C200H-CPU02-E added to the table. **Page 20:** C200H-CPU01-E/02-E/03-E added to the table. **Page 37:** EM Area bank number added to word n+8. **Page 109:** Section *6-6-1 Using the Sync Time Set Bit* added. **Page 114:** Description on stopping an open program added to the end of *Connecting the Terminal*. **Page 115:** Description on T/R indicator added to the end of step 5. **Page 116:** Cautionary information on power supply changed. **Pages 191 and 193:** Table for C200H/HS/HE/HG/HX changed to one for C200HE/HG/HX. **Page 216:** Information added to the end of *Remarks.* **Page 233:** Description on DEVICE UNAVAILABLE IN PORT X error added to *Trace.* **Page 254:** Precautionary note on restarting added to *Special I/O Unit Restart Bits.* **Page 260:** Description on program startup time and power interruptions/restart added to *Executing BASIC Interrupts during Command Execution.* **Page 274:** Notes 4 and 5 added. **Page 288:** List of reserved words added as Appendix H.