

To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: <http://www.renesas.com>

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (<http://www.renesas.com>)

Send any inquiries to <http://www.renesas.com/inquiry>.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
 - “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
 - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
 - “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

USER'S MANUAL

Phase-out/Discontinued

IE-17K

**CLICE
VERSION 1.6**

USER'S MANUAL

NEC

Phase-out/Discontinued

IE-17K

**CLICE
VERSION 1.6**

Phase-out/Discontinued

SIMPLEHOST is a trademark of NEC Corporation.

Windows is a trademark of MicroSoft Corporation.

PC/AT is a trademark of IBM Corporation.

The information in this document is subject to change without notice.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.

NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or of others.

Phase-out/Discontinued

PREFACE

Intended Readers :

This manual is intended for engineers who use the 4-bit single-chip microcomputer 17K series and are responsible for using the IE-17K and designing and developing application system.

Purpose :

This purpose of this manual is to describe the IE-17K in-circuit emulator that is used when designing and developing 17K series application systems and its interpreter CLICE (Command Language for In-Circuit Emulator) and to provide the user with an understanding of the various functions of this IE.

Organization :

This manual consists primarily of the following:

- General
- Specifications
- Installation
- Starting
- Description of commands
- Editor
- Programmable pattern generator (PPG)
- Program execution
- SE board PROM creation
- Error messages

Requirements :

Readers of this manual must have a general understanding of electric and logic circuits and microcomputers.

To gain an understanding of the functions of the IE-17K

→ Read this manual in accordance with the table of contents.

To study the functions of a specific command in detail

→ Read Chapter 5 "Description of Commands".

Legend :

The following symbols are used in this manual:

- ↵ : Line feed input
- { } : Indicate that one of the character strings inside the braces should be selected.
- [] : Indicates that omission is possible.
- : Indicates console input.
(under bar)
- | : Indicates control key input.
- ␣ : Indicates space key input.

Terminology :

The terms used in this manual are defined below.

Term	Meaning
Target device	Device to be emulated (this chip)
User program	Program to be debugged (program written by the user).
Target system	System to be debugged (system created by the user). The target system includes the target program and user hardware. In the narrow sense, it indicates the hardware only.

CONTENTS

CHAPTER 1.	GENERAL	1-1
1.1	General	1-1
1.2	Features	1-3
1.2.1	Interface with Target System	1-3
1.2.2	Program Memory	1-3
1.2.3	Emulation Method	1-3
1.2.4	Break Functions	1-3
1.2.5	Real-Time Trace Function	1-4
1.2.6	Data Memory Coverage Function	1-5
1.2.7	Program Memory Coverage Function	1-5
1.2.8	Programmable Pattern Generator (PPG) Function	1-6
1.2.9	Other Features	1-6
1.3	Composition	1-7
1.3.1	System Diagram	1-7
1.3.2	Block Diagram	1-7
CHAPTER 2.	SPECIFICATIONS	2-1
2.1	Main LSI	2-1
2.2	Console Interface	2-1
2.3	Environmental Conditions	2-2
2.4	Power Requirement	2-2
2.5	Internal Power Supply	2-2
2.6	Power Requirement of Each Board	2-2
2.7	Dimensions	2-2
2.8	Exterior Views	2-3
2.9	Accessories	2-5
CHAPTER 3.	INSTALLATION	3-1
3.1	Memory Board and Supervisor Board Removal	3-1
3.2	Switch Settings	3-3
3.2.1	Memory Board Switch Settings	3-3

3.2.2	Supervisor Board Switch Settings	3-6
3.3	Connector Connections	3-8
3.3.1	Memory Board Internal Connectors	3-8
3.3.2	Supervisor Board Internal Connectors	3-9
3.4	SE Board Installation	3-10
3.5	Connection to Host Machine	3-12
3.6	Connection to PROM Programmer	3-13
3.7	Connection to Target System	3-14
CHAPTER 4. START-UP		4-1
4.1	Communications Using Windows (Version 3.1)	4-1
4.1.1	Terminal Start-Up	4-2
4.1.2	Settings	4-5
4.1.3	Downloading a Program to IE-17K	4-9
CHAPTER 5. DESCRIPTION OF COMMANDS		5-1
5.1	Prompt	5-2
5.2	Command Line Format	5-4
5.3	Command Buffer	5-6
5.4	Character Set	5-7
5.4.1	Special Control Characters	5-7
5.4.2	Special Characters	5-10
5.4.3	Dummy Character	5-12
5.5	Expression	5-14
5.6	Constant	5-16
5.7	Variables	5-17
5.7.1	Array	5-17
5.7.2	Q-Register	5-17
5.8	Built-In Macro Commands	5-19
5.8.1	Program Memory Control Commands.....	5-19
5.8.2	Data Memory Control Commands	5-40
5.8.3	Peripheral Circuit Control Commands	5-47
5.8.4	Emulation Commands	5-53
5.8.5	Break/Trace Condition Control Commands	5-62
5.8.6	Coverage Display Command	5-98

5.8.7	Program Pattern Generator (PPG) Control Commands	5-101
5.8.8	Help Command	5-111
CHAPTER 6.	PROGRAMMABLE PATTERN GENERATOR (PPG)	6-1
6.1	PPG Data Display and Modification	6-1
6.2	PPG Probe Valid Bits and Step Rate Setting	6-1
6.3	PPG Starting and Stopping	6-2
6.4	PPG Usage Precautions	6-3
6.5	PPG Examples	6-4
CHAPTER 7.	PROGRAM EXECUTION	7-1
7.1	Real-Time Emulation	7-1
7.2	Break Point Setting	7-2
7.2.1	Break by Program Address	7-4
7.2.2	Break by Data Memory Modification	7-8
7.2.3	Break by Multiple Break Condition	7-12
7.3	1-Step Emulation	7-14
CHAPTER 8.	SE BOARD PROM CREATION	8-1
CHAPTER 9.	ERROR MESSAGES	9-1
9.1	Command Errors	9-1
9.2	Hardware Errors	9-8
APPENDIX A.	PRIMITIVE COMMANDS	A-1
A.1	Primitive Commands Table	A-2
A.2	Array Table	A-5
A.3	Condition Register Offset Address	A-8
A.4	Condition Unit Register Offset Address	A-10

A.5	Primitive Commands Descriptions	A-13
A.5.1	Pointer	A-13
A.5.2	Function	A-17
A.5.3	Assignment	A-25
A.5.4	Argument Stack	A-32
A.5.5	Q-Stack	A-35
A.5.6	Macro	A-38
A.5.7	Control	A-44
A.5.8	Display	A-51
A.5.9	Others	A-58
A.6	EDITOR	A-65
A.6.1	Command Buffer Editing	A-65
A.6.2	Q-Register Editing	A-65
A.6.3	Editor Commands	A-66
APPENDIX B. BUILT-IN MACRO COMMANDS		B-1
B.1	Program Memory Control Commands	B-1
B.2	Data Memory Control Commands	B-2
B.3	Peripheral Circuit Control Commands	B-3
B.4	Emulation Commands	B-4
B.5	Break/Trace Condition Control Commands	B-5
B.6	Coverage Display Command	B-5
B.7	Program Pattern Generator (PPG) Control Commands	B-6
B.8	Help Command	B-6

List of Figures

Figure No.	Title	Page
1-1	IE-17K System Diagram	1-7
1-2	IE-17K Block Diagram	1-8
2-1	IE-17K Exterior Views	2-3
2-2	Nomenclature	2-4
3-1	Board Layout	3-2
3-2	Memory Board Switches Layout	3-3
3-3	RS-232-C Interface Circuit Diagram	3-5
3-4	Accessory Cable Connection	3-6
3-5	Supervisor Board Switches Positions	3-6
3-6	DIP Switch Settings	3-7
3-7	Connectors Layout (Memory Board)	3-8
3-8	Connectors Layout (Supervisor Board)	3-9
3-9	SE Board Installation	3-11
3-10	IE-17K and PC-9800 Series Connection	3-12
3-11	IE-17K and PROM Programmer Connection	3-13
3-12	IE-17K and Target System Connection	3-14
4-1	Terminal Start-Up	4-3
4-2	Setting of Communication Conditions	4-6
4-3	Terminal Settings	4-8
4-4	Loading Wait Status	4-10
4-5	Sending of Text File	4-12
4-6	File Transfer	4-13
4-7	File Transfer End	4-15
5-1	CLICE Positioning	5-1
7-1	Break Condition Setting	7-3
7-2	Break by Break Condition Sequence	7-3

List of Tables

Table No.	Title	Page
1-1	Combination of CLICE and SIMPLEHOST Versions	1-2
5-1	Break/Trace Conditions Table	5-82
5-2	Trace State Transition	5-88
A-1	Commands Table	A-2
A-2	Array Table	A-5
A-3	Condition Register Offset Address	A-9
A-4	Condition Unit Register Offset Address	A-10

Phase-out/Discontinued

CHAPTER 1. GENERAL

This chapter describes the features and system configuration of the IE-17K.

1.1 GENERAL

The IE-17K is a software development tool for the 4-bit single-chip microcontroller 17K Series.

The IE-17K supports all the models in the 17K Series. A dedicated SE board is available for each model. The IE-17K is used in conjunction with these boards. The SE board has an emulation function of the hardware unique to each model and can also be used alone for program evaluation.

The IE-17K consists of two boards, memory board and supervisor board.

The IE-17K can be operated alone by connecting it to a terminal. A more advanced debugging environment can also be realized by connecting the IE-17K to the host machine and operating SIMPLEHOSTTM, which acts as the man-machine interface software between the IE-17K and the operator.

The combination of IE-17K interpreter CLICE (Command Language for In-Circuit Emulator) and SIMPLEHOST versions which can be used together is shown below.

Table 1-1 Combination of CLICE and SIMPLEHOST Versions

CLICE SIMPLEHOST	Ver. 1.5	Ver. 1.6
Ver. 1.10	o	x
Ver. 1.11	x	o

1.2 FEATURES

1.2.1 INTERFACE WITH TARGET SYSTEM

The actual product is used as the interface with the target system so that the electrical specifications are the same as those of the target product.

1.2.2 PROGRAM MEMORY

A CMOS static RAM mounted on the SE board is used as the program memory.

1.2.3 EMULATION METHOD

Two user program emulation methods are available, real-time emulation and 1 step emulation.

1.2.4 BREAK FUNCTIONS

(1) Programmable break function

The following four steps can be set.

- ① Break when single condition established.
- ② Multiple conditions of ① above are set and break is generated when one, or all, of these conditions are established.
- ③ Multiple conditions (up to 4) of ② above are set and break is generated when one of these conditions is established.
- ④ Break is generated when the conditions of ③ above are established in the set order.

The following can be set as break conditions.

- . Program memory address
- . Data memory address
- . Data memory contents
- . Register file address
- . Register file contents
- . Stack level
- . State of external pin (logic analyzer)
- . Interrupt
- . DMA
- . Operation code
- . Instruction execution count
- . Condition established count

(2) Error detection function

This function aborts the program or issues a warning when the program accessed a resource not allowed by the software development target product, etc.

It detects the following errors.

- . Access to an illegal data memory
- . Access to an illegal system register
- . Stack level overflow/underflow
- . Read or test of memory to which data was not written even once

1.2.5 REAL-TIME TRACE FUNCTION

This function stores the program executed result in real time. The trace memory size is 32K steps.

(1) The trace data are as follows.

- . Program memory address
- . Executed instruction code
- . Skipped instructions
- . Written data memory address
- . Written data memory contents
- . IE-17K logic analyzer terminal status
- . Relative time of each execution instruction

(2) Trace on/off condition can be specified.

1.2.6 DATA MEMORY COVERAGE FUNCTION

This function stores the data memory addresses which were written.

The following information can be obtained with this function.

- . Unwritten bits
- . Bits written "1"
- . Bits written "0"
- . Bits written "0" and "1"

1.2.7 PROGRAM MEMORY COVERAGE FUNCTION

This function stores the number of times each program memory address is executed.

The maximum value of each address counter is 255, and becomes 255 when an address is executed 255 times or more. This counter is counted up when the instruction of that address is not skipped but is executed, or when that address is referenced by table reference instruction (MOVT, etc.) and is not counted up when the instruction is skipped.

1.2.8 PROGRAMMABLE PATTERN GENERATOR (PPG) FUNCTION

The IE-17K has an on-chip programmable 14-channel pattern generator. The number of programmable steps is 8K steps. The execution speed per step can be set from approximately 1 us/STEP to 13333 us/STEP in 1 us steps.

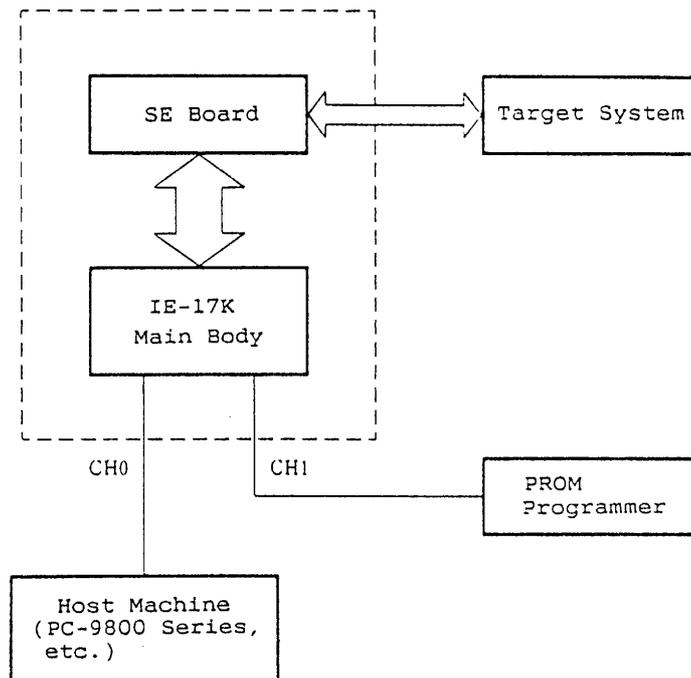
1.2.9 OTHER FEATURES

- (1) The IE-17K has two RS-232-C serial channels. Channel 0 is for console use and channel 1 is for PROM programmer, etc. use. A more advanced debugging environment can be realized by connecting channel 0 to a host machine PC-9800 Series and operating the man-machine interface software SIMPLEHOST.
- (2) EMI countermeasures are incorporated and VCCI Standards are satisfied.
- (3) A4 size (21 cm x 30 cm x 10 cm) is extremely compact and easy to carry.
- (4) Internal switching regulator allows use with commercial power.
- (5) IE-17K has a space for storing probes, etc.

1.3 COMPOSITION

1.3.1 SYSTEM DIAGRAM

Figure 1-1 IE-17K System Diagram

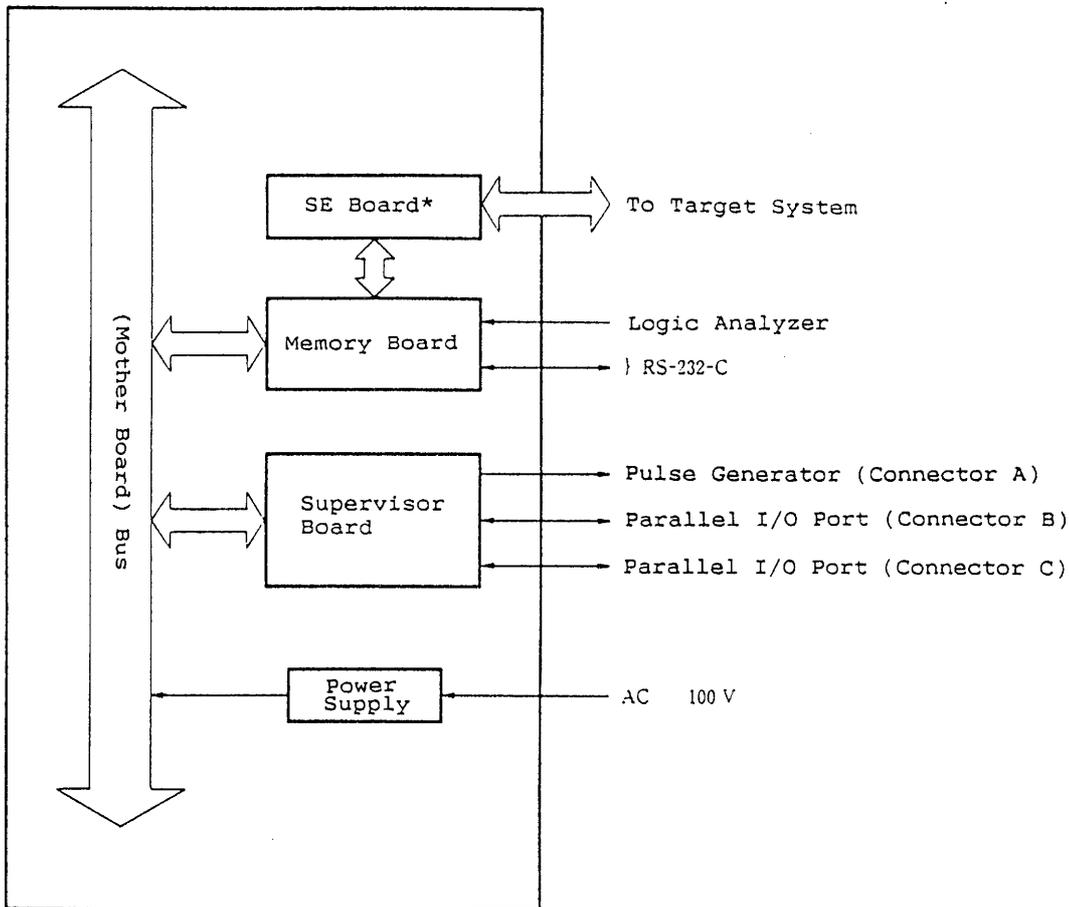


1.3.2 BLOCK DIAGRAM

The IE-17K consists of a mainframe and accessories. The mainframe consists of the following parts.

- . Cabinet (including connectors, switches, etc.)
- . Internal power supply
- . Supervisor (SV) board
- . Memory board
- . Mother board

Figure 1-2 IE-17K Block Diagram



* : An SE board (optional) is available for each model.

CHAPTER 2. SPECIFICATIONS

2.1 MAIN LSI

<Supervisor Board>

Supervisor CPU	uPD70116D	x1
Supervisor peripheral	uPD71011Cx1	x1
Supervisor peripheral	uPD71086C	x2
Supervisor peripheral	uPD71055G	x2
Monitor ROM	uPD27C512D	x2
RAM	uPD41256V	x16
RAM	uPD4364G	x2

<Memory board>

Memory	uPD43256G	x15
Memory	uPD4364G	x1
Interrupt controller	uPD71059G	x1
Timer/counter	uPD71054G	x2
Serial controller	uPD71051G	x2
8-bit latch	uPD71082C	x2

2.2 CONSOLE INTERFACE

RS-232-C x 2ch (CH0, CH1)

Baud rate (bps) : 600, 1200, 2400, 4800, 9600, 19200,
38400, 76800

Character length : 7, 8 bits

Stop bit length : 1, 2 bits

Parity : None, even, odd

2.3 ENVIRONMENTAL CONDITIONS

Operating temperature range : +10 to +40°C
Storage temperature range : -10 to +50°C
(no condensation)

2.4 POWER REQUIREMENT

AC 85 to 220 V

2.5 INTERNAL POWER SUPPLY

DC +5 V 2.0 A (MAX.)
DC +12 V 0.2 A (MAX.)

2.6 POWER REQUIREMENT OF EACH BOARD

<Memory board>

DC +5 V 110.0 mA (TYP.)
DC +12 V 32.5 mA (TYP.)

<Supervisor board>

DC +5 V 1140.0 mA (TYP.)

2.7 DIMENSIONS

Cabinet dimensions 210 mm x 300 mm x 100 mm (excluding projecting parts)

2.8 EXTERIOR VIEWS

Figure 2-1 IE-17K Exterior Views

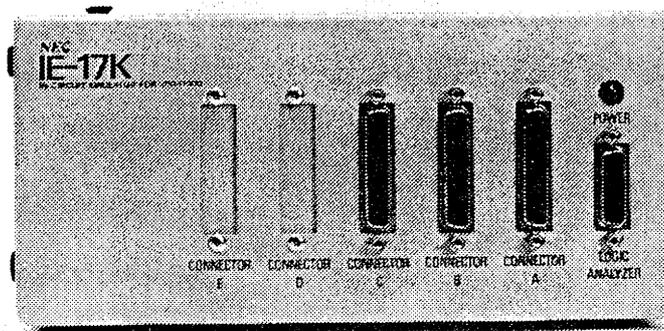
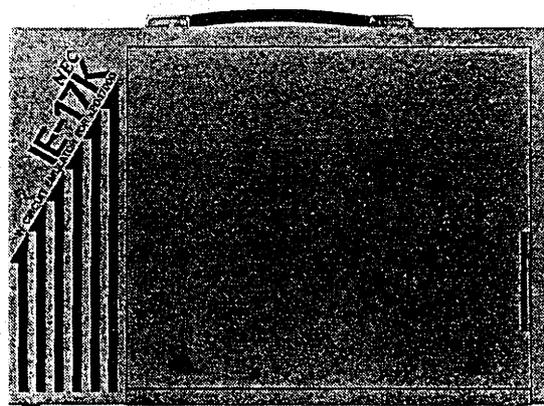
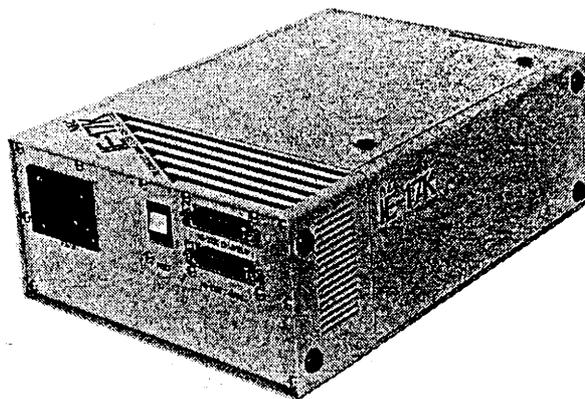
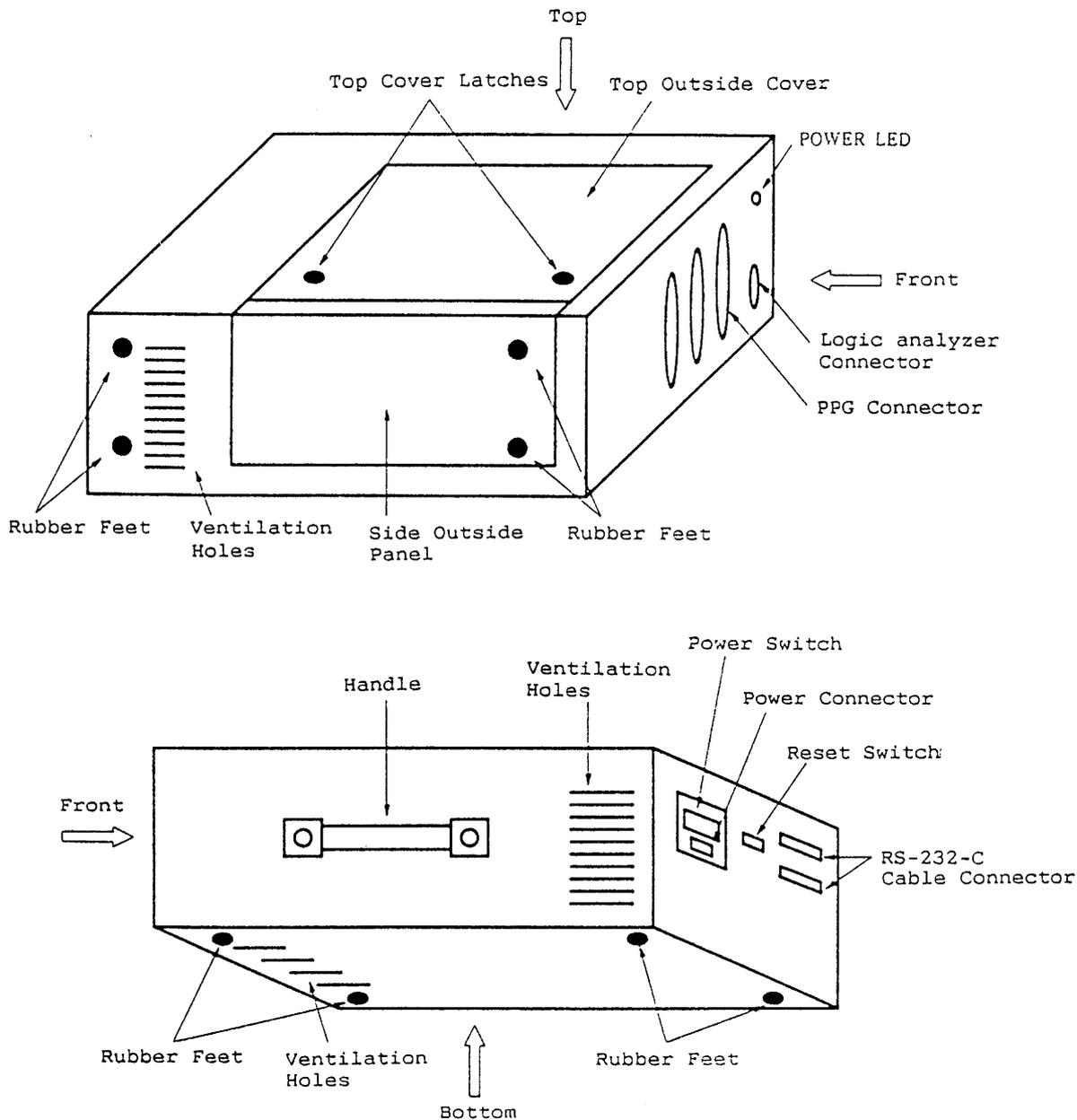


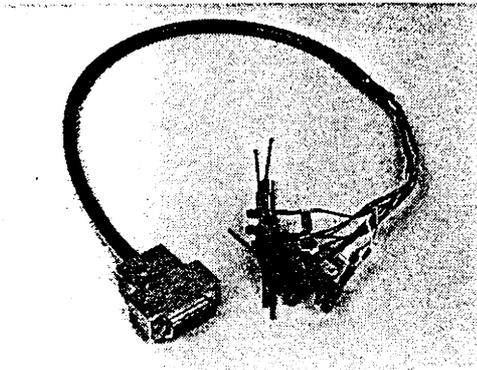
Figure 2-2 Nomenclature



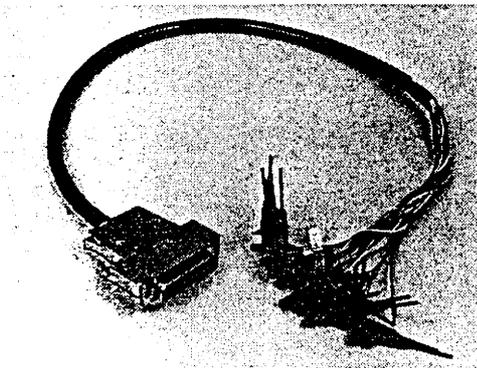
2.9 ACCESSORIES

When the IE-17K is shipped, the following parts are packed in the same carton as IE-17K accessories.

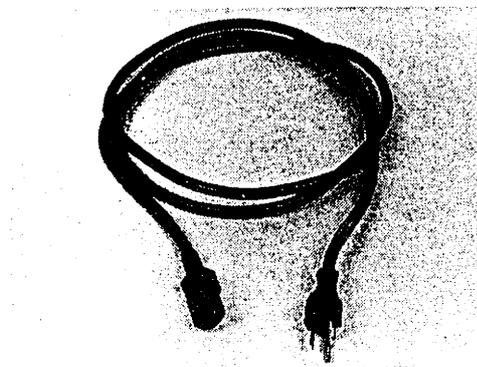
- (1) Logic analyzer probe (in cabinet) 1



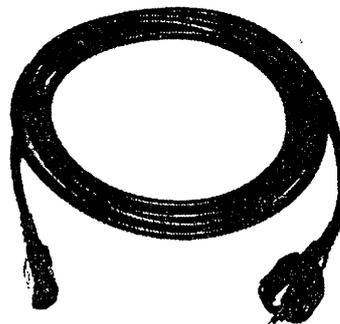
- (2) PPG probe (in cabinet) 1



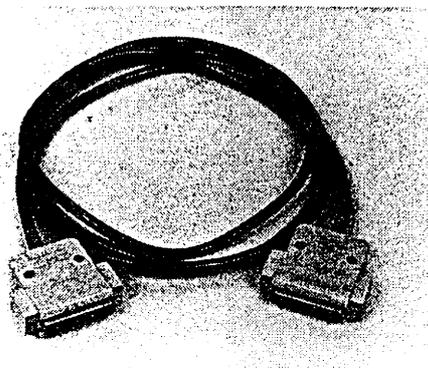
- (3) 100 V power cable 1



(4) 200 V power cable 1



(5) RS-232-C cable (cross cable) 1



(6) Others

- . User's manual (English, Japanese) 1 each
- . Warranty 1 copy
- . Packing list 1 copy

CHAPTER 3. INSTALLATION

3.1 MEMORY BOARD AND SUPERVISOR BOARD REMOVAL

The IE-17K consists of two boards, memory board and supervisor board. Usually these boards do not have to be removed.

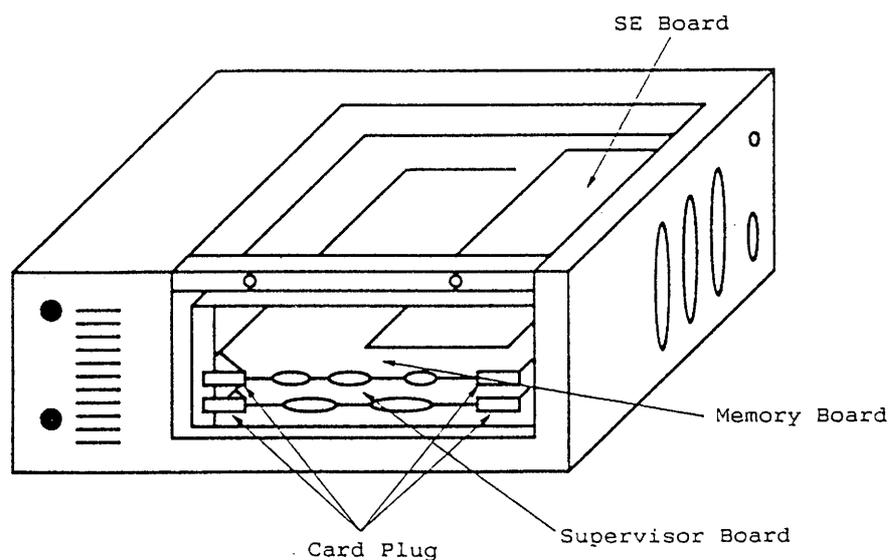
If these boards must be removed for IC replacement, switch setting, etc., proceed as follows.

- ① Remove the side outside panel. (Remove the mounting screws.)
- ② Remove the screws holding the side inside panel and remove the side inside panel from the IE-17K.
- ③ When an SE board is mounted at the memory board, first remove it from the memory board.
- ④ Disconnect all the cables connected to the memory board and supervisor board connectors.
- ⑤ Remove the supervisor board at the bottom slot. The supervisor board can be easily removed by pulling the card pullers installed to the supervisor board forward.
- ⑥ Remove the memory board at the top slot. The memory board can be removed easily by pulling the card puller installed to the memory board forward.

[Removal and installation precautions]

Always begin removal from the supervisor board and installation from the memory board.

Figure 3-1 Board Layout

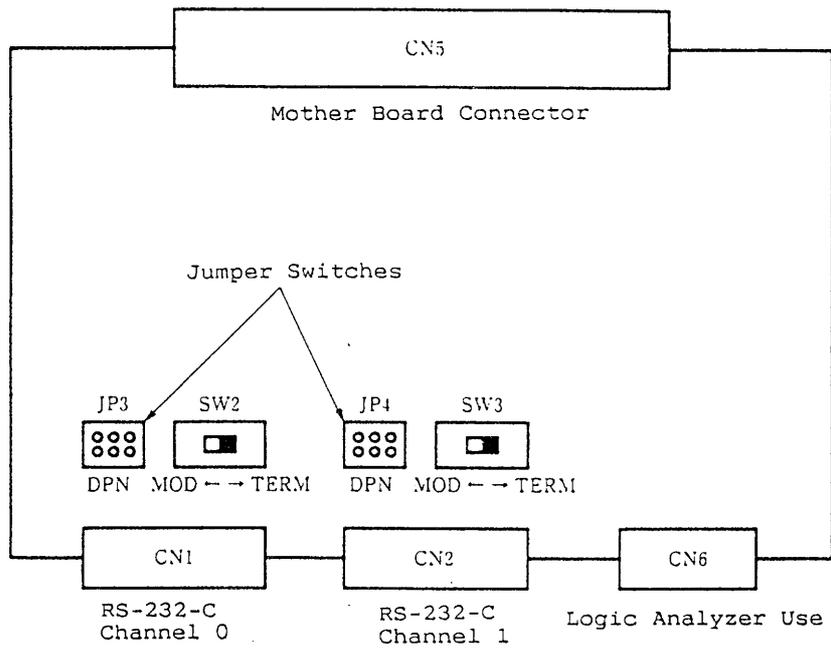


3.2 SWITCH SETTINGS

Set the switches on each board as described below.

3.2.1 MEMORY BOARD SWITCH SETTINGS

Figure 3-2 Memory Board Switches Layout



The switches on the memory board are for RS-232-C setting.

JP3 and SW2 are for channel 0 and JP4 and SW3 are for channel 1.

JP3 and JP4 switch the RTS signal. Set them to match the host machine used.

SW2 and SW3 switch the terminal mode and modem mode. The switches set at the factory are as follows.

JP3, JP4	Open
SW2, SW3	Terminal mode

When the IE-17K is connected to a PC-9800 Series with the accessory RS-232-C cable, it can be used in the factory setting state above.

Figure 3-3 RS-232-C Interface Circuit Diagram

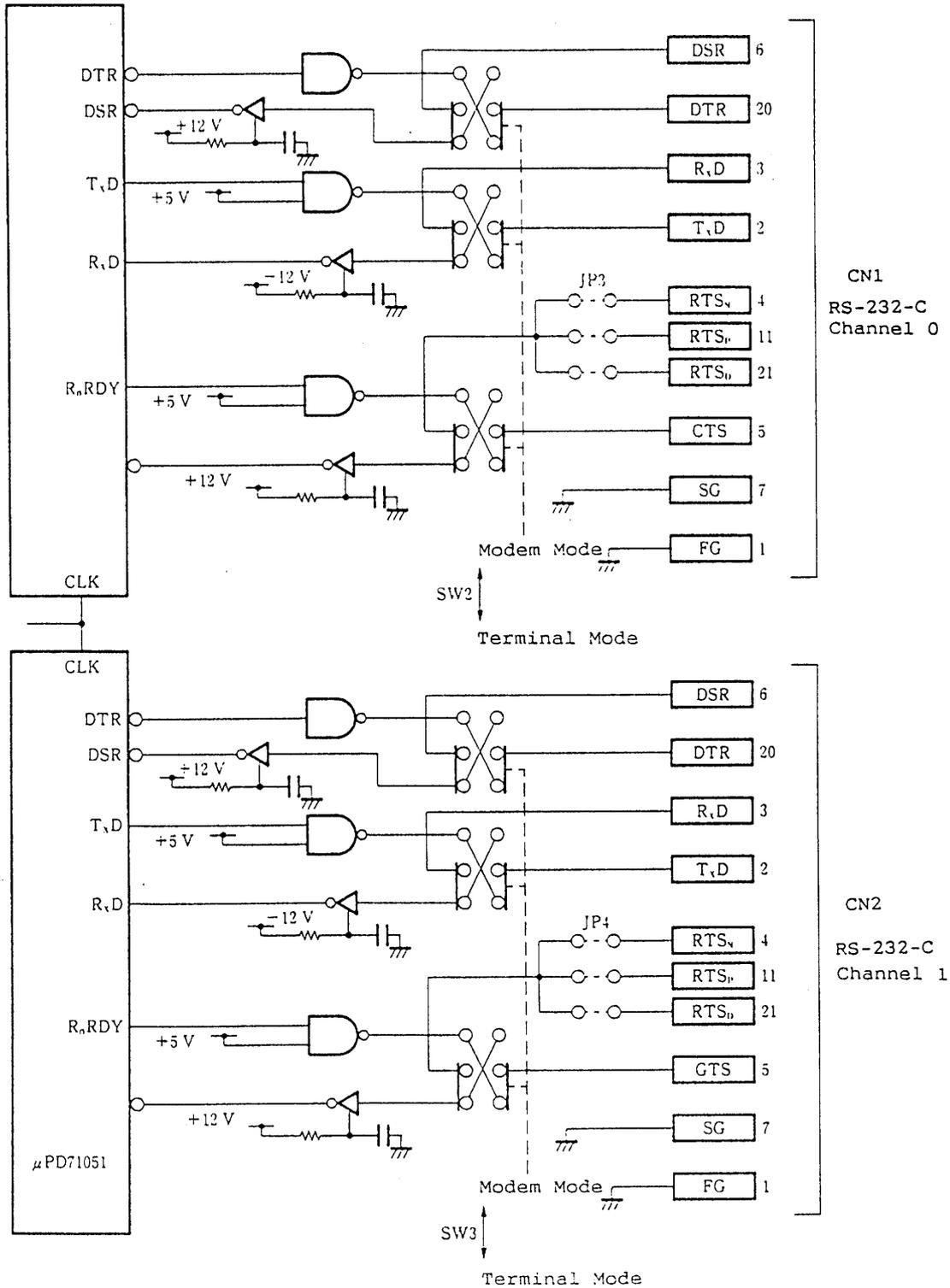
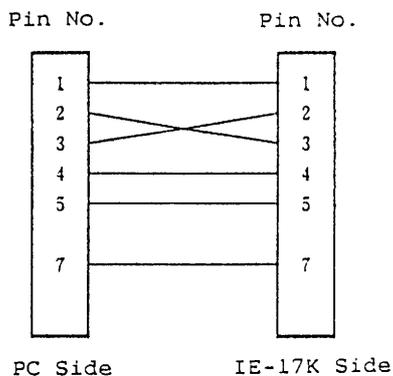


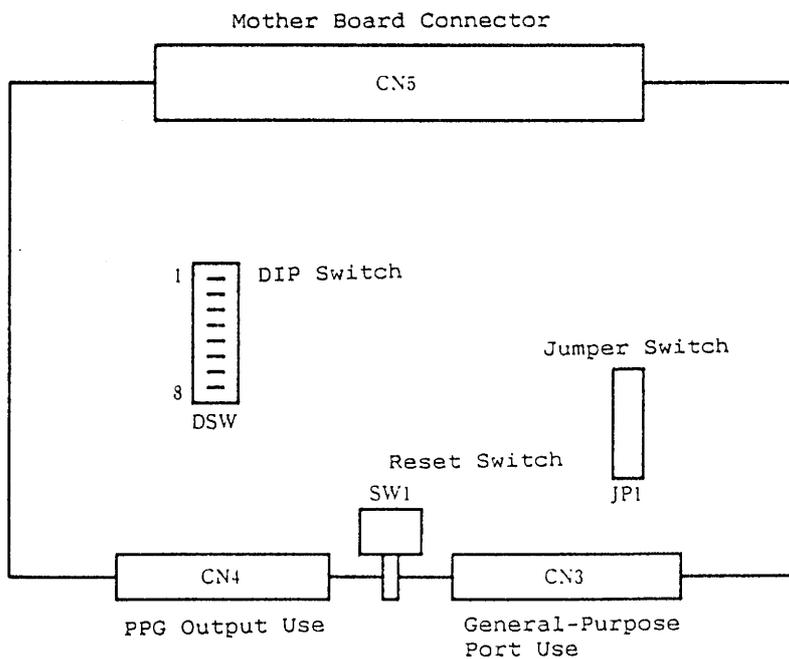
Figure 3-4 Accessory Cable Connection

[RS-232-C Cable (cross cable)]



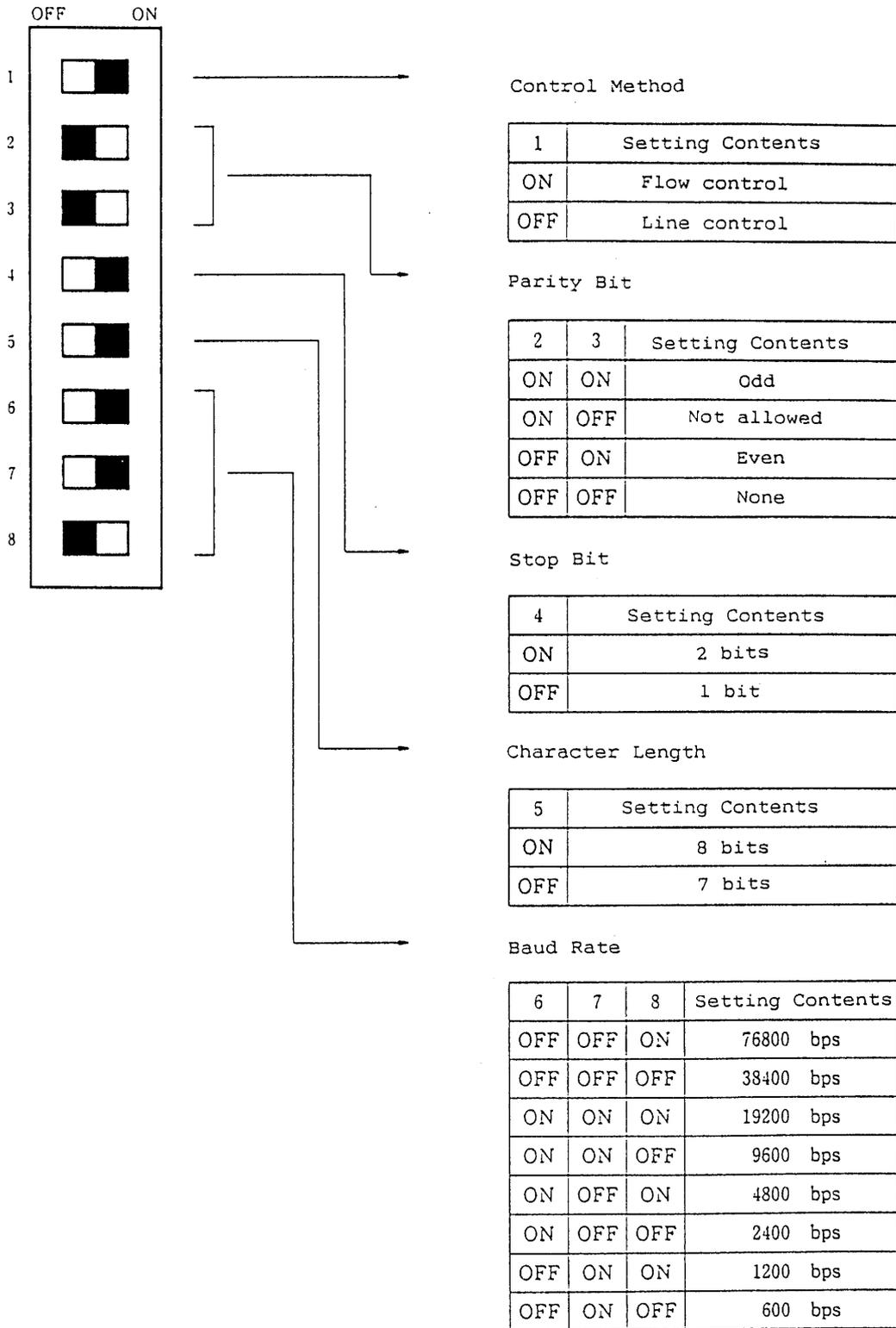
3.2.2 SUPERVISOR BOARD SWITCH SETTINGS

Figure 3-5 Supervisor Board Switches Positions



RS-232-C setting is performed by DIP switch.
Use JP1 in the factory setting state.

Figure 3-6 DIP Switch Settings



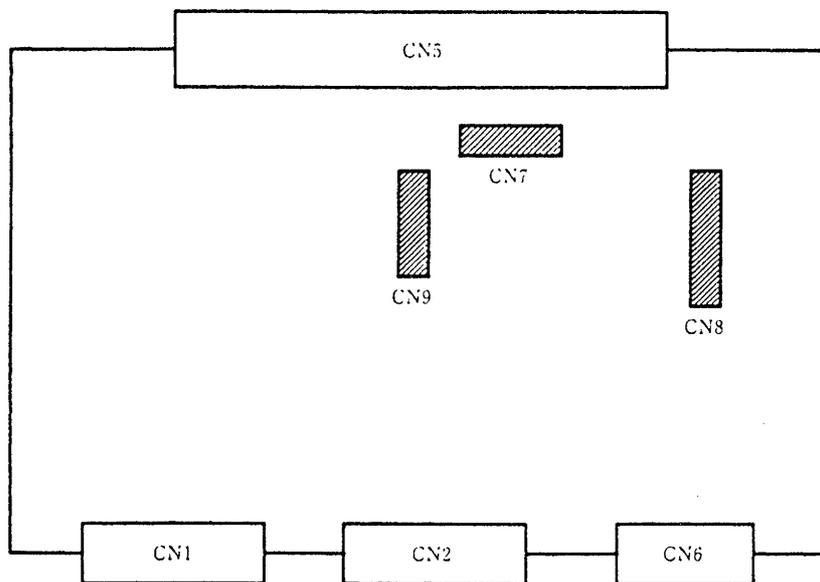
Remarks : The shaded area is factory setting

3.3 CONNECTOR CONNECTIONS

3.3.1 MEMORY BOARD INTERNAL CONNECTORS

The layout of the three connectors CN7, CN8, and CN9 of the SE board and memory board connection section and connector CN5 for mother board connection is shown below.

Figure 3-7 Connectors Layout (Memory Board)



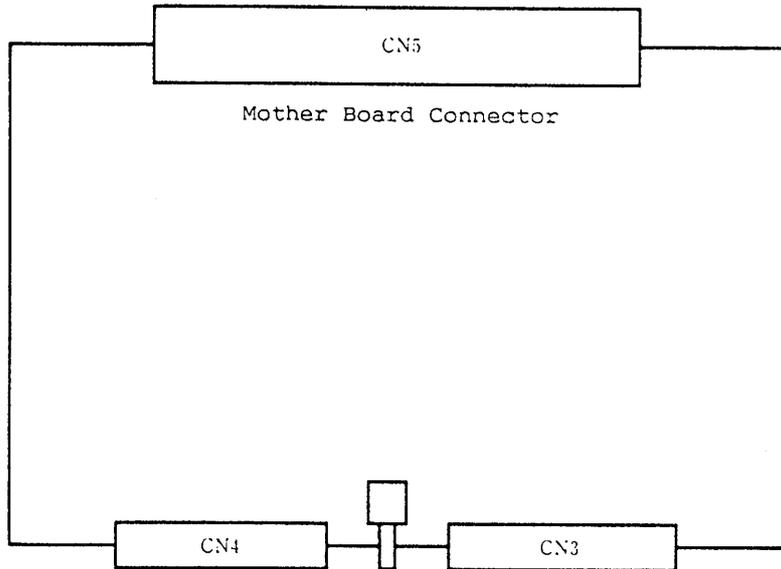
The connectors are connected as follows.

- CN1 Connects to cable connector stamped CN1
- CN2 Connects to cable connector stamped CN2
- CN6 Connects to 15-pin cable connector
- CN7]
- CN8] Mounts SE boards
- CN9]
- CN5 Connects to mother board

3.3.2 SUPERVISOR BOARD INTERNAL CONNECTORS

The layout of the supervisor board connectors is shown below.

Figure 3-8 Connectors Layout (Supervisor Board)



The connectors are connected as follows.

- CN3 Connects to 50-pin cable connector
- CN4 Connects to 25-pin cable connector
- CN5 Connects to mother board

3.4 SE BOARD INSTALLATION

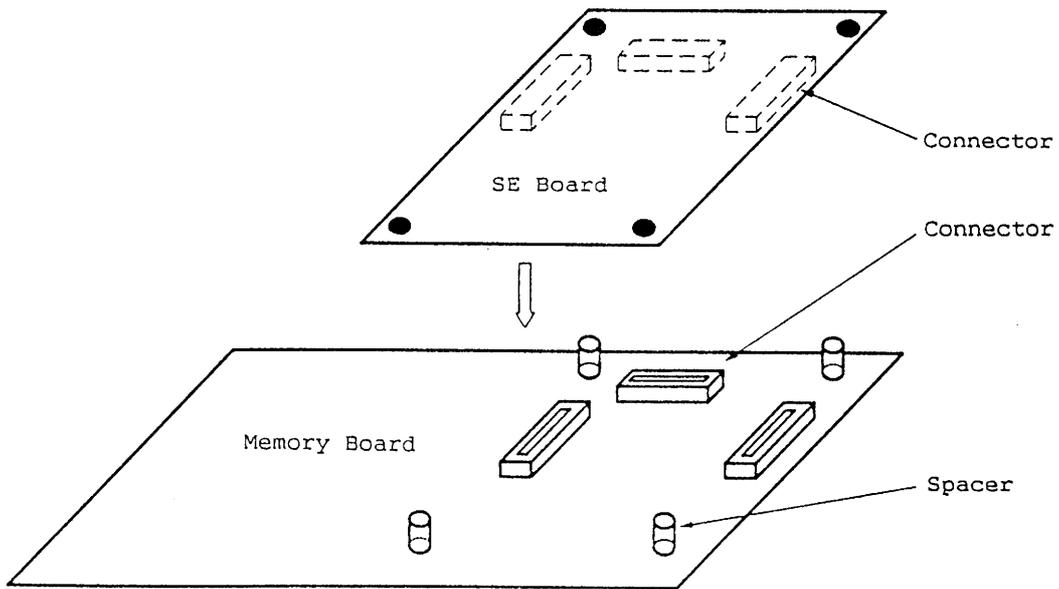
When the IE-17K is shipped, the memory board and supervisor board are installed to the IE-17K as control boards. However, the SE board corresponding to the 17K Series model is not installed. Therefore, when development the 17K Series, the SE board corresponding to the model must be installed to the IE-17K separately from the IE-17K.

For a detailed description of the SE boards, refer to the user's manual for each SE board. Then installing an SE board to the IE-17K, proceed as follows.

- ① Pull the cover latches at the top of the IE-17K and remove the top outsider cover.
- ② Remove the top inside cover mounting screws and remove the top inside cover.
- ③ Remove the screws attached to the spacers on the memory board.
- ④ Connect the connector on the memory board and the connector at the rear top of the SE board.

The SE board is installed by pushing it in perpendicularly to the memory board and is removed by pulling it out perpendicularly.

Figure 3-9 SE Board Installation



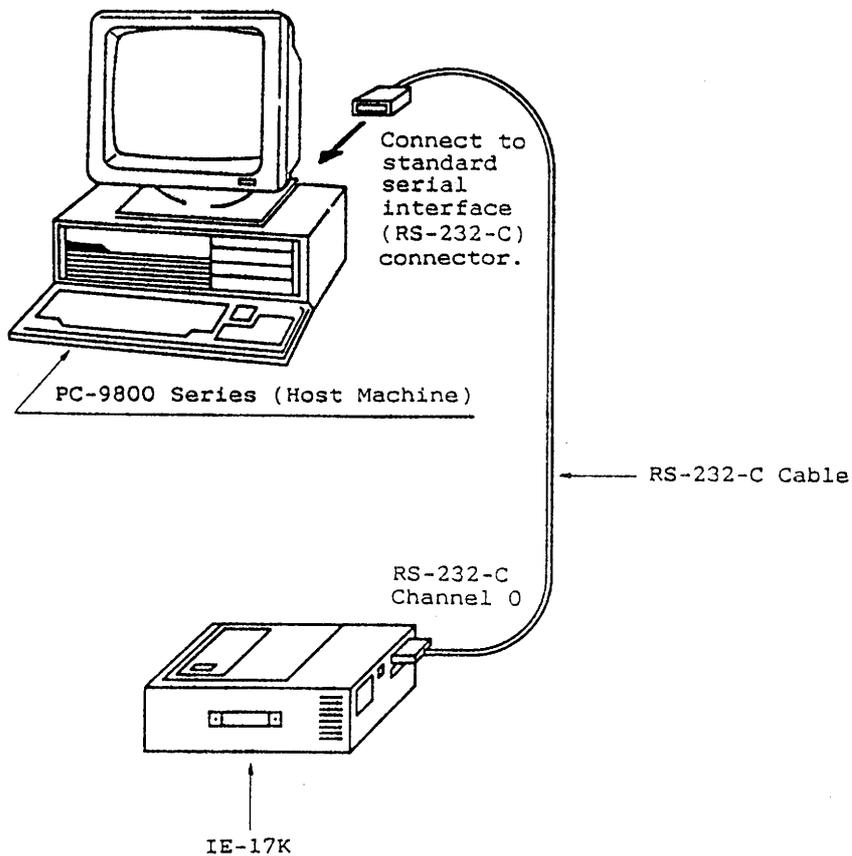
- ⑤ Fasten the SE board and the memory board with the screws removed at step ② .
- ⑥ Reinstall the top inside and outside covers.

3.5 CONNECTION TO HOST MACHINE

This section describes an example of connection when the PC-9800 Series is used as the host machine.

Turn OFF the IE-17K and PC-9800 Series power switch and connect the RS-232-C channel 0 connector of the IE-17K to the standard serial interface (RS-232-C) connector of the PC-9800 Series with the RS-232-C cable supplied with the IE-17K.

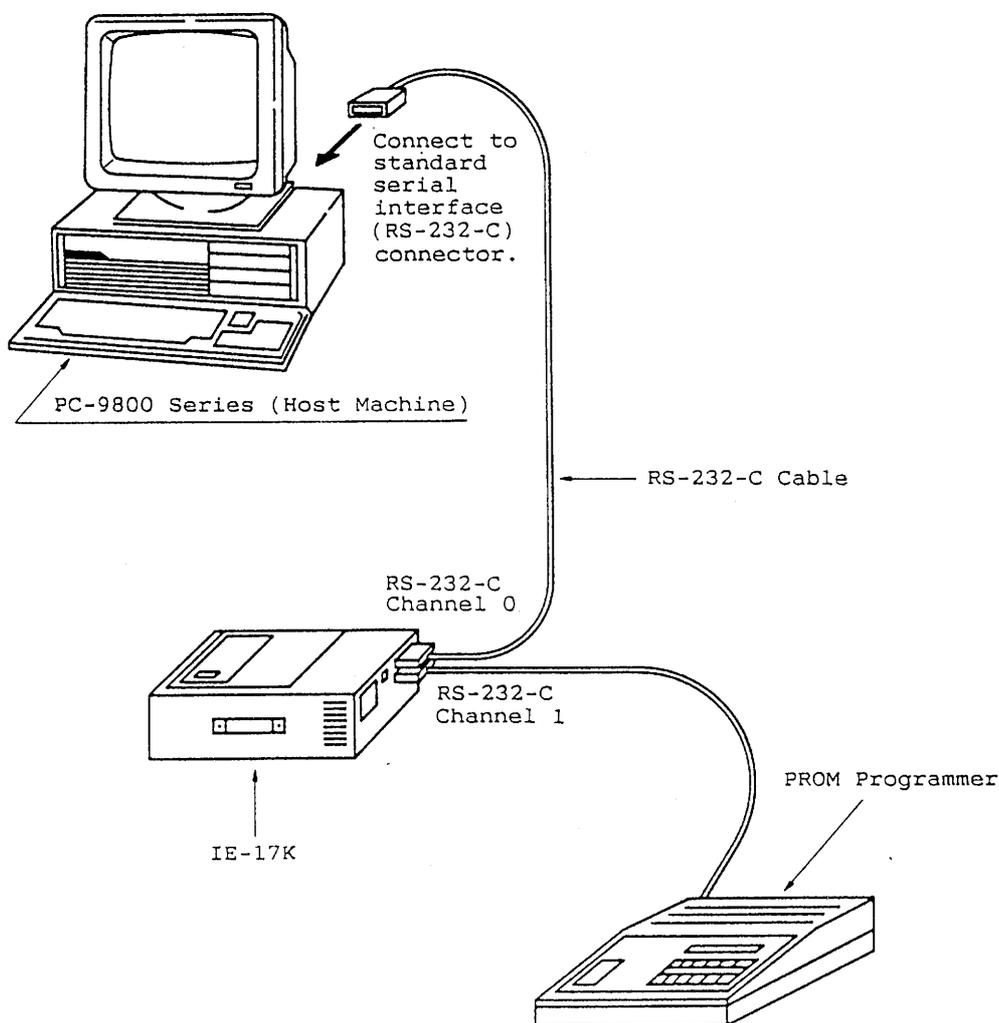
Figure 3-10 IE-17K and PC-9800 Series Connection



3.6 CONNECTION TO PROM PROGRAMMER

To load the program from the IE-17K body to the PROM programmer, in the state in which the IE-17K body is connected to the host machine (PC-9800 Series), connect the RS-232-C channel connector of the IE-17K body to the PROM programmer with the PROM programmer RS-232-C cable.

Figure 3-11 IE-17K and PROM Programmer Connection

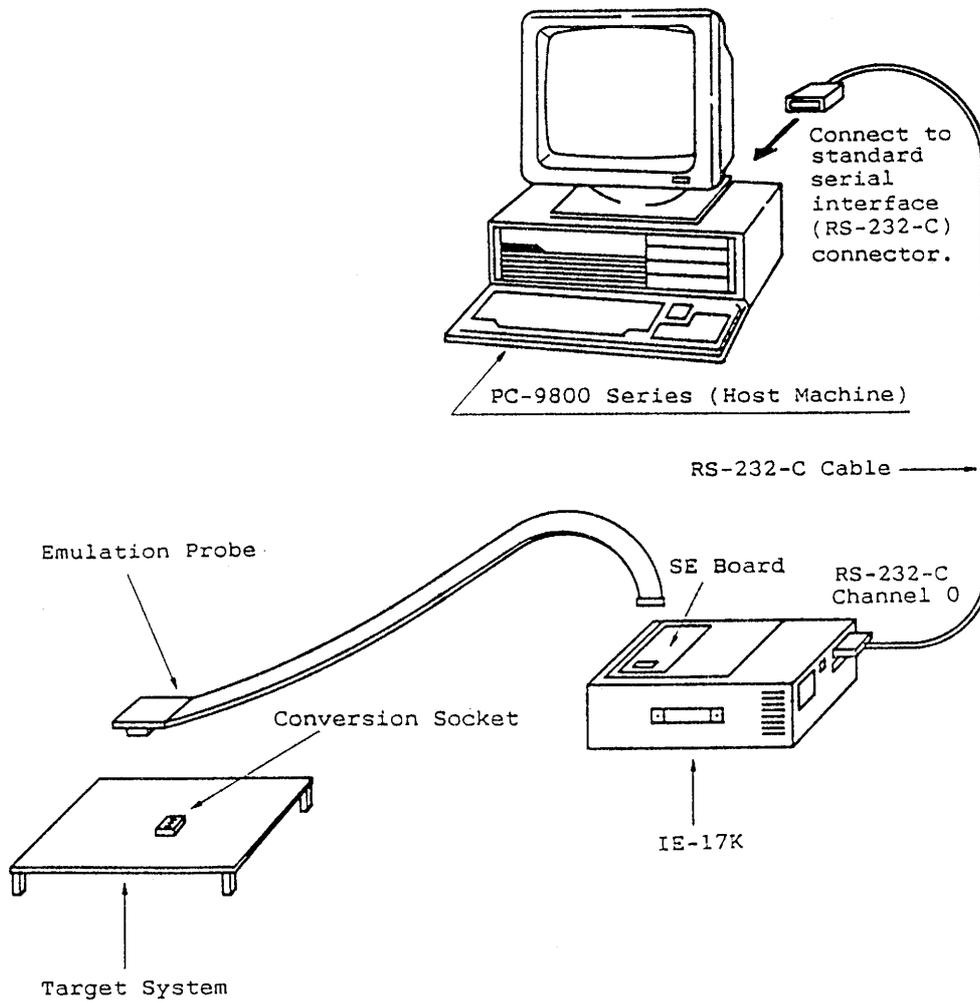


3.7 CONNECTION TO TARGET SYSTEM

Connect the emulation probe to the SE board and the target system.

For more information, refer to the user's manual for each SE board.

Figure 3-12 IE-17K and Target System Connection



CHAPTER 4. START-UP

The IE-17K is used by connecting it to a PC-9800 series or IBM PC/ATTM host machine using a dedicated IE-17K RS-232-C cable (supplied with the IE-17K).

Communication and start-up of the IE-17K is performed using commercially sold software that is RS-232-C compatible.

This chapter introduces the method of communication using a WindowsTM (version 3.1) terminal. If communication is performed using commercially sold communications software, this chapter should be used as a reference when starting up the IE-17K.

NEC has prepared special communications software, called SIMPLEHOST, which is to be used with the IE-17K (man-machine interface software) and that operates using Windows.

Besides having communications functions for use with the IE-17K, in order to smoothly perform debugging, SIMPLEHOST has a command menu that is aimed at making operation manual free, and because the execution results are graphical, there is no need to read this chapter when SIMPLEHOST is used.

4.1 COMMUNICATIONS USING Windows (Version 3.1)

The description below will center around using Windows (Version 3.1) for the PC-9800 series, and how to connect the IE-17K to the PC-9800 series.

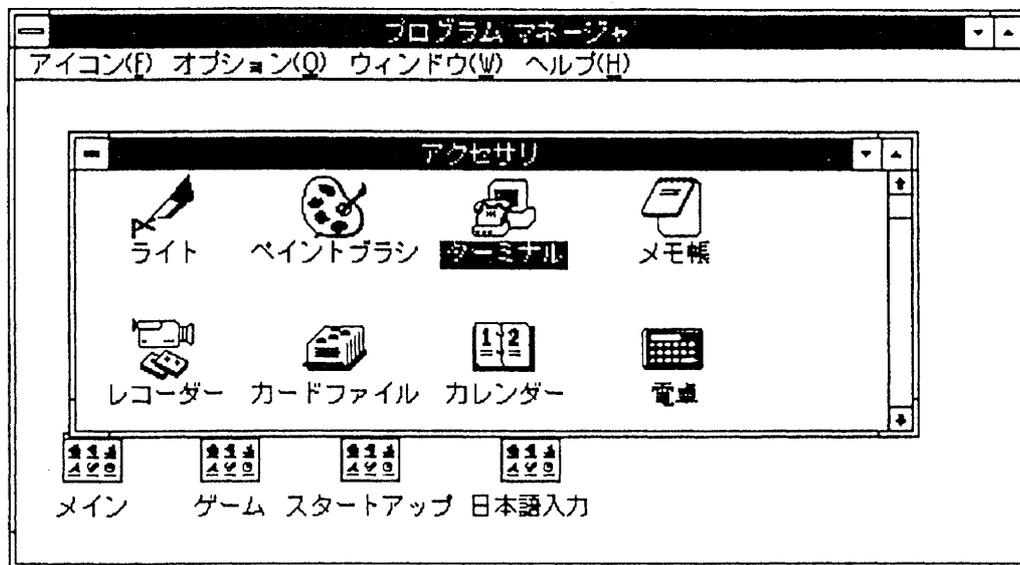
Before starting up Windows, check to make sure that the SE board is installed in the IE-17K and PC-9800 series, and that they are connected using the dedicated IE-17K RS-232-C cable, and that the IE-17K power has been turned ON (see Chapter 3 "Installation").

4.1.1 TERMINAL START-UP

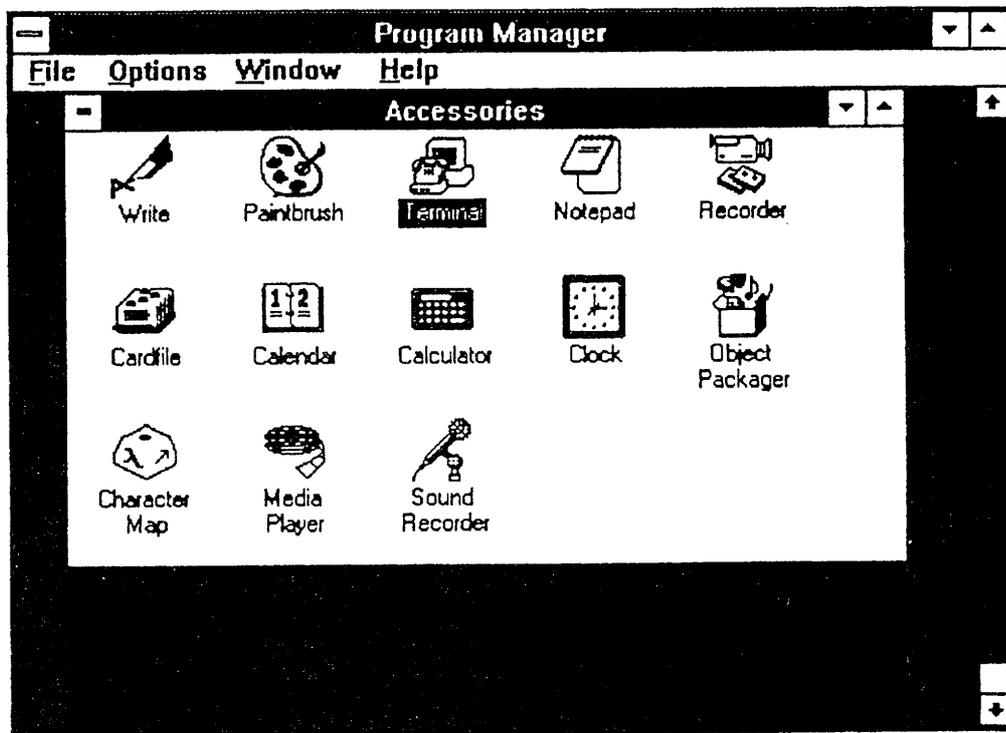
First, starting from the Windows opening screen, select (double click) the "Accessory" group icon, and select the "Terminal" icon, and then execute it (double click). An example selection screen is given below.

Figure 4-1 Terminal Start-Up

(a) PC-9800 Series



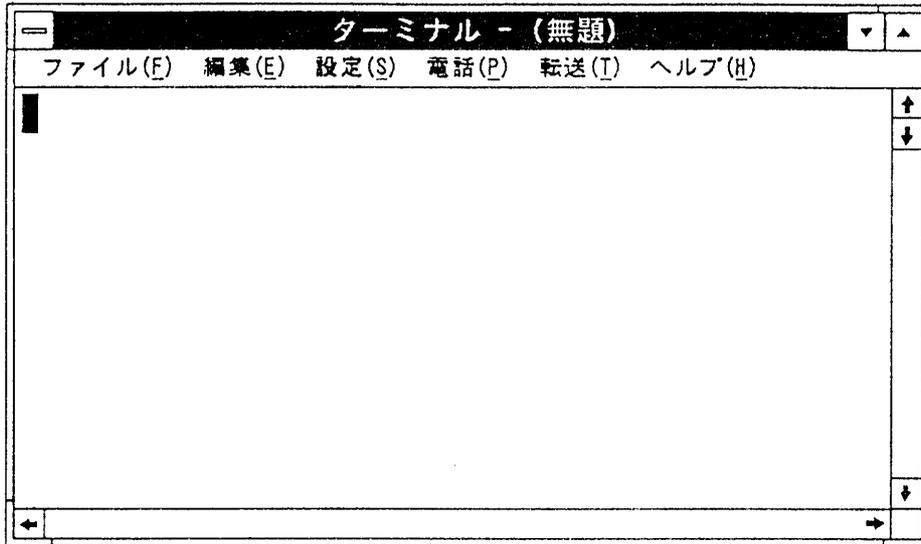
(b) IBM PC/AT



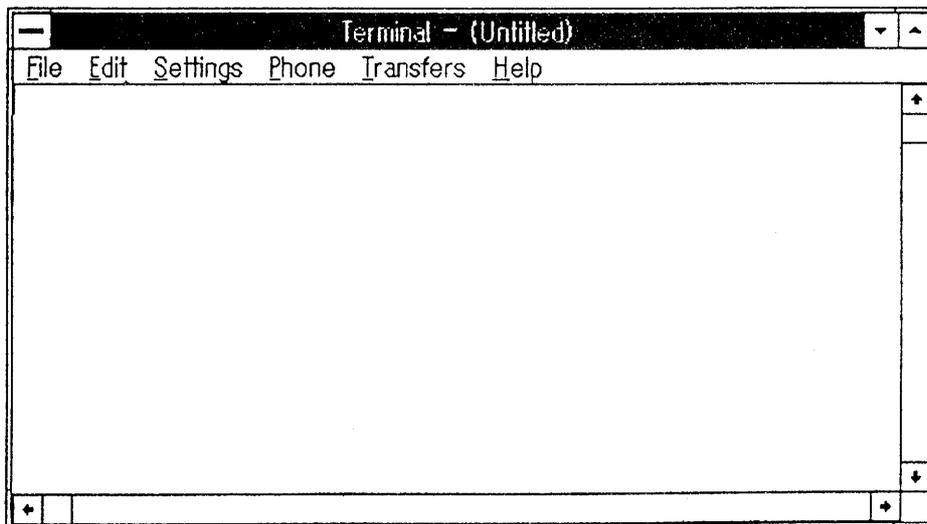
The "Terminal" window is opened after double clicking on the the "Terminal" icon of the "Accessory" group.

Figure 4-1 Terminal Start-Up (cont'd)

(a) PC-9800 Series



(b) IBM PC/AT



4.1.2 SETTINGS

In order to exchange data and commands with the IE-17K, the "Communication Condition Settings", which specify the communication speed, etc., and the "Terminal Preferences", which specify the operation of the terminal during communication, are set.

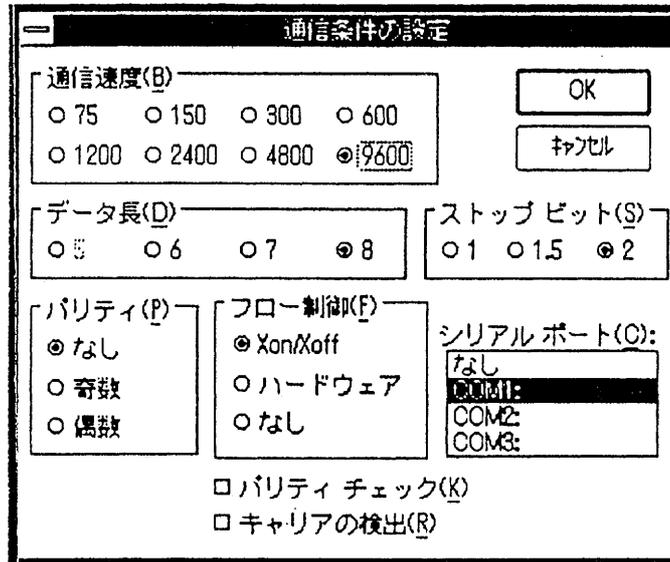
(1) Communication condition settings

The communication settings for the IE-17K at the time of shipping are: 9600 baud communication speed, 8-bit data length, 2 stop bits, no parity, and Xon/Xoff flow control. Therefore, the Windows settings must be set to match these conditions.

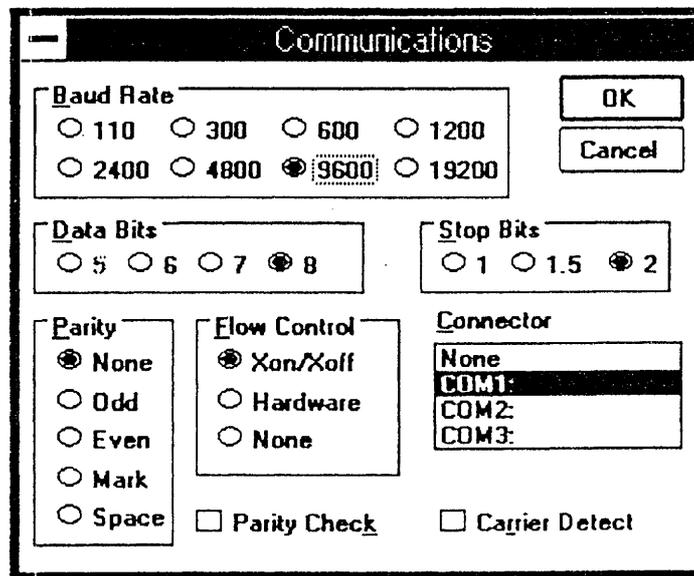
Select "Communication Conditions" from the menu bar item "Settings" in the "Terminal" window. After doing this, the "Communications" dialog box will appear, and the conditions should be set as shown in Figure 4-2.

Figure 4-2 Setting of Communication Conditions

(a) PC-9800 Series



(b) IBM PC/AT

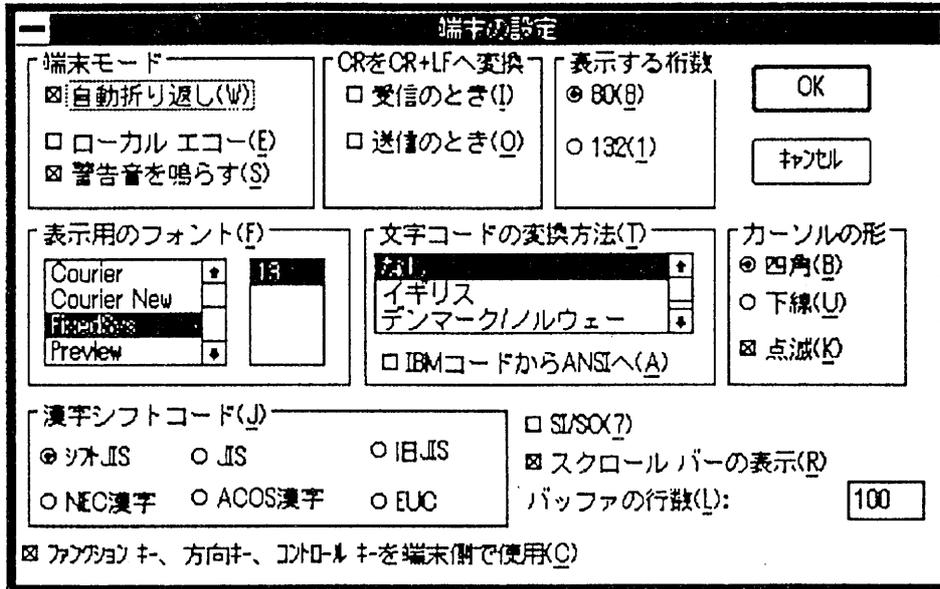


After all settings have been completed, click the "OK" button to end "Communications".

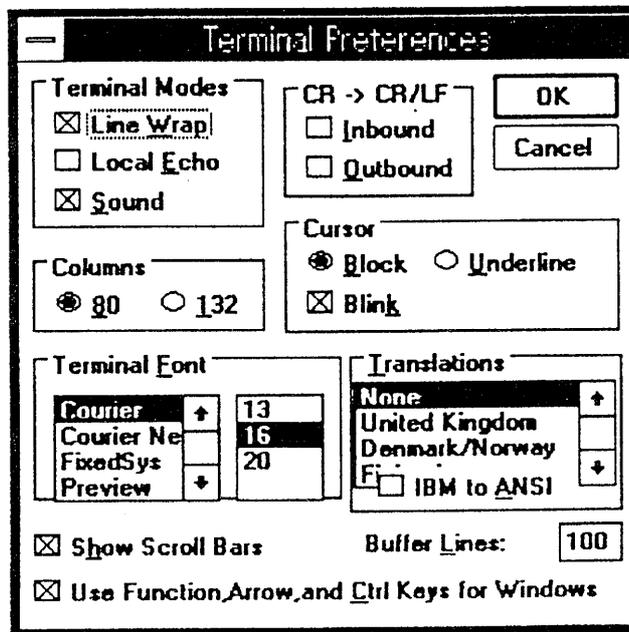
Next, select "Terminal Preferences" from the menu bar item "Settings" in the "Terminal" window, to specify the operation of the terminal during communication. The "Terminal Preferences" dialog box shown in Figure 4-3 will be displayed. Except for "Sound" and "Cursor", the settings should be as shown in Figure 4-3.

Figure 4-3 Terminal Settings

(a) PC-9800 Series



(b) IBM PC/AT



After all settings have been completed, click the "OK" button to end "Terminal Preferences".

4.1.3 DOWNLOADING A PROGRAM TO IE-17K

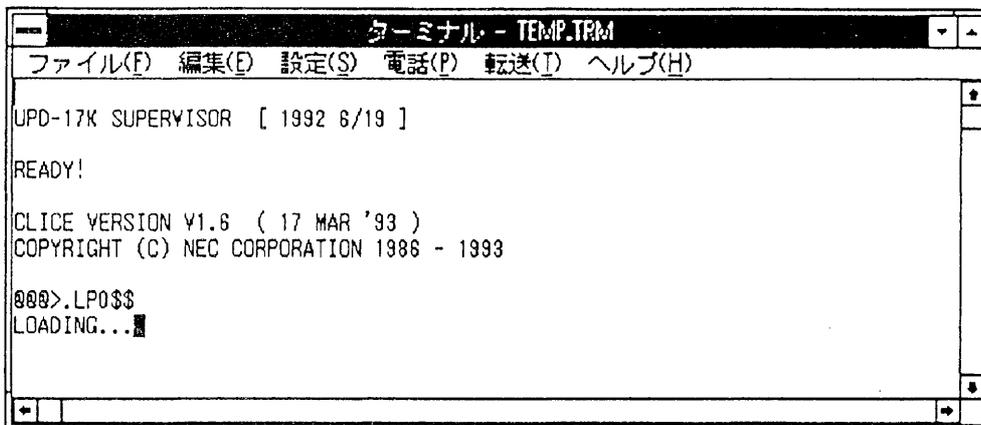
After the "Communications Settings " and "Terminal Preferences" have been completed, it is possible to communicate with the IE-17K. After pressing the reset switch on the "Terminal" window screen, an opening message and @@@> prompt are sent from the IE-17K to the "Terminal" window screen. If the @@@> prompt is not returned, the following problems may exist and should be checked:

- ① The SE board is not installed properly.
- ② Power is not being supplied to the SE board. (There are some SE boards that require two power supplies.)
- ③ The special IE-17K RS-232-C cable is not connected.

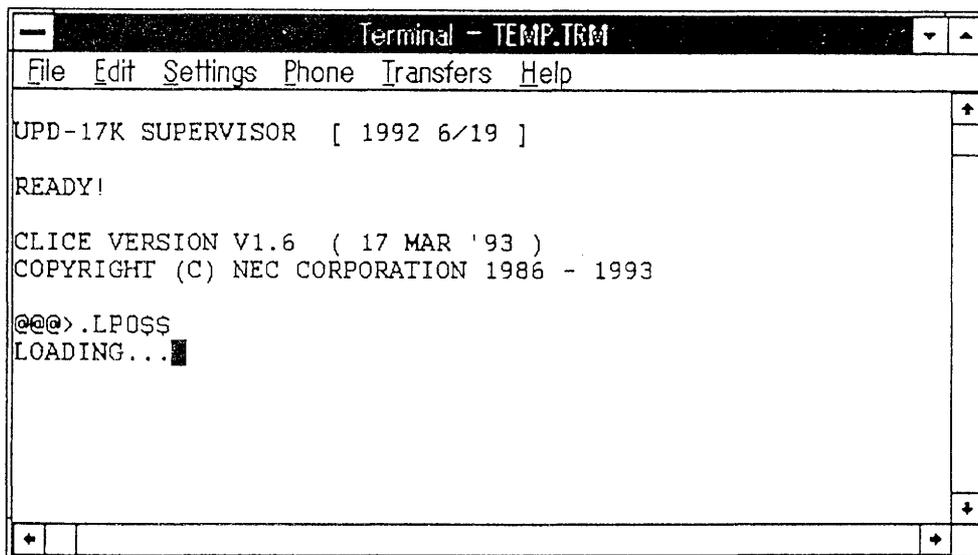
When the @@@> prompt has been returned properly, use the keyboard to input .LPO\$\$ after the @@@> prompt, to set the IE-17K to wait for loading. This condition is shown in Figure 4-4.

Figure 4-4 Loading Wait Status

(a) PC-9800 Series



(b) IBM PC/AT

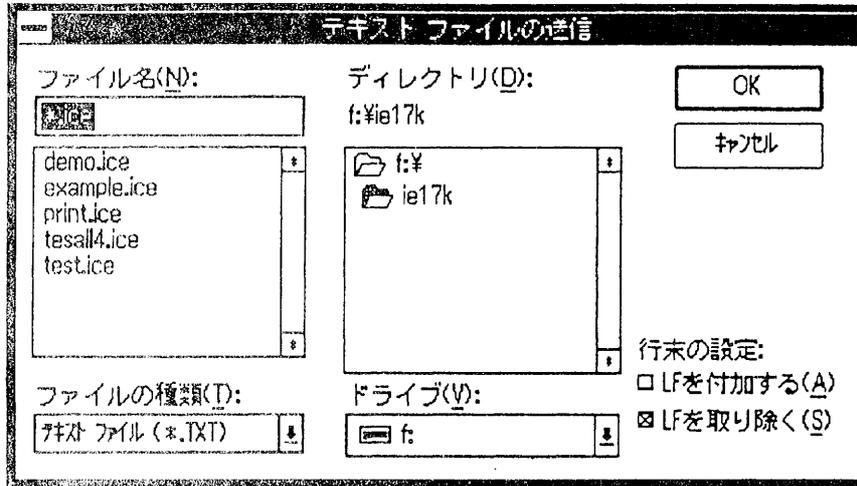


Next, select "Transfers" item from the menu bar in the "Terminal" window, and then select "Send Text File". From the "Send Text File" screen, input *.ICE in the "File Name". Search for the drive or directory having that file using "Drives" or "Directories", then select the file from the "File Name" list, or input the file name directly in the text box. After the file has been selected, click the "OK" button to start sending the file.

Figure 4-5 shows the "Send Text File" screen, and Figure 4-6 shows the screen during file transfer.

Figure 4-5 Sending of Text File

(a) PC-9800 Series



(b) IBM PC/AT

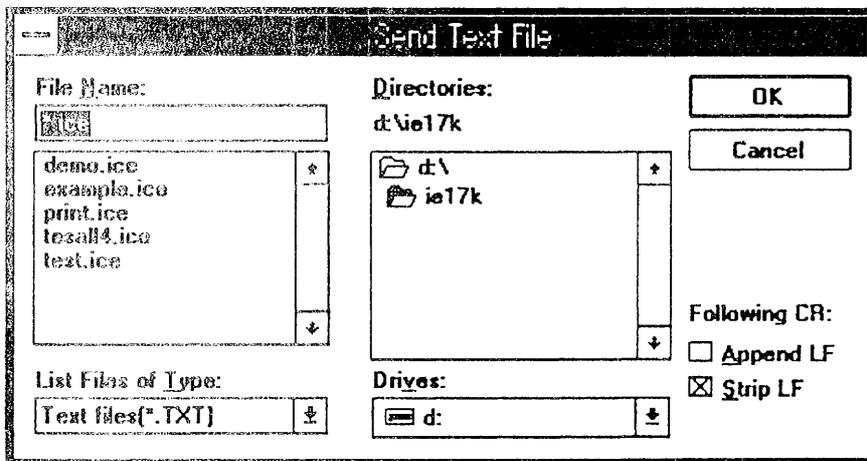
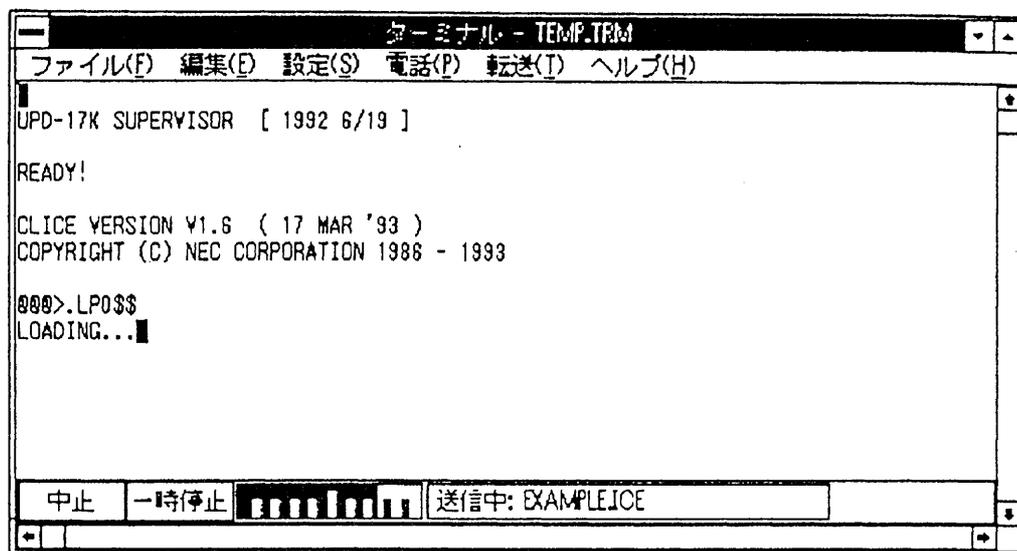
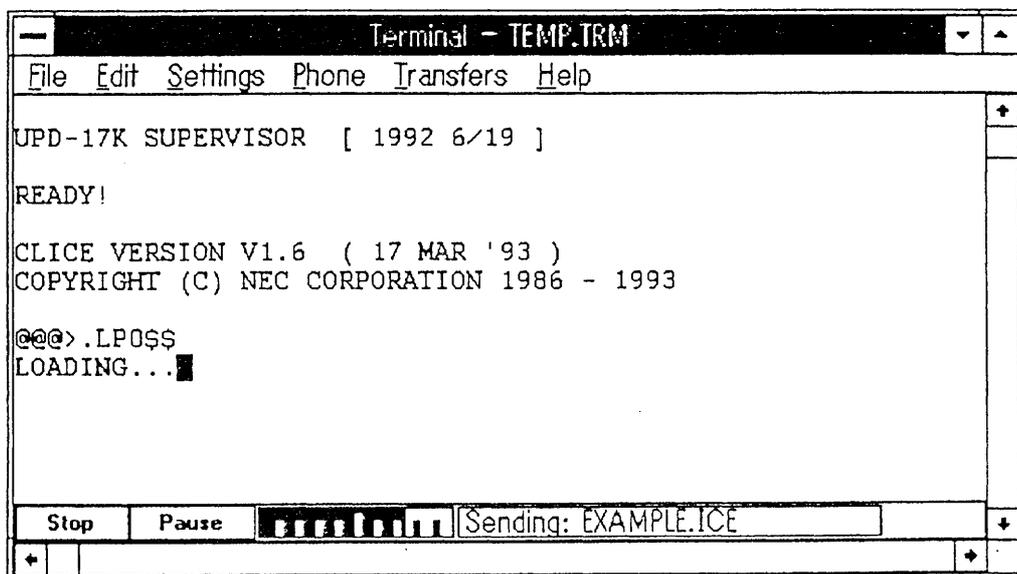


Figure 4-6 File Transfer

(a) PC-9800 Series



(b) IBM PC/AT

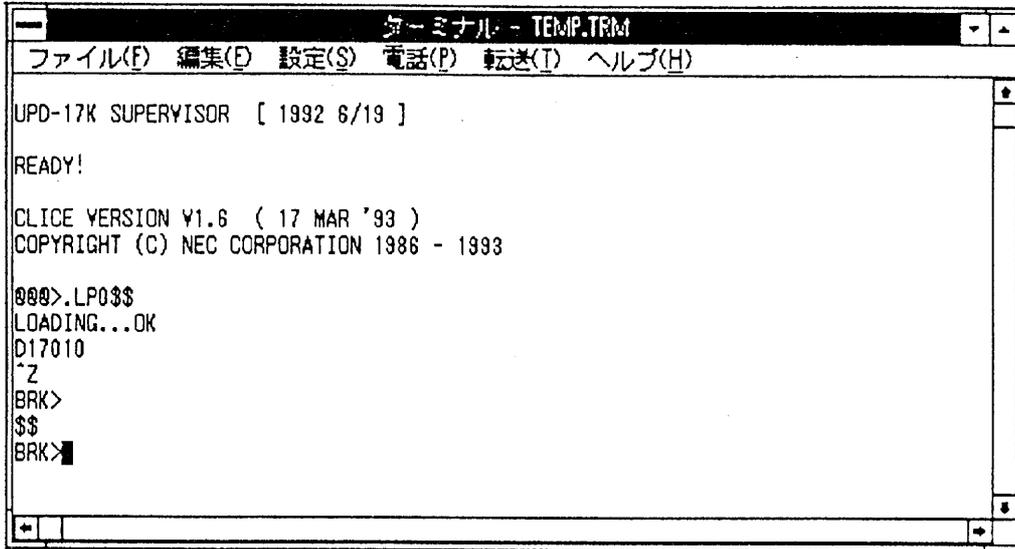


After the file has been transferred, the BRK> prompt is returned together with "OK" and the "Device Name". After this happens, press the ESC key two times or input \$\$, and make sure that the IE-17K is set to be able to receive commands. If the BRK> prompt is returned when the ESC key is pressed twice or when \$\$ is input, it means that the built-in IE-17K macro commands such as .R, or .RN can be used. For details about the built-in macro commands, see 5.8 "Built-In Macro Commands".

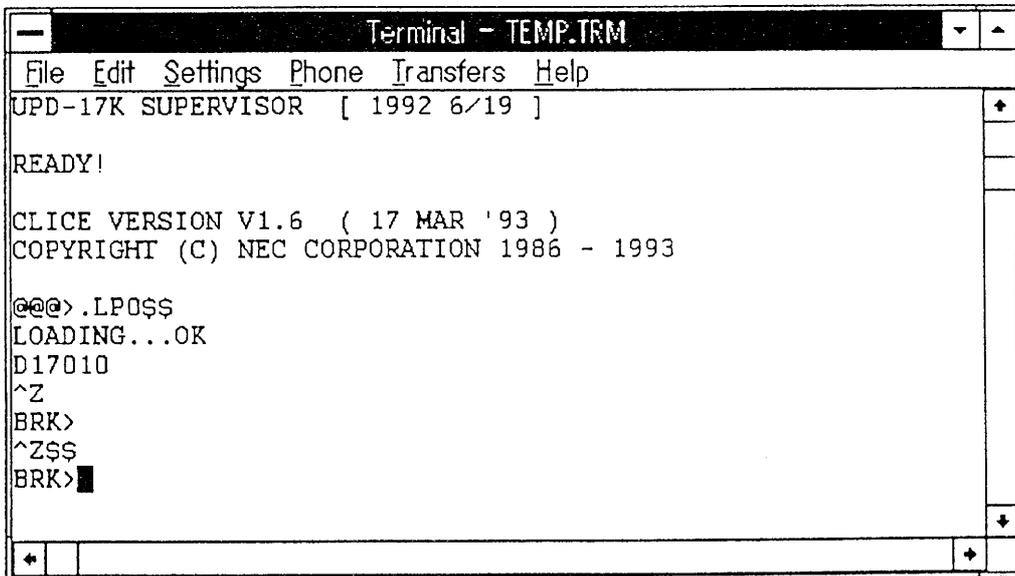
If the BRK> prompt is not returned, it is possible that power is not being supplied to this chip on the SE board, or the SE board may be reset. Therefore, refer to the SE board user's manual and check the settings.

Figure 4-7 File Transfer End

(a) PC-9800 Series



(b) IBM PC/AT



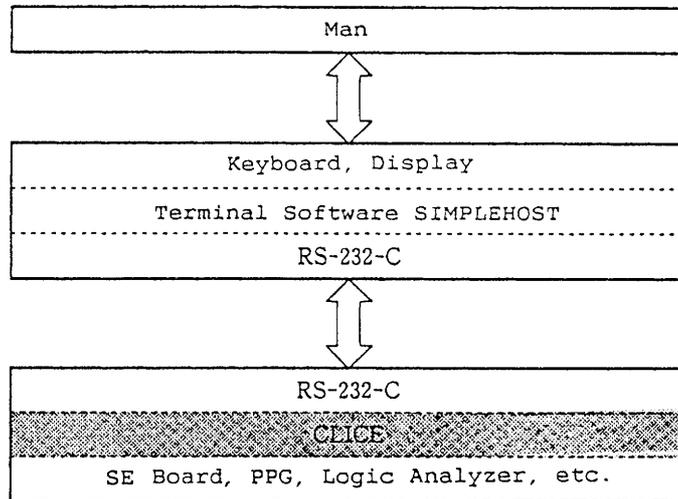
Phase-out/Discontinued

CHAPTER 5. DESCRIPTION OF COMMANDS

The IE-17K contains a command processing system known as CLICE (Command Language for In-Circuit Emulator).

This chapter explains the description, special characters, etc. usage conditions, and the detailed functions for all the commands supported by CLICE.

Figure 5-1 CLICE Positioning



There are two kinds of command, built-in macro commands and primitive commands.

Built-in macro commands are described by a period and two upper case alphabetic characters, such as [.AP]. Built-in macro commands are used when using the basic functions of the IE-17K.

Primitive commands are a group of commands which offer a more advanced debugging method for those with experience in program development using the basic functions of the IE-17K (see Appendix A "Primitive Commands").

When the IE-17K power is turned on and CLICE is started, the CLICE title, version No., etc. are displayed at the console as shown below.

```
UPD-17K SUPERVISOR [ **** **/** ]
```

```
READY!
```

```
CLICE VERSION V*.* (** *** ****)
```

```
COPYRIGHT (C) NEC CORPORATION **** - ****
```

```
@@@>
```

*(asterisk) indicates version No. and date.

5.1 PROMPT

The CLICE prompts show the status of the IE-17K target device. It also shows that key input is possible. The prompts indicate the following states.

- ① @@@> ... Waiting to load HEX file at IE-17K starting
- ② BRK> ... Target device stopped
- ③ RUN> ... Target device running
- ④ STP> ... Target device executed a STOP instruction
- ⑤ HLT> ... Target device executed a HALT instruction
- ⑥ DMA> ... Target device is running in the DMA mode
- ⑦ DSP>DS command is executing
- ⑧ RES> ... Reset signal is input to target device

[Note]

- (1) When prompt ① is displayed, that is, when the IE-17K is started, the model used is set at the IE-17K, therefore, the program must be loaded by restarting the IE-17K by .Q command or by entering an .LP0 or .LP1 command.
- (2) When the status of the target device changes during command input (prompt changes from RUN to BRK, etc.), the command string input up to that point is output and command input is accepted after the new prompt.

(Example) When prompt changes from RUN to BRK while [0,100.DP\$\$] is being input

```
RUN> 0,100      ..... Prompt changes  
                                from RUN to BRK  
                                here.
```

```
BRK> 0,100.DP$$ .. CLICE outputs 100.
```

In the example above, since the status of the target device changes from RUN to BRK when 0,100 is input, line feed is performed automatically and the prompt shows the new state and the command string 0,100 is output and the system waits for key input.

- (3) Once the prompt changes from RUN> to STP>, HLT>, DMA>, or RES>, the prompt does not automatically return to the original prompt even if that state is reset. In this case, the prompt changes the next time \$\$ is input.

5.2 COMMAND LINE FORMAT

Commands are input in the following format.

```
xxx> command $$
```

xxx> part is called "prompt" and shows the IE-17K operation.

A command is executed by entering the command after the prompt and striking the `ESC` or `$` key two times. (When the `ESC` key is input, "\$" is echoed back.)

The two \$ symbols input after the command indicate the end of the command and are called "terminator". When the terminator is input, the IE-17K starts execution of that command.

Multiple commands can be input consecutively by separating them from each other by a delimiter (\$) as shown below.

```
xxx> command1$command2$ ..... $commandn$$
```

Some commands require a delimiter as a delimiting symbol and other commands do not. \$ (ESC code) is used as the command delimiter.

One \$ input after a command that does not require a delimiter has no affect on execution of the command.

When multiple commands which do not require a delimiter are described, \$ can be inserted between the commands to facilitate reading of the command string.

A command string is executed by input of two consecutive \$. In short, input of two consecutive \$ ends the command lines and starts command execution.

A command string can be corrected before it is terminated.

There are commands with arguments and commands without arguments.

The basic command formats are shown below.

- ① <numeric argument><command name>
- ② <numeric argument><command name><Q-register identifier>
- ③ <command name><Q-register identifier><character string>\$
- ④ <command name><Q-register identifier>
- ⑤ <command name>
- ⑥ <numeric argument><command name><character string>\$

An expression can also be described where a numeric argument is described in all commands.

5.3 COMMAND BUFFER

CLICE has a command buffer which stores the command input from the console.

In most cases, the input characters are stored in the command buffer directly.

However, in the following cases, the input characters are not stored directly.

- ① Special control characters
(See 5.4.1 "Special Control Characters".)
- ② * and ? input immediately after the prompt
(See "* command (assignment to Q-register)", "? command (error display)" in Appendix A.5.9 "Others".)

When command execution ends normally and the prompt is displayed, the command buffer is cleared.

5.4 CHARACTER SET

The ASCII character set can be used with CLICE.

The control characters (ASCII codes 00 to 1FH) from control A (↑A) to control _ (↑^) are displayed by ^ and an alphabetic character.

CLICE interprets the two characters ^ and A as if control A (↑A) were input even if they are input consecutively.

(Example) Pressing the A key while pressing the control key is called control A (abbreviated ↑A). At this time, ASCII code 01H is input at CLICE and the two characters ^ (ASCII code 5EH) and A (ASCII code 41H) are displayed as ^A.

5.4.1 SPECIAL CONTROL CHARACTERS

CLICE has the following special control characters.

(1) DEL

When the DEL (ASCII code 7FH) is input, the previous character is deleted.

When the previous character is a control character, the two characters ^ and alphabetic character are deleted.

(2) ESC

When the ESC (ASCII code 1BH) key is input, \$ (ASCII code 24H) is displayed.

1BH is stored in the command buffer.

(3) Control C

Control C (\uparrow C)(ASCII code 03H) is a special control character for interrupt command execution. The two characters \wedge and C are displayed in that order.

When \uparrow C is input before a command string is terminated, the entire input command string becomes invalid and the prompts displayed and the system waits for command input.

However, in this case, the data in the command buffer is not cleared. (See "* command (assignment to Q-register)" in Appendix A.5.9 "Others".)

To interrupt execution after a command is terminated (i.e., after command execution starts), \uparrow C is input twice.

When execution of a command string is interrupted, the following message is displayed.

ABORTED!

At this time, the interrupted command string is saved in the command buffer.

(4) Control X

The control X (\uparrow X)(ASCII code 18H) key deletes the line containing the cursor.

The command string deleted by \uparrow X is not saved in the command buffer.

(5) Control H

The control H (↑H) and BS keys (ASCII code 08H) perform the same operation as the DEL key.

(6) Control U

Control U (↑U)(ASC code 15H) is functionally the same as ↑X, but displays ^U and line feed.

↑U is the command deletion control character for TTY type consoles other than CRT.

(7) Control G

When control G (↑G)(ASCII code 07H) is input, the contents of the current line are displayed.

Similar to ↑U, ↑G this is a control character for a TTY type console.

(8) Control E

When control E (↑E)(ASCII code 05H) is input before a command string is terminated, the IE-17K enters the edit mode (See Appendix A.6 "Editor".)

(9) Control S

When control S (↑S)(ASCII code 13H) is input, character display of the command executing at that time is temporarily interrupted.

Execution is resumed by entering control Q (↑Q) (ASCII code 11H).

5.4.2 SPECIAL CHARACTERS

CLICE has the following special characters.

(1) ^

^ (ASCII code 5EH) is always followed by a character.

For the character following ^, the same character as a control character is stored to the command buffer.

(Example) When B is input from the cursor following ^, the same code as when ↑B is input (ASCII code 02H) is stored to the command buffer.

(2) \$

When \$ (ASCII code 24H) is input, ESC (ASCII code 1BH) is stored to the command buffer.

(3) Control R

For control R (↑R)(ASCII code 12H), the character following ↑R is stored to the command buffer unchanged.

This is used when you want to store a character which only performs special control character, control character, etc. operation to the command buffer.

(Example) ^R ↑R ... ↑R (ASCII code 12H) is stored as one character.

^R^R ... The two characters ^ (ASCII code 5EH) and R (ASCII code 52H) are stored.

↑R ↑R ... ↑R (ASCII code 12H) is stored
as one character.

↑R^R ... The two characters ^ (ASCII
code 5EH) and R (ASCII code
52H) are stores.

However, ↑R does not apply to ↑C and DEL. These
characters can only be stored by editor command.

(4) Other special control characters

Besides the special control characters, CLICE has the
following characters.

Control B (↑B), control D (↑D), control P (↑P),
. (period).

Of these, ↑B, ↑D, and ↑P are prefixes which show
the type of constant to be described later.

. (period) is a prefix which identifies a built-in
command.

5.4.3 DUMMY CHARACTER

"Dummy character" is a character which is not defined as a command itself, but does not generate an error even if executed.

There are dummy characters that perform a special operation when they are output to the console.

Therefore, dummy characters can be inserted into a character string to simplify reading of the character string.

(1) CR

When CR (↑M)(ASCII code 0DH) is output to the console, line feed is performed.

(2) LF

LF (↑J)(ASCII code 0AH) is not stored to the command buffer unchanged.

To store LF to the command buffer, input ↑J after ↑R.

(3) Blank

When blank (ASCII code 20H) is output to the console, a one character blank is displayed.

(4) NULL

NULL (↑@)(ASCII code 00H) is not stored to the command buffer.

To store NULL to the command buffer, input ↑@ after ↑R.

(5) TAB

TAB (↑I)(ASCII code 09H) displays 8 columns of blanks.

↑I (ASCII code 09H) is stored to the command buffer.

5.5 EXPRESSION

The operators that can be used in expressions by CLICE are shown below.

+ ... + sign or addition symbol
 - ... - sign or subtraction symbol
 * ... Multiplication symbol
 / ... Division symbol
 ^^ ... Remainder symbol
 & ... AND symbol
 # ... OR symbol
 | ... Exclusive-OR symbol
 ~ ... Negate symbol
 { ... Left shift symbol
 } ... Right shift symbol

Expressions are all evaluated not in expression operation priority order, but from left to right.

However, when desiring to change the priority, () can be used.

Constants, variables, and functions can be used as the component element of each item of an expression.

(Example) The evaluated value (right side of →) of the examples shown below are all shown in decimal.

①	xxx>3+4	→ 7
②	xxx>10+ ↑D10* ↑B10	→ 52
③	xxx>10+ (↑D10* ↑B10)	→ 36
④	xxx> ↑B1010} 1	→ 5
⑤	xxx> ↑B1011} 1& ↑B1000	→ 8
⑥	xxx> ~FFFFFFC	→ 3
⑦	xxx>5^^3+ (5/3)	→ 3
⑧	xxx>5^^3+5/3	→ 2

- ⑨ `xxx>Q1+@ ↑FDTM ↑V` → Sum of contents of Q-register 1 and array assigned to top address of data memory (that is, contents of data memory address 0)
- ⑩ `xxx>_ ↑FPRM+ (100*2) ↑V` → Contents (16 bits) of program memory address 100H

5.6 CONSTANT

Hexadecimal, decimal, and binary integers and 17K series instruction identifiers (1-4-3-4-4 bit format) can be used with CLICE to represent constants.

The range of values which can be represented is,

decimal : -2^{31} to $(+2^{31}-1)$

hexadecimal : 0 to FFFFFFFFH

Two's complement representation is used as negative representation.

The constants representation method is shown below.

Binary constant: Represented by adding ↑B in front of the binary number

(Example) ↑B1010 represents the decimal number 10.

Decimal constant : Represented by adding ↑D in front of the decimal number.

(Example) ↑D324 represents the decimal number 324.

Hexadecimal constant : Represented by decimal number only.

(Example) F1 represents the decimal number 241.

1-4-3-4-4 bit format constant : Represented by adding ↑P in front of the hexadecimal number.

(Example) ↑P074F0 represents NOP.

5.7 VARIABLES

CLICE has arrays and Q-registers as the variable concept.

5.7.1 ARRAY

"Array" represents all the resources managed by CLICE.

Each element of an array corresponds to a target device resource.

Therefore, when a value is assigned to an array by CLICE, data is written to the hardware in the target device corresponding to the element of that array.

When the contents of an array element are referenced, the data (status) of the corresponding hardware at that array element is read.

Each element of an array can be referenced by pointer and data can be assigned by XB, XC, and XW command.

5.7.2 Q-REGISTER

CLICE has a registers (Q-registers) that can store a value or character string. This register corresponds to the variable concept of general programming languages.

The Q-registers are described by the character Q followed by an upper case alphabetic character or numeric. (The upper case alphabetic character is called the Q-register identifier.)

(Examples) Q1, QA, Q8, QZ, etc.

All the Q-registers can be used as numeric variable or character variable.

The kind of variable a Q-register is used as is determined by the contents stored in it.

When CLICE starts, all the Q-registers are cleared.

The range of values which can be stored in the Q-registers is the same as the range of values of constants. When a value is assigned to a Q-register, that value can be used by an expression.

A character string is stored to the Q-registers by U command and * command. When a character string is stored to a Q-register, that character string can be executed as a macro command.

5.8 BUILT-IN MACRO COMMANDS

This section describes the built-in macro commands which are used when executing the basic functions of the IE-17K.

The symbols used in the formats described in this section are defined below.

↵ : Line feed input
{ } : Indicates that one of the character strings described in the { } is to be selected.
[] : Input can be omitted
 (Underline) : Console input

5.8.1 PROGRAM MEMORY CONTROL COMMANDS

The commands are listed in order of the actual procedures.

- (1) Program memory load
 .LP (Load Program Memory)
- (2) Program memory verify
 .VP (Verify Program Memory)
- (3) Program memory initialize
 .IP (Initialize Program Memory)
- (4) Program memory modification
 .CP (Change Program Memory)
- (5) Assemble command
 .AP (Assemble Program)
- (6) Program memory dump
 .DP (Dump Program Memory)

- (7) Disassemble command
.UP

- (8) Program memory search
.FP (Find Program Memory)

- (9) Program memory save
.SP (Save Program Memory)

- (10) PROM data output
.XS (Save PROM Date)

- (11) IE-17K restarting
.Q

.LP0 .LP1 Load Program Memory

Format : { .LP0 } { .LP1 }

RS-232-C channel 0: LP0 channel 1: LP1

[Function] Inputs the contents of an AS17K ICE file from the RS-232-C channel specified by .LP0 or .LP1.

(Example) Load the program from channel 0.

@@@> .LP0\$\$

[Notes] . When the power is turned on, or when the IE-17K is reset (prompt @@@>), load the AS17K ICE file by .LP.

. When the program loaded by this command occupies only a part of the program memory, the previous program remains at the unloaded part of program memory.

. The program coverage is cleared.

.VPO .VP1 Verify Program Memory

Format : { .VPO } { .VP1 }

RS-232-C channel 0: VPO channel 1: VP1

[Function] Verifies the contents of program memory and the data of the AS17K ICE file sent from the RS-232-C channel specified by .VPO or .VP1.

When verifying, "Verify...NG" is always displayed for areas outside the user program. This is because in the memory where the ICE file is stored, the IE-17K processes data in a portion of the assembly environment information area and in the SE board environment information area.

Refer to Chapter 5 "Load Module File Format", of the device file user's manual for details about the assembly environment information area and the SE board environment information area, and check to make sure that the address where the error occurred is outside the user program area.

(Example)

```
BRK>.VPO$$
Verify ... OK
00
BRK>000000000030303231313011
:1007C40038303731441212FF073356202020F0F6
:1007D40000000000.....
:0407FC000100C11522
:00000001FF
```

- [Note] . When the data memory information is different, "VERIFY NG DATA INITIAL VALUE" is displayed.
- . When EPA is different, "Verify NG EPA" is displayed.
 - . When IFL and DFL are different, "Verify NG IFL DFL" is displayed.

.IP Initialize Program Memory

Format : [α], β , γ .IP

α : Start address

β : End address

($\alpha \leq \beta$, $\alpha > \beta$ generates an error)

γ : Initialize data (1-4-3-4-4 bit format)

[Function] Initializes the contents of program memory addresses α to β to γ .

When α is 0, α can be omitted.

(Example 1) Initialize addresses 10H to 20H to 074F0.

BRK>10,20,074F0.IP\$\$

(Example 2) Initialize addresses 0H to 20H to 120FF.

BRK>,20,120FF.IP\$\$

.CP Change Program Memory

Format : [α].CP

α : Program memory address to be changed

[Function] Changes the contents of program memory address α .

When α is 0, α can be omitted.

(Example) Change the program contents beginning from address 100.

BRK>100.CP\$\$

0100:074F0-120F5 074F0-14001

074F0-11000 074F0-06100

0104:074F0- ↵

↳ \$\$ also possible
in exchange for ↵

When a 14344 format value is input up to 5 digits, the cursor automatically moves to the next address.

To end operation, input "↵" or "\$\$" instead of a value.

BRK>100.CP\$\$

0100:074F0- SPC

Space key pressed

0100:074F0-074F0 074F0-■

Input wait of next
address

When the space key is input instead of a value, the program contents are not changed and the cursor moves to the next address.

If the wrong value is input, it can be corrected with the "DEL" key. (This also applies to the "BS" key.)

0100:120AF-120A1 074F0-120_

"DEL" key pressed.

↓

0100:120AF-120A1 074F0-12_

"DEL" key pressed.

↓

0100:120AF-120A1 074F0-1_

"DEL" key pressed.

↓

0100:120AF-120AF 074F0-__

"DEL" key pressed.

↓

0100:120AF-__

"DEL" key pressed.

↓

00FF:120C1-__

Remarks __: Cursor

.AP Assemble Command

Format : [α].AP[β]

α : Start address

β : Q-register name

[Function] Assembles the mnemonic applied to Q-register β and stores it beginning from program memory address α .

When α is 0, α can be omitted.

When Q-register β is omitted, the code of program memory address α is reverse assembled and displayed and the IE-17K enters the mnemonic input mode. In this mnemonic input mode, the program memory contents can be changed in the mnemonic level.

(Example 1)

BRK>5.AP\$\$

0005: MOV 05,#5 - \$\$ Quits mnemonic

BRK> input mode by \$\$
input

BRK>.AP\$\$

0000: MOV 00,#1 - MOV 01,#1 ↵

Assembles by ↵
input and goes to
next address (BS
key can be used)

0001: MOV 10,#2 - ■

↳ Input wait

BRK>.AP\$\$

0000: MOV 00,#1 - MOV 01,#1 ↵

ASSEMBLE ERROR

If assembly error occurs, returns to original address and waits for input

0000: MOV 00,#1 - ■

↳ Input wait

BRK>.AP\$\$

0000: MOV 00,#1 - ↵ Only ↵ input results in next address

0001: MOV 10,#2 - ■ input wait

↳ Input wait

BRK>.AP\$\$

0000: MOV 00,#1 - MOV 01,#1

In input wait state, ↑P input displays string input for the last time (MOV 01,#1)

0001: MOV 10,#2 - MOV 01,#1 ■

↳ ↑P (CTRL + P)

(Example 2)

BRK>USMOV 01,#05

Assign character string to Q-register S by U command.

ADD 1,78

\$\$

BRK>5.APS\$\$

Assemble contents of Q-register S.

BRK>5,6.DP\$\$

Dump assembled result.

0005: 1D015 00781

- [Note] (1) Describe RF addresses by 40H to BFH.
However, 00H to 3FH, when written, it treated
the same as 80H to BFH. C00H to FFH is
treated as 40H to 7FH.
- (2) Use space or tab as the mnemonic and operand
separator.
- (3) Describe addresses in hexadecimal numbers.
- (4) Describe immediate data with the symbol #
followed by a hexadecimal number.

```
(Example):  MOV    11,#0      STOP  0
            ADD    0,11      BR     005F
            POKE   81,WR     BR     @AR
            PUT    01,DBF    MOV    @5,00
```

- (5) When an error is generated during assembly,
the error line and its contents are displayed
and assembly stops. At this time, the codes
up to the line before the error line are
stored to program memory. The operand range
is not checked.
- (6) When the assembly contents exceed the last
program memory address, storage is continued
from address 0.
To use EPA, specifies address 8000H or a
subsequent address.

(Example)

```
: 8000.APD$$
Assemble and store contents of
Q-register D at EPA.
```

.DP Dump Program Memory

Format : [α][, β].DP

α : Start address

β : End address ($\alpha \leq \beta$, $\alpha > \beta$: error)

[Function] Dumps the program contents of addresses α to β .

When α is 0, α can be omitted. If ", β " is omitted, the end address becomes $\alpha + 3FH$.

(Example 1) Dump the contents of addressed 10H to 20H in 1-4-3-4-4 bit format.

BRK>10,20.DPSS

```
0010:074F0 074F0 074F0 074F0
      074F0 074F0 074F0 074F0
0018:074F0 074F0 074F0 074F0
      074F0 074F0 074F0 074F0
0020:074F0
```

*(Example 2) Dump the contents of addressed 0 to 10H.

BRK> 10.DP\$\$

```
0000:074F0 074F0 074F0 074F0
      074F0 074F0 074F0 074F0
0008:074F0 074F0 074F0 074F0
      074F0 074F0 074F0 074F0
0010:074F0
```

(Example 3) Dump the contents beginning from address 10H.

(Dump addresses 10H to 10H+3FH.)

BRK> 10.DP\$\$

```
0010:1D790 1D7D0 1D7E0 074F0
      074F0 074F0 074F0 167E0
0018:1D770 08770 10771 08771
      10771 08772 10771 08773
0020:10771 08774 10771 08775
      10771 08776 10771 08777
0028:10771 08778 10771 08779
      10771 0877A 10771 0877B
0030:10771 0877C 10771 0877D
      10771 0877E 10771 0877F
0038:1D000 074F0 074F0 074F0
      074F0 074F0 074F0 074F0
0040:1D7F0 00000 074F0 074F0
      0B7D0 097E0 0C049 1C146
0048:0C050 097F2 0C170 09000
      0C171 18770 09770 0C172
```

.UP Reverse Assemble Command

Format : [α][, β].UP [γ]

α : Start address
 β : End address ($\alpha \leq \beta$, $\alpha > \beta$: Error)
 γ : Q-register name

[Function] When γ is omitted, this command reverse assembles and displays the contents of program memory addresses α to β . At this time, EPA information is also output.
 When γ is specified, this command reverse assembles the contents of program memory addresses α to β and stores the mnemonics to Q-register γ .
 When α is 0, α can be omitted. When ", β " is omitted, the end address becomes $\alpha + 10$.

(Example 1)

BRK> .UP\$\$ When both start address and end omitted.

EPA	ADDR	CODE	MNEMONIC
	0000	070E0	RET
	0001	007F0	ADD 0,7F
	0002	002A5	ADD 5,2A
	0003	00558	ADD 8,55
	0004	0000F	ADD F,00
	0005	1000F	ADD 00,#F
	0006	102A5	ADD 2A,#5
	0007	1055A	ADD 55,#A
	0008	107F0	ADD 7F,#0

0009 027F0 ADDC 0,7F
BRK>8.UP\$\$ When only start address
specified.

EPA	ADDR	CODE	MNEMONIC
	0008	007F0	ADD 0,7F
	0009	002A5	ADD 5,2A
	000A	00558	ADD 8,55
	000B	0000F	ADD F,00
	000C	1000F	ADD 00,#F
	000D	102A5	ADD 2A,#5
	000E	1055A	ADD 55,#A
	000F	107F0	ADD 7F,#0
	000E	1055A	ADD 55,#A
	000F	107F0	ADD 7F,#0

BRK>44,4B.UP\$\$ When both start address and
end address specified.

EPA	ADDR	CODE	MNEMONIC
	0044	057F0	XOR 0,7F
1	0045	052A5	XOR 5,2A
1	0046	05558	XOR 8,55
1	0047	0500F	XOR F,00
	0048	1500F	XOR 00,#F
	0049	152A5	XOR 2A,#5
	004A	1555A	XOR 55,#A
	004B	157F0	XOR 7F,#0

(Example 2)

BRK>10,20.UPB\$\$ Reverse assemble contents
of program memory addresses
10H to 20H and load result
into Q-register B.

BRK>50.APB\$\$ Assemble contents of Q-
register B and expand at
program memory address 50H
and subsequent address.

(Example 3)

BRK>10,20.UPB\$\$

Reverse assemble contents of program memory addresses 10H to 20H and load result into Q-register B.

BRK>.EDB\$\$

Edit by edit command.

>

.

.

(Editing example omitted)

.

.

>

BRK>10.APB\$\$

Assemble contents of Q-register B and expand at program memory address 10H and subsequent addresses.

[Note] Codes which cannot be reverse assembled are displayed at the mnemonic field as [DW].

.FP Find Program Memory

Format : [α], β , γ [, δ].FP

α : Start address
 β : End address
 γ : Data to be searched
 δ : Mask data
(γ , δ : 1-4-3-4-4 bit format)

[Function] Searches the contents of γ masked by δ at the program memory contents from addresses α to β .

When α is 0, α can be omitted.

When δ is omitted, the mask data becomes 1F7FF.

(Example) Find if 12xxx is in addressed 0 to 300H.

BRK>0,300,12000,1F000.FP\$\$

0110:12120 0120:12200
0140:12240 0152:12250
0160:12151 0180:12152

[Note] "Mask data" is 1-4-3-4-4 bit format data with 1 set in the bit to be searched and 0 set in bits which can be 1 or 0.

.SP0 .SP1 Save Program Memory

Format : { .SP0 } { .SP1 }
RS-232-C channel 0: SP0 channel 1: SP1

[Function] Outputs the contents of program memory to the RS-232-C channel specified by .SP0 or .SP1.

The output format is the same as the AS17K ICE file format.

(Example) Output the contents of program memory to channel 1.

```
BRK> .SP1$$  
:1000000063A03CF061273CF0EFE040423CE0EF04AD  
:10001000EF10EF20EF30EF91EF00EF10EF20EF3017  
:10002000EF90E8C0E8D0E8D0E8F03CA138A538A6B9  
:1000300038A738E0E820E8303CF03CF080219030F0  
:10004000F7F4601C38E0B204E8E0E8F038A538E0E6  
:10005000E830E82038E08031902038E0F6F4605055
```

)

.XS0 .XS1 Save PROM Data

Format : { .XS0 } { .XS1 }

RS-232-C channel 0: XS0 channel 1: XS1

[Function] Outputs the contents of program memory in AS17K PROM file format at the RS-232-C channel specified by .XS0 or .XS1.

(Example) Output the contents of program memory at channel 1.

BRK> .XS1\$\$

:1000000063A03CF061273CF0EF040423CE0EF04AD
:10001000EF10EF20EF30EF91EF00EF10EF20EF3017
:10002000EF90E8C0E8D0E8E0E8F03CA138A538A6B9
:1000300038A738E0E820E8303CF03CF080219030F0
:10004000F7F4601C38E0B204E8E0E8F038A538E0E6
:10005000E830E82038E08031902038E0F6F4605055

)

.Q Restart IE-17K

Format : .Q

[Function] Restarts the IE-17K from the previous state when the IE-17K reset switch is pressed and when the power is turned off and then turned back on. When restarting is successful, the same program as before need not be loaded.

(Example) @@@>.Q\$\$
17XXX
BRK>

. Product name is displayed 2 to 3 seconds after .Q command is input.

[Note] If the product name is not displayed 2 to 3 seconds after the .Q command is executed, the program will be lost. Therefore, after the IE-17K is reset, reload the program.

Since the program is also lost when the message

Your program must be lost.
Please load it again!

is displayed instead of the product name, reload the program after resetting the IE-17K.

Whether the program is preserved or lost is judged by whether or not the program checksum is matched.

Therefore, part of the program may be lost even if the product name is displayed.

The time at which the program is preserved after the IE-17K power is turned off depends on the SE board.

5.8.2 DATA MEMORY CONTROL COMMANDS

- (1) Data memory initialize
.ID (Initialize Data Memory)

- (2) Date memory change
.CD (Change Data Memory)

- (3) Data memory dump
.DD (Dump Data Memory)

- (4) All data memory dump
.D (Dump All Data Memory)

.ID Initialize Data Memory

Format : [α], β , γ .ID

α : Start address

β : End address ($\alpha \leq \beta$, $\alpha > \beta$: Error)

γ : Contents

[Function] Initializes the contents of data memory address α to β to γ .

When α is 0, α can be omitted.

(Example 1) Initialize contents of addressed 10H to 20H to 0.

BRK>10,20,0.ID\$\$

(Example 2) Initialize contents of addressed 0 to 20H to 1.

BRK>,20,1.ID\$\$

.CD Change Data Memory

Format : [α].CD

α : Data Memory address to be changed

[Function] Change the contents of data memory address α .
When α is 0, can be omitted.

(Example) Change the contents from data
address 0.

BRK> .CD\$\$

0000 0-0 1-0 2-0 3-0 4-0 5-0 6-0 7-0
0008 8-0 9- ↵

↳ \$\$ also possible in
exchange for ↵

When one data is input, the cursor moves
address. To end operation, input " | " or "\$\$" instead of a value.

BRK> 100.CD\$\$

0100 3- SPC

Space key pressed

0100 3-3 2- ■

Input wait of next
address

When the space key is input instead of a
value, the data contents are not changed and
the cursor moves to the next address.

If the wrong value is input, it can be corrected with the "DEL" key. (This also applies to the "BS" key.)

0010:2-3 4-5 6-__ "DEL" key pressed.

↓

0010:2-3 5-__ "DEL" key pressed.

↓

0010:3-__ "DEL" key pressed.

↓

000F:4-__

Remarks __: Cursor

.DD Dump Data Memory

Format : [α][, β].DD

α : Start address

β : End address ($\alpha \leq \beta$, $\alpha > \beta$: Error)

[Function] Dumps the contents of data memory addresses α to β .

When α is 0, α can be omitted.

(Example 1) Dump the contents of data memory addressed 0 to 80H.

BRK>0,80.DD

```
0000:0 1 2 3 4 5 6 7 8 9 A B C D E F
0010:0 1 2 3 4 5 6 7 8 9 A B C D E F
0020:0 1 2 3 4 5 6 7 8 9 A B C D E F
0030:0 1 2 3 4 5 6 7 8 9 A B C D E F
0040:0 1 2 3 4 5 6 7 8 9 A B C D E F
0050:0 1 2 3 4 5 6 7 8 9 A B C D E F
0060:0 1 2 3 4 5 6 7 8 9 A B C D E F
0070:0 1 2 3 4 5 6 7 8 9 A B C D E F
0080:0
```

(Example 2) Dump the contents of address 30H.
(Dump the contents from address 30H to address 7FH.)

BRK>30.DD\$\$

```
0030:0 0 0 3 0 5 0 7 8 9 0 1 4 6 0 F
0040:0 1 2 3 4 5 6 0 8 3 A B C D E F
0050:0 0 2 D F F F 0 4 9 A 0 C 0 0 F
0060:0 1 2 3 4 5 6 0 8 9 A 0 0 D 0 F
0070:0 C 2 B 4 5 6 0 8 7 9 B C 0 0 0
```

- [Note] . When " α " is omitted, the data from address α to the last address of the bank allocated address α is dumped.
- When address α of a register file is specified, the data from address α to the end of the register file is dumped.
- . The contents of uninstalled data memory are indicated by "-".
 - . Dump of addressed 0080 to 00BF dumps the contents of the register file. If the register file is not installed, the status of the internal bus is displayed.

.D Dump All Data Memory

Format : .D

[Function] Dumps all the data memory contents.

(Example) BRK> .D\$\$

```
0000:0 0 1 C 6 0 0 0 0 0 0 0 0 0 0 0
0010:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0020:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0030:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0040:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0050:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0060:4 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0
0070:5 0 F 0 0 0 0 0 1 0 0 7 0 0 C 0

0080:2 5 2 3 4 5 6 1 7 A A B C D E 1
0090:0 0 0 3 4 0 6 0 0 9 A B C D E 0
00A0:0 1 2 3 4 5 6 1 0 9 A B C D E 0
00B0:0 0 2 3 4 7 7 F 0 0 A B C D E 2
```

)

[Note] Register file is also dumped.

5.8.3 PERIPHERAL CIRCUIT CONTROL COMMANDS

- (1) Peripheral register contents display
.GD (Get & Display)

- (2) Peripheral register contents read
.GE (GET)

- (3) Write to peripheral register
.PD (Put Direct)

- (4) Indirect write to peripheral register
.PU (PUT)

.GD Get & Display command

Format : α .GD

α : Peripheral address

[Function] Display the contents of peripheral address α in hexadecimal.

(Example 1)

```
BRK>1.GD$$                   Display contents of  
00000002                   peripheral address 1.
```

(Example 2)

```
BRK>UP1.GD^A TIME1 ^A2.GD^A TIME2 ^A3.GD^A SIO  
^A$$                   Define macro P.  
BRK>MP$$               Execute macro P.  
00000034 TIME1  
00000022 TIME2  
00000004 SIO
```

[Note] When peripheral address α is not this model, the following message is displayed.

?POS INVALID ADDRESS

.GE Get Command

Format : α .GE β
α : Peripheral address β : Q-register name

[Function] Assigns the value of peripheral address α to Q-register β by numeric.

(Example 1)

```
BRK>1.GEA$$      Assigns the value of peripheral
                 address 1 to Q-register A.
BRK>QA=H$$10      Displays contents of Q-register
                 A in hexadecimal.
```

(Example 2)

```
BRK>UQ1.GEA2.GEB3.GEC^A SIO= ^AQA=H^A
TM1= ^AQB=H^A TM2= ^AQC=H$$
                                 Define macro Q.
BRK>MQ$$ SIO= 34 TM1= 66 TM2= 2
                                 Execute macro Q.
BRK>
```

[Note] [\wedge A] in macro Q defined in Example 2 is an instruction to display character strings directly. If there is a character string you want to display at macro-defined command execution, enclose that character string in [\wedge A].

When peripheral address α is not a value at this model, the following message is displayed.

?POS INVALID ADDRESS

.PD Put Direct Command

Format : α , β .PD

α : Peripheral address

β : Data

[Function] Assigns numeric β to peripheral address α .

(Example) BRK>1,55.PD\$\$

Assign 55H to peripheral
address 1.

[Note] When peripheral address α is a value not at this
model, the following message is displayed.

?POS INVALID ADDRESS

5.8.4 EMULATION COMMANDS

- (1) Reset
.R (Reset)
- (2) Program execution
.RN (Run)
- (3) Program execution (reset condition)
.BG (Run Beginning Condition)
- (4) Break
.BK (Break)
- (5) Program start address change
.CA (Change Start Address)
- (6) Step operation
.S (Step)
- (7) Display
.DS (Display)

.R Reset

Format : .R

[Function] Resets the SE board.

(Example) BRK>.R\$\$

[Note] . The register file and data memory contents become the same as the reset contents of the target product.

. The data coverage contents are cleared.

. The run start address becomes 0H.

.RN Run

Format : .RN

[Function] Executes the program from the current specified program run state address.

The conditions used at break and trace do not change.

(Example) BRK> .RN\$\$
RUN>

.BG Run Beginning Condition

Format : .BG

[Function] Executes the program from the currently specified program run start address.

However, the following conditions used at break and trace are set.

<Reset contents>

- . Counter used at level 1 (reset value: 0)
- . Sequential stack used at level 2 (to initial value)
- . Trace on, tract one shot, and trace off specification (all to trace state)
- . Level 1 condition

(Example) BRK>.BG\$\$
RUN>

.BK Break

Format : .BK

[Function] Stop program execution.

Display the contents of the system registers and general registers at this time.

This command can be accepted in the break state also.

(Example) RUN> .BK\$\$

ADDR INSTRUCTION

```
0002 074F0 BREAK ..... Break processed
                                instruction
0003 074F0 OVERRUN ..... Instruction
                                executed last
0004 0C004 NEXT ..... Instruction to
                                be executed
                                next
```

```
PC SP AR WR BR MP IX
0004 3 0700 0 0 *** 000
PSW: DB CP CY Z IXE MPE JG
      0 0 0 0 0 0 0 0 ] System
                        registers
```

```
RP 0123456789ABCDEF
000 00000000000000320 ..... General
                                registers
```

.CA Change Start Address

Format : α .CA
α : Run start address

[Function] Changes the program run start address.

(Example) Change the program run start address to 100H.

BRK>100.CA\$\$

.S Step

Format : [α].S

α : Number of times

[Function] Runs the program that specified number of times.

(Example 1) Perform step operation.

BRK> .SS\$

BR	RP	PC	INST	MNEMONIC	
0	00	0000	074F0	NOP	: <u>SPC</u> Advance 1 step at space key input.
0	00	0001	1D000	MOV 00, #0	: <u>SPC</u> Advance 1 step at space key input.
0	00	0002	1D011	MOV 01, #1	: <u>SPC</u> Advance 1 step at space key input.
0	00	0003	1000A	ADD 00, #A	: <u>\$\$</u> Terminate step operation at \$\$ input.

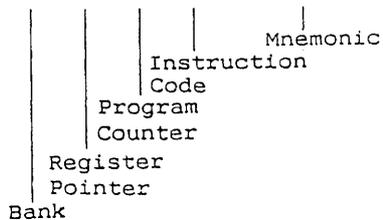
(Example 2) Perform 4 step operations.

BRK> 4.S\$

Specify step number.

BR	RP	PC	INST	MNEMONIC	
0	00	0000	074F0	NOP	:
0	00	0001	1D000	MOV 00, #0	:
0	00	0002	1D011	MOV 01, #1	:
0	00	0003	1000A	ADD 00, #A	:

] Execute 4 steps.



[Note] . Step operation is ended by \$\$ or ↵(return) input.

. When α is 0 or omitted, operation is performed 1 step at each instruction.

.DS Display

Format : .DS

[Function] Enables the liquid crystal display during break (this function only applies to the product with LCD controller).

Depending on the product, the liquid crystal display may go off during break. This command is used when desiring to view the liquid crystal display during break.

(Example) BRK> .DS\$\$
DSP>

- [Note] . Since a RUN state (BR instruction executed repeatedly) is provided as an emulation state, the trace and coverage contents are not guaranteed after this command is used.
- . Input of any key returns to the original state (break state).

5.8.5 BREAK/TRACE CONDITION CONTROL COMMANDS

- (1) Break/trace condition change
.CC (Change Break/Trace Condition)

- (2) Trace on/off condition change
.CT (Change Trace Condition)

- (3) Break/trace condition dump
.DC (Dump Break Condition)

- (4) Trace table dump
.DT (Dump Trace Table)

- (5) Break/trace condition save
.SC (Save Break/Trace Condition)

- (6) Break/trace condition load
.LC (Load Break/Trace Condition)

- (7) Break/Trace condition verify
.VC (Verify Break/Trace Condition)

.CC Change Break/Trace Condition

Format : .CC

[Function] Sets and changes the break and trace conditions.

[Explanation] Four independent break/trace conditions can be set simultaneously. In short, the condition setting unit is 4 units. When items are set in 4 units, .CC command level 1 is used. When each unit is set as a break condition, .CC command level 2 is used and when each unit is set as a trace condition, the .CT command is used.

The .CC command sets the conditions interactively.

The setting items are shown below.

↵ (return) is input when the predefined value of items C) to L) is input and \$ is input when exiting from the setting.

<Break condition setting items (for level 1)>

. A) LEVEL (1,2): ? 1

Selects the level. There are two levels: 1 and 2.

. B) UNIT (0 to 3): ?

Selects the unit. There are four units: 0 to 3. For the items which can be set at each unit, see Table 5-1.

CATG (C to L): ?

Specifies which setting item of C) to L) from which condition setting is to be performed. Depending on the unit, there may be no setting items. When no items are specified, setting becomes possible from the item after that item.

. C) CONDITION AND(1)/OR(0) : Predefined value ?

Selects if the setting items from D) to K) are ANDed or ORed. When the AND condition is selected, the unit break condition is established after all the setting items in the unit are satisfied. Therefore, when you want to remove a condition from the AND conditions, set the conditions so that that condition is always satisfied.

However, since the timing signal contributes to establishment of conditions E) and I), to remove from the AND conditions, set 1 at RELEASE-FROM-AND-.

. D) PROG ADDR UPPER: Predefined value ?

Specifies the upper limit of the program address break/trace conditions range.

PRO ADDR LOWER: Predefined value ?

Specifies the lower limit of the program address break/trace conditions range.

MATCH(1)/UNMATCH(0): Predefined value

?

When MATCH is specified, the above program address range becomes the break/trace condition.

When UNMATCH is specified, outside the above program address range becomes the break/trace condition.

- . E) RELEASE DATAMEMORY FROM AND YES(1)/
NO(0): Predefined value ?

When releasing item E) related to data memory from the AND condition of D) to K) (when 1 selected at (C), 1 is input. When the OR condition of D) to K) is selected (when 0 is selected at (C)), the contents of this setting are ignored and can be either 1 or 0.

The data memory condition becomes the AND of the three conditions DATA ADDR, CURRENT DATA, and PREVIOUS DATA (may not exist, depending on the unit).

DATA ADDR: Predefined value ?

Sets the break trace/trace condition using the data memory address in which data is written.

DATA ADDR MASK: Predefined value ?

Sets the mask data for the break/trace condition data memory address. The mask data is hexadecimal data with 1 set in the bit of the data memory address to be made the break/trace condition and 0 set in bits which may be either 1 or 0.

Since this item does not exist at unit 2, the data memory address break/trace condition cannot be masked.

MATCH(1)/UNMATCH(0): Predefined value
?

When MATCH is specified, the DATA ADDR value above becomes the break/trace conditions.

When UNMATCH is specified, a value other than the DATA ADDR value above becomes the break/trace condition.

CURRENT DATA: Predefined value ?

Sets the break/trace condition by written data memory value.

CURRENT MASK: Predefined value ?

Sets the mask data for the value of the break/trace condition data memory.

Since this item does not exist at unit 2, the data memory break/trace condition cannot be masked.

MATCH(1)/UNMATCH(0): Predefined value
?

When MATCH is specified, the CURRENT DATA value above becomes the break/trace condition.

When UNMATCH is specified, a value other than the CURRENT DATA value above becomes the break/trace condition.

PREVIOUS DATA DISABLE YES(1)/NO(0):

Predefined value ?

Since the DATA ADDR, CURRENT DATA, and PREVIOUS DATA break/trace conditions are AND conditions.

When you want to remove the PREVIOUS DATA break/trace condition from the item of E), input 1.

PREVIOUS DATA: Predefined value ?

Sets the break/trace condition by value of data memory to which data is previously written.

MATCH(1)/UNMATCH(0): Predefined value ?

When MATCH is specified, the PREVIOUS DATA value above becomes the break/trace condition.

When UNMATCH is specified, a value other than the PREVIOUS DATA value becomes the break/trace condition.

. F) SP LEVEL UPPER: Predefined value ?

Specifies the upper limit of the stack pointer break/trace condition range.

SP LEVEL LOWER: Predefined value ?

Specifies the lower limit of the stack pointer break/trace condition range.

MATCH(1)/UNMATCH(0): Predefined value

?

When MATCH is specified, within the stack pointer range above becomes the break/trace condition.

When UNMATCH is specified, outside the stack pointer above becomes the break/trace condition.

. G) INST CODE: Predefined value ?

Sets the break/trace condition by instruction code to be executed.

The instruction code description is 1-4-3-4-4 bit format.

INST MASK: Predefined value ?

Sets the break/trace condition by instruction code to be executed.

MATCH(1)/UNMATCH(0): Predefined value

?

When match is specified, the above instruction code becomes the break/trace condition.

When UNMATCH is specified, codes other than the above instruction code become the break/trace condition.

. H) PORT DATA: Predefined value ?

Sets the break/trace condition by input value from the logic analyzer probe connected to connector A.

The unit and logic analyzer probe pin correspondence is

Unit		logic analyzer probe pin
0	_____	ST0
1	_____	ST1
2	_____	ST2
3	_____	ST3

PORT MASK: Predefined value ?

Sets the mask data for the break/trace condition port data.

EDGE(1)/LEVEL(0): Predefined value ?

When 0 is set at the port data, when EDGE is specified, the falling edge is made the break/trace condition and when LEVEL is specified, pin low level becomes the break/trace condition.

When 1 is set at the port data, when EDGE is specified, the rising edge becomes the break/trace condition and when LEVEL is specified, pin high level becomes the break/trace condition.

MATCH(1)/UNMATCH(0): Predefined value ?

When MATCH is specified, the state of the port data above becomes the break/trace condition.

When UNMATCH is specified, other than the port data state above becomes the break/trace condition.

XREQ DATA: Predefined value ?

Sets the break/trace condition by input value from the XREQ pin of the logic analyzer probe connected to connector A.

The logic analyzer probe XREQ pin is a dedicated external break/trace signal input pin.

XREQ MASK: Predefined value ?

Sets the mask data for the break/trace condition XREQ data.

EDGE(1)/LEVEL(0): Predefined value ?

When 0 is set at the XREQ data, when EDGE is specified, the falling edge is made the break/trace condition and when LEVEL is specified, pin low level becomes the break/trace condition and when LEVEL is specified, pin low level becomes the break/trace condition.

When 1 is set at the XREQ data, EDGE is specified, the rising edge becomes the break/trace condition and when LEVEL is specified, pin high level becomes the break/trace condition.

MATCH(1)/UNMATCH(0): Predefined value
?

When MATCH is specified, the state of the XREQ data above becomes the break/trace condition.

When UNMATCH is specified, other than the state of the XREQ data above becomes the break/trace condition.

Setting at this item becomes the OR condition of the break/trace condition by port data setting and break/trace condition by XREQ setting.

. I) Currently, this item is not supported.

Therefore, always mask the break/trace condition of this item as follows:

```
I) RELEASE MAR FROM AND YES(1) / NO(0) : 0 ? 1
   MAR DATA           : 0 ? 0
   MAR MASK            : 0 ? 1

   MATCH(1) / UNMATCH(0): 0 ? 0
```

. J) INTERRUPT ACKNOWLEDGE: Predefined value ?

Sets the break/trace condition by interrupt generation.

When 1 is set at this item, the break/trace condition is established when an interrupt is generated during program execution.

The address that starts break and trace by interrupt generation is the corresponding vector address.

INTERRUPT MASK: Predefined value ?

Sets the mask data for the predefined value related to break/trace condition interrupt.

MATCH(1)/UNMATCH(0): Predefined value ?

When MATCH is specified, the set value for the interrupt set above becomes the break/trace condition.

When UNMATCH is specified, other than the set value for the interrupt set above becomes the break/trace condition value.

. K) DMA: Predefined value ?

Sets the break/trace condition by DMA (Direct Memory Access) generation.

When "DMA generated" is made the break/trace condition, 1 is set and when "DMA not generated" is made the break/trace condition, 0 is set.

Note that when DMA is generated, break is not generated.

DMA MASK: Predefined value ?

Sets the mask data for the break/trace condition DMA set value.

MATCH(1)/UNMATCH(0): Predefined value ?

When MATCH is specified, the set value for DMA above becomes the break/trace condition.

When UNMATCH is specified, other than the set value for DMA above becomes the break/trace value.

. L) COUNTER SOURCE SELECT

NO(0)/INST(1)/CONDITION(2)/INST AFTER
CONDITION(3): 0 ?

Sets the break/trace condition by counter overflow.

The counter is initialized to 0 and becomes an up counter that is incremented by one.

- . NO(0) ... Do not use counter.
- . INST(1) ... Count number of instruction executions unconditionally.
- . CONDITION(2) ... Count number of executions of instructions that satisfy break/trace condition as unit set at C) to K).
- . INST AFTER CONDITION(3) ... Count number of executions of executed instructions after conditions of items C) to K) satisfied.

TERMINAL COUNTER: Predefined value ?

Sets the counter final value.

COUNTER MASK: Predefined value ?

Sets the mask data value for the set value of the break/trace condition counter.

MATCH(1)/UNMATCH(0): Predefined value ?

When MATCH is specified, the counter value above becomes the break/trace condition.

When UNMATCH is specified, a value other than the counter value above becomes the break/trace condition.

(Output example for each unit)

<Unit 0>

```

BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER      : FFFF ?
   PROG ADDR LOWER    : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Program Memory
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ?
   DATA ADDR          : 000 ?
   DATA ADDR MASK     : 000 ?
   MATCH(1)/UNMATCH(0)-1 : 0 ?
   CURRENT DATA       : 0 ?
   CURRENT MASK        : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Data Memory
F) SP LEVEL UPER      : 0 ?
   SP LEVEL LOWER     : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Stack Pointer
H) PORT DATA         : 0 ?
   PORT MASK          : 0 ?
   EDGE(1) / LEVEL(0) : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Logic Analyzer Probe STO Pin
   XREQ DATA         : 0 ?
   XREQ MASK          : 0 ?
   EDGE(1) / LLEVEL(0) : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Logic Analyzer Probe XREQ Pin
J) INTERRUPT ACKNOWLEDGE : 0 ?
   INTERRUPT MASK      : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Interrupt
K) DMA                : 0 ?
   DMA MASK            : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] DMA
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
   TERMINAL CONTER      : 0000 ?
   COUNTER MASK         : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Counter

```

<Unit 1>

- BRK> CC\$\$
- A) LEVEL(1 , 2) : ? 1
 - B) UNIT (0 - 3) : ? 1
CATG (C - L) : ? C
 - C) CONDITION AND(1) / OR(0) : 0 ?
 - D) PROG ADDR UPER : FFFF ?
PROG ADDR LOWER : 0000 ?
MATCH(1) / UNMATCH(0) : 0 ?
] Program Memory
 - E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ?
DATA ADDR : 000 ?
DATA ADDR MASK : 000 ?
MATCH(1) / UNMATCH(0) : 0 ?
CURRENT DATA : 0 ?
CURRENT MASK : 0 ?
MATCH(1) / UNMATCH(0) : 0 ?
PREVIOUS DATA DISABLE YES(1) / NO(0) : 0 ?
PREVIOUS DATA : 0 ?
MATCH(1) / UNMATCH(0) : 0 ?
] Data Memory
 - H) PORT DATA : 0 ?
PORT MASK : 0 ?
EDGE(1) / LEVEL(0) : 0 ?
MATCH(1) / UNMATCH(0) : 0 ?
] Logic Analyzer Probe ST1 Pin
 - I) RELEASE MAR FROM AND YES(1) / NO(0) : 0 ?
MAR DATA : 0 ?
MAR MASK : 0 ?
MATCH(1) / UNMATCH(0) : 0 ?
] *
MAR
 - L) COUNTER SOURCE SELECT
NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
TERMINAL CONTER : 00 ?
COUNTER MASK : 00 ?
MATCH(1) / UNMATCH(0) : 0 ?
] Counter

* Not currently supported.

<Unit 2>

```
BRK> .CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 2
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER      : FFFF ?
   PROG ADDR LOWER    : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Program Memory
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : 0 ?
   DATA ADDR         : 000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   CURRENT DATA      : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Data Memory
H) PORT DATA         : 0 ?
   PORT MASK          : 0 ?
   EDGE(1) / LEVEL(0) : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Logic Analyzer Probe ST2 Pin
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
   TERMINAL CONTER      : 00 ?
   COUNTER MASK         : 00 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Counter
```

<Unit 3>

```
BRK> .CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 3
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) : 0 ?
D) PROG ADDR UPER      : FFFF ?
   PROG ADDR LOWER    : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Program Memory
G) INST CODE          : 00000 ?
   INST MASK          : 00000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Instruction Code
H) PORT DATA         : 0 ?
   PORT MASK          : 0 ?
   EDGE(1) / LEVEL(0) : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
   ] Logic Analyzer Probe ST3 Pin
```

<Break condition setting items (for level 2)>

Level 2 is used when setting each unit (units 0 to 3) set at level 1 as a break condition.

Setting is divided into four layers based on the DEPTH concept.

The OR condition of four units can be set in one DEPTH. The units specified 1 becomes the target unit of the OR condition. Units specified 0 are invalid.

When the OR condition in DEPTH is satisfied, the program waits for the next DEPTH condition and when the DEPTH-0 condition is satisfied, break is generated.

The condition order is DEPTH3 to DEPTH0. DEPTH that starts at INITIAL DEPTH setting can be specified.

(Example) BRK>.CC\$\$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2 : 0123
 DEPTH-3 : 0101 0000
 DEPTH-2 : 1101 ? 1111
 DEPTH-1 : 1101 ? 1010
 DEPTH-0 : 1101 ? 0001
 INITIAL DEPTH : 0 ? 1

Set so that a break is generated if the unit 3 condition set at level 1 is established after the unit 0 or unit 2 condition set at level 1 is established.

Table 5-1 Break/Trace Conditions Table

Item	UNIT0	UNIT1	UNIT2	UNIT3
C) CONDITION AND(1)/OR(0)	o	o	o	o
D) PROG ADDR UPER PROG ADDR LOWER MATCH(1)/UNMATCH(0)	o	o	o	o
E) RELEASE DATA MEMORY FROM AND YES(1)/NO(0) DATA ADDR	o	o	o	x
----- DATA ADDR MASK	o	o	x	x
----- MATCH(1)/UNMATCH(0) CURRENT DATA	o	o	o	x
----- CURRENT MASK	o	o	x	x
----- MATCH(1)/UNMATCH(0)	o	o	o	x
----- PREVIOUS DATA DISABLE YES(1)/NO(0) PREVIOUS DATA MATCH(1) UNMATCH(0)	x	o	x	x
F) SP LEVEL UPER SP LEVEL LOWER MATCH(1)/UNMATCH(0)	o	x	x	x
G) INST CODE INST MASK MATCH(1)/UNMATCH(0)	x	x	x	o
H) PORT DATA PORT MASK EDGE(1)/LEVEL(0) MATCH(1)/UNMATCH(0)	o (ST0)	o (ST1)	o (ST2)	o (ST3)
----- XREQ DATA XREQ MASK EDGE(1)/LEVEL(0) MATCH(1)/UNMATCH(0)	o	x	x	x
I) RELEASE MAR FROM AND YES(1)/NO(0) MAR DATA MAR MASK MATCH(1)/UNMATCH(0)	x	o*	x	x

(to be continued)

(cont'd)

Item	UNIT0	UNIT1	UNIT2	UNIT3
J) INTERRUPT ACKNOWLEDGE INTERRUPT MASK MATCH(1)/UNMATCH(0)	o	x	x	x
K) DMA DMA MASK MATCH(1)/UNMATCH(0)	o	x	x	x
L) COUNTER SOURCE SELECT NO(0)/INST(1)/CONDITION(2)/ INST AFTER CONDITION(3) TERMINAL COUNTER COUNTER MASK MATCH(1)/UNMATCH(0)	o	o	o	x

o ... Settable

x ... Not settable

*: Not currently supported

.CT Change Trace Condition

Format : .CT

[Function] Changes the trace on/off condition.

[Explanation]

Sets each unit set at .CC level 1 as the trace condition. Trace on/off condition setting is shown below.

```
BRK>.CT$$  
TRACE CONDITION MODE  
D : TRACE DON'T CARE ----①  
T : TRACE ON ----②  
U : TRACE OFF ----③  
S : TRACE ONE SHOT ----④  
LEVEL 1 UNIT : 0123  
              : DDDD ?  
              └───┘  
              Predefined Value
```

- ① Trace is not affected even if the unit trace condition is established.
- ② When the unit trace condition is satisfied, trace starts.
- ③ When the unit trace condition is satisfied, trace ends.
- ④ Only the place where the unit trace condition is established is traced.
(Trace one-shot)

(Example) BRK>.CT\$\$
TRACE CONDITION MODE
D : TRACE DON'T CARE
T : TRACE ON
U : TRACE OFF
S : TRACE ONE SHOT
LEVEL 1 UNIT : 0123
 : DDDD ? IUSS

When the unit 0 condition is established, trace starts.

When the unit 1 condition is established, trace ends.

When the unit 2 or 3 condition is established, trace starts.

- [Note] . When the trace condition is satisfied by multiple units at the same point, the trace condition priority is

TRACE ON>TRACE ONE SHOT>TRACE OFF

- . There are two kinds of trace, address trace and status trace.
Address trace is performed without regard to this command.
- . For execution after .R input and when execution is started by .BG, trace is turned on.
- . For execution after .R input and when execution is started by .BG, the contents set by .CT are not affected.

- . After TRACE ON and TRACE OFF, that state is held even if set to TRACE DON'T CARE.

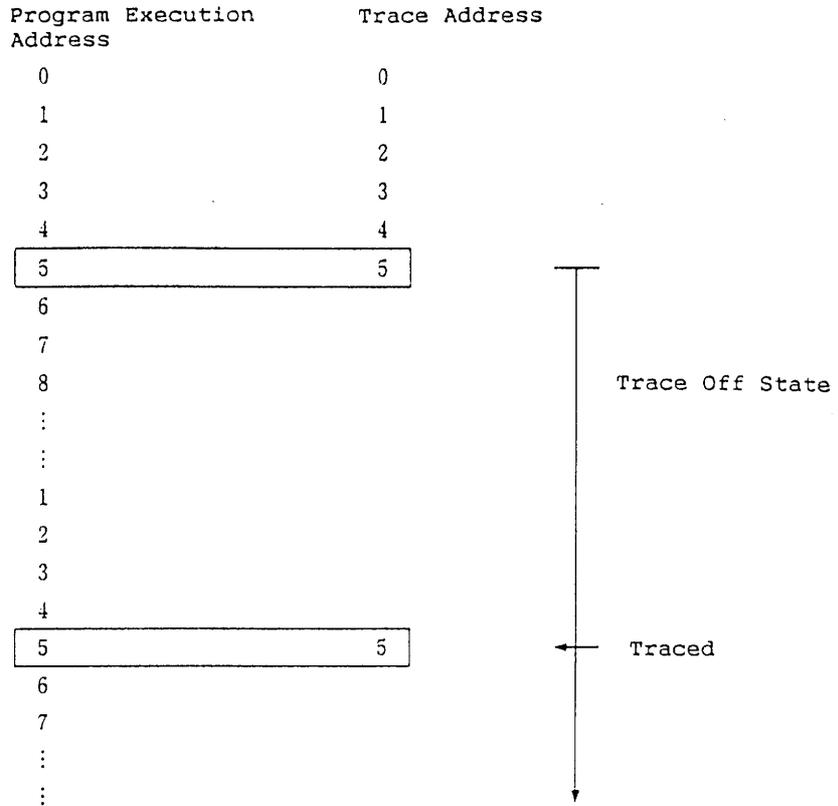
TRACE ONE SHOT is effective only in the trace off state.

When TRACE ONE SHOT is specified in the trace off state, only an address which satisfies the condition is traced.

After TRACE ONE SHOT is specified, even if TRACE DON'T CARE is specified, TRACE ONE SHOT specification is not held, but the program enters the trace off state before TRACE ONE SHOT is specified.

- . When TRACE OFF is specified, trace is not executed after trace off, but the execution address that decides the start of trace off is traced. In this case, the same operation as TRACE ONE SHOT specification is performed.

(Example 1) When trace off continues after trace off starts at address 5H.



(Example 2) When TRACE DON'T CARE is specifies at address 5H after trace off starts at address 5H.

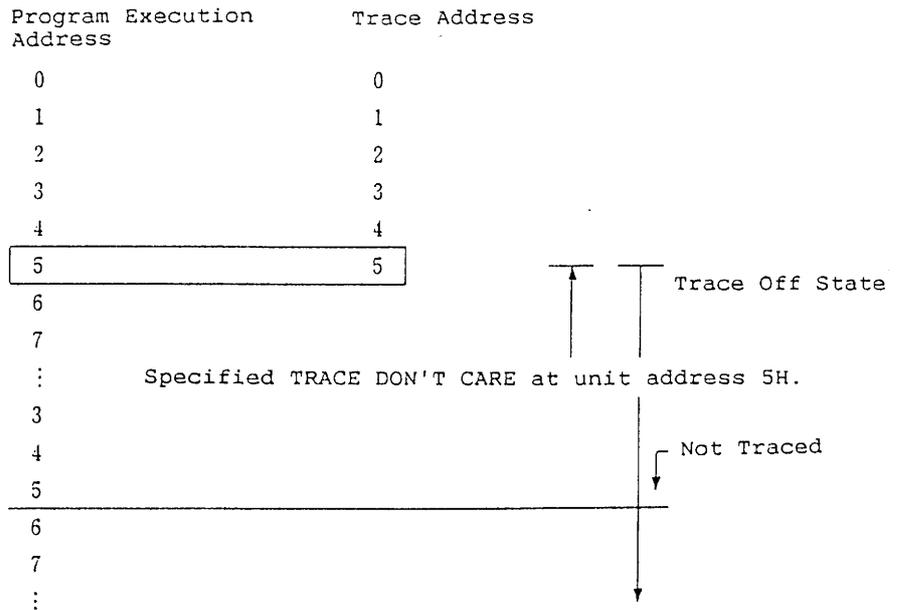


Table 5-2 Trace State Transition

Current Trace State Condition	Trace On	Trace Off	Trace One Shot
Trace on	Trace conditions	Trace starts	Trace starts
Trace off	Trace ends	Trace off continues	Trace one shot ends
Trace one shot	Trace continues (one short invalid)	Trace one shot starts	Trace one shot continues (newest condition valid)

.DC Dump Break Condition

Format : .DC

[Function] Dumps the break/trace conditions.

(Example) Dump the units 0 to 3 break/trace condition.

```
BRK>.DC$$  
UNIT (0 - 3) ? 0  
CONDITION : OR  
PROG ADDR : FFFF - 0000 UNMATCH  
DATA ADDR : 000 <000> UNMATCH  
CRNT : 0 < 0 > UNMATCH  
SP LEVEL : F - 0 UNMATCH  
PORT DATA : 0 <0> LEVEL UNMATCH XREQ : 0 <1> LEVEL UNMATCH  
INTERRUPT : 0 <0> UNMATCH  
DMA : 0 <0> UNMATCH  
COUNT SEL : NO 0000 <0000> UNMATCH  
TRACE SEL : TRACE ON
```

```
BRK>.DC$$  
UNIT (0 - 3) ? 1  
CONDITION : OR  
PROG ADDR : FFFF - 0000 UNMATCH  
DATA ADDR : 000 <000> UNMATCH  
CRNT : 0 < 0 > UNMATCH  
PRVS : 0 UNMATCH  
PORT DATA : 0 <0> LEVEL UNMATCH  
MAR DATA : 0 <0> UNMATCH  
COUNT SEL : NO 00 < 00> UNMATCH  
TRACE SEL : TRACE OFF
```

```
BRK>.DC$$  
UNIT (0 - 3) ? 2  
CONDITION : OR  
PROG ADDR : FFFF - 0000 UNMATCH  
DATA ADDR : 000 <000> UNMATCH  
CRNT : 0 < 0 > UNMATCH  
PORT DATA : 0 <0> LEVEL UNMATCH  
COUNT SEL : NO 00 < 00> UNMATCH  
TRACE SEL : TRACE DON'T CARE
```

```
BRK>.DC$$  
UNIT (0 - 3) ? 3  
CONDITION : OR  
PROG ADDR : FFFF - 0000 UNMATCH  
INST CODE : 0000 <0000> UNMATCH  
PORT DATA : 0 <0> LEVEL UNMATCH  
TRACE SEL : TRACE DON'T CARE
```

[Note] < >: Mask data

.DT Dump Trace Table

Format :[α , β].DT

α : Dump start trace number ($\alpha \leq \beta$; β : Dump end trace number $\alpha > \beta$: Error)

[Function] Dumps the trace contents from specified trace number α to β . When both α and β are omitted, the contents of the end of the trace table are displayed. System reset initializes the trace table and clears the trace counter to 0.

For trace within 32K (32768 decimal) steps, the trace counter shows the end of the trace table.

For trace exceeding 32K steps, the trace contents of the newest 32K steps are stored to the trace table and the trace counter becomes 7FFFH (32767).

There are two kinds of trace, address trace and status trace.

Address trace traces the newest program execution contents without regard to the trace conditions.

Status trace traces the range specified by the trace conditions.

Status trace includes much more information than address trace.

(Example 1) Dump the address trace results of trace numbers 0 to 10.

BRK>0, ^D10.DT\$\$

ADDRESS(1) / STATUS (0) TRACE? 1 ↵

TR_NO	ADDR	MNEMONIC	INST	
0000	0000	MOV 00 ,#A	1D00A	
0001	0001	MOV 01 ,#B	1D01B	
0002	0002	MOV 02 ,#C	1D02C	
0003	0003	MOV 03 ,#D	1D03D	
0004	0004	MOV 04 ,#E	1D04E	
0005	0005	MOV 05 ,#F	1D05F	
0006	0006	MOV 06 ,#0	1D060	
0007	0007	MOV 07 ,#1	1D071	
0008	0008	MOV 08 ,#2	1D082	
0009	0009	MOV 09 ,#3	1D093	
0010	000A	NOP	074F0	
(1)	(2)	(3)	(4)	(5)

(Example 2) Dump the status trace results from 0 to 10H.

BRK>0.10.DTSS

ADDRESS(1) / STATUS (0) TRACE? 0 ↵

TR_NO	ADDR	INSTRUCTION	PORT	WA	DB	JG	TIME
0000	0000	MOV 00 ,#A	1D00A	11111111	000	A	0 0000001
0001	0001	MOV 01 ,#B	1D01B	11111111	001	B	0 0000002
0002	0002	MOV 02 ,#C	1D02C	11111111	002	C	0 0000003
0003	0003	MOV 03 ,#D	1D03D	11111111	003	D	0 0000004
0004	0004	MOV 04 ,#E	1D04E	11111111	004	E	0 0000005
0005	0005	MOV 05 ,#F	1D05F	11111111	005	F	0 0000006
0006	0006	MOV 06 ,#0	1D060	11111111	006	0	0 0000007
0007	0007	MOV 07 ,#1	1D071	11111111	007	1	0 0000008
0008	0008	MOV 08 ,#2	1D082	11111111	008	2	0 0000009
0009	0009	MOV 09 ,#3	1D093	11111111	009	3	0 0000010
0010	000A	NOP	074F0	11111111	04F	0	0 0000011
0011	000B	NOP	074F0	11111111	04F	0	0 0000012
0012	000C	NOP	074F0	11111111	04F	0	0 0000013
0013	000D	NOP	074F0	11111111	04F	0	0 0000014
0014	000E	NOP	074F0	11111111	04F	0	0 0000015
0015	000F	BR 000F	0C00F	11111111	000	F	0 0000016
0016	0010	BR 000F	0C00F	11111111	000	F	0 0000017
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)(9) (10)

- (1) Trace number decimal display
- (2) Trace number hexadecimal display
- (3) Program address (program counter value)
- (4) Instruction mnemonic display
- (5) Instruction code (1-4-3-4-4 bit format)

- (6) State of each pin of logic analyzer probe

Corresponds to pin ST0, ST1, ,
ST7, from right to left.

- (7) Data memory write address

Effective when data is written to data
memory*.

- (8) Data bus

Shows the value written when data is
written to data memory*.

- (9) "*" (asterisk) is displayed for
instructions that are skipped at skip
instruction execution.

- (10) Time stamp

Set to 1 by .RN command and counted up by
one each time an instruction is executed
(However, when a MOVT instruction is
executed, it is counted up by 2).

*: This is valid only when written
in the data memory. The
contents displayed when writing
to the register file (PEEK
instruction), or when writing to
the peripheral register (PUT
instruction) are contents on the
the IE-17K bus and so the
displayed contents are invalid.

.SC0 .SC1 Save Break/Trace Condition

Format : { .SC0 .SC1 }

RS-232-C channel 0 : SC0 channel 1 : SC1

[Function] Outputs the break/trace conditions set at .CC level 1 to the RS-232-C channel specified by .SC0 or .SC1 in Intel hexadecimal format.

(Example) Output break/trace conditions to channel 0.

```
BRK>.SC0$$  
:104143000B0B0B0B00FF02020200000000001003A  
:1041530000010100000000FFFF0000100000FF0746  
:104163000B000F040000F00040000FFFF00C0011C  
:104173000001000F0000010100010100000000FF29  
:10418300FF000010000000FFFF0000100000FF0709  
:104193000B000F0400040F00000000FFFF000001EC  
:1041A3000001000F0400010000010000000000FF7  
:1041B300FF000008000000FFFF0000100000FF07E1  
:1041C3000B000F0400000F00000000FFFF000001C0  
:1041D3000001000F0000010000010000000000FFCB  
:1041E300FF000008000000FFFF0000100000FF07B1  
:1041F30000000F0000000F00000000FFFF1000018F  
:104203000001000F0000010000010000000000FF9A  
:07421300FF0000000000000A5  
:00000001FF
```

.LC0 .LC1 Load Break/Trace Condition

Format : { .LC0 } { .LC1 }

RS-232-C channel 0 : LC0 channel 1 : LC1

[Function] Stores the data sent to IE-17K in the area in which the IE-17K internal break/trace conditions are stored.

(Example) Input the break/trace conditions from channel 0.

BRK> .LCO\$\$

.VC0 .VC1 Verify Break/Trace Condition

Format : { .VC0 } { .VC1 }

RS-232-C channel 0 : VC0 channel 1 : VC1

[Function] Verifies the IE-17K internal break/trace conditions and the data sent to IE-17K.

If the conditions and data are the same, the message

Verify OK

and if the conditions and data are not the same, the message

Verify NG

is output.

(Example) Verify the break/trace condition input from channel 0.

BRK> .VCO\$\$
Verify OK

5.8.6 COVERAGE DISPLAY COMMAND

- (1) Coverage Memory Dump
 - .DM (Dump Coverage Memory)

.DM Dump Coverage Memory

Format : [α , β].DM

α : Start address ($\alpha \leq \beta$; β : End address $\alpha > \beta$: Error)

[Function] Dumps the contents of the coverage memory.

There are two coverage objectives, PC (Program Counter) and DATA.

- . PC coverage records the number of executions for the executed address by 0 to FFH. For values over FFH, FFH is displayed.
- . DATA coverage displays the data memory write state (bit units). The display is defined below.

<Definition of display>

- Bit not written even once
* Bit written 0 and 1
0 Bit written 0 only
1 Bit written 1 only

- . When PC coverage is selected, when α and β are omitted, program memory addresses 0 to 7FH are displayed.

- . When DATA coverage is selected, when α and β are omitted, data memory addresses 0 to 3FH are displayed. Register files are outside the coverage objective.

(Example 1) Display the contents of PC coverage.

```
BRK>.DMSS
PC (1) / DATA (0) COVERAGE : ? 1↵
ADDR 0 1 2 3 4 5 6 7 8 9 A B C D E F
0000 FF FF
0010 FF FF
0020 FF FF
0030 FF FF
0040 FF FF
0050 FF FF
0060 FF FF
0070 FF FF
```

(Example 2) Display the contents of DATA coverage.

```
BRK>.DMSS
PC (1) / DATA (0) COVERAGE : ? 0↵
ADDR 0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F
0000 ---- ---- ---- ---- ---- ---- ----
0008 ---- ---- ---- ---- ---- ---- ----
0010 ---- ---- ---- ---- ---- ---- ----
0018 ---- ---- ---- ---- ---- ---- ----
0020 ---- ---- ---- ---- ---- ---- ----
0028 ---- ---- ---- ---- ---- ---- ----
0030 ---- ---- ---- ---- ---- ---- ----
0038 ---- ---- ---- ---- ---- ---- ----
```

[Note] Depending on the device, when α and β are omitted, an error may be generated.

5.8.7 PROGRAM PATTERN GENERATOR (PPG) CONTROL COMMANDS

- (1) PPG data initialize
.IG (Initialize PPG Date)
- (2) PPG data change
.CG (Change PPG Data)
- (3) PPG data dump
.DG (Dump PPG Data)
- (4) PPG execute/stop/operate mode setting
.EG (Execute PPG)
- (5) PPG data save
.SG (Save PPG Data)
- (6) PPG data load
.LG (Load PPG Data)
- (7) PPG data verify
.VG (Verify PPG Data)

.IG Initialize PPG Data

Format : [α , β , γ].IG

α : Start address (0 to 1FFF) ($\alpha \leq \beta$;
β : End address (0 to 1FFF) $\alpha > \beta$: Error)
γ : Data (0 to FFFF)

[Function] Initializes the PPG of addresses α to β to γ .

When all the PPG data (0 to 1FFF) is cleared to 0, α , β and γ can be omitted.

(Example 1) Clear all the PPG data to 0.

BRK> .IG\$\$

(Example 2) Initialize the contents of PPG data addresses 0 to FFH to 5555.

BRK> 0,FF,5555.IG \$\$

[Note] .IG input is not accepted while PPG is executing.

.CG Change PPG Data

Format : [α].CG

α : PPG data to be changed (0 to 1FFF)

[Function] Changes the PPG data of address α . When α is 0, α can be omitted.

(Example 1) Change the PPG data from address 0.

```
BRK>.CG$$  
0000:0000000000000000 - 1111111111111111 ↵  
0001:0000000000000000 - $
```

(Example 2) Change the PPG data from address 100.

```
BRK>100.CG$$  
0100:1001101110110011 - 1001111111111111 ↵  
0101:1001101110110000 - ■
```

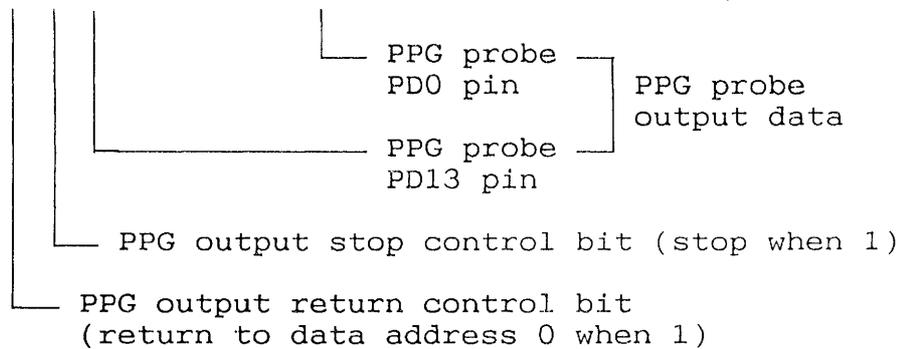
└─> Address
└─> Current Data
└─> Input Wait Cursor

[Note] . ↵ only input does not change the current data, but only shifts to the next address.

. \$ input ends change.

- . The change data which is input is 16 characters and .
- . The DEL key deletes the character input last. When there is no character to be deleted, this operation is not performed.
- . The PPG data items are meaningful.

← 16 bits →
0 0 0 0



- . When setting the PPG stop address, set the stop control bit to 1 for two consecutive steps.
- . When .CG is performed while PPG is executing, PPG stops.
- . The PPG data set at address 0H is output for a time of double the set step rate.
- . After PPG execution starts, provide an interval of at least 3us up to the next stop before setting the stop control bit.

.DG Dump PPG Data

Format : [α], β .DG
α : Start address (0 to 1FFF) ($\alpha \leq \beta$; β : End address (0 to 1FFF) $\alpha \leq \beta$: Error)

[Function] Display the PPT data of addressed α to β .

(Example) Dump the PPG data of addressed 0 to 10H.

```
BRK>0,10.DG$$  
0000 : 0000000000000000  
0001 : 0000000000000000  
0002 : 0000000000000000  
0003 : 0000000000000000  
0004 : 0000000000000000  
0005 : 0000000000000000  
0006 : 0000000000000000  
0007 : 0000000000000000  
0008 : 0000000000000000  
0009 : 0000000000000000  
000A : 0000000000000000  
000B : 0000000000000000  
000C : 0000000000000000  
000D : 0000000000000000  
000E : 0000000000000000  
000F : 0000000000000000  
0010 : 0000000000000000
```

[Note] When .IG is performed while PPG is executing, PPG stops.

.EG Execute PPG

Format : .EG

- [Function]
- ① Starts PPG.
 - ② Stops PPG.
 - ③ Sets the valid bits and the step rate as the PPG operation mode.

<Operation mode setting method>

- . When desiring to fix the output of PPG probe pins PD0 to PD13 at 0 (Low Level), set the SELECT BIT to 0. For output according to the data, set the SELECT BIT to 1.

The SELECT BIT setting and PPG probe pin correspondence is shown below.

SELECT BIT : 001-----1

PD0 Pin
PD13 Pin

- . STEP RATE sets the output time per PPG data step. It can be set from approximately 1 us to 13333 (decimal) per step. When 0 is selected, the step rate becomes 13333 us.

(Example) Fix the output of PPG pins 13 and 1 to 0 and set the execution speed per step to 10 ms.

```
BRK>.EGSS  
PPG  RUN (1) / RESET (2) / SELECT (3) : ? 3 ↵  
SELECT BIT : 0001111111111101↵  
STEP RATE : 10000↵
```

- [Note] . The PPG data set at address OH is output at a time of double the set step rate.
- . When RESET is performed during RUN, PPG stops and the valid bits set by SELECT BIT are fixed at high level.
 - . When RUN is performed again during RUN, RUN is continued.
 - . When SELECT is specified during RUN, the PPG stops.
 - . When the PPG is stopped by stop control bit, the valid PPG data of the stopped address is output.
 - . The exact expression for computing the execution speed per step is shown below.

$$\text{(Execution speed)} = \frac{1}{4.9152 \times 10^6} \times \text{INT} (4.9152 \times (\text{STEP RATE}))^*$$

* INT [] represents the maximum integer value that does not exceed [].

.LG0 .LG1 Load PPG Data

Format : { .LG0 } { .LG1 }

RS-232-C channel 0 : LG0 channel 1 : LG1

[Function] Inputs the PPG data from the RS-232-C channel specified by .LG0 or .LG1.

(Example) Input the PPG data from channel 0.

BRK> .LGO\$\$

.VG0 .VG1 Verify PPG Data

Format : $\left\{ \begin{array}{l} .VG0 \\ .VG1 \end{array} \right\}$

RS-232-C channel 0 : VG0 channel 1 : VG1

[Function] Verifies the PPG data and the Intel hexadecimal format data sent from the RS-232-C channel specified by .VG0 or .VG1.

If the data are the same, the message

Verify OK

is output and if the data are not the same, the message

Verify NG

is output.

(Example) Verify the PPG data input from channel 0.

BRK> .VG0\$\$

Verify OK

5.8.8 HELP COMMAND

- (1) Display of all commands
.H (Help)

.H Help

Format : .H

[Function] Display the commands table.

(Example) Display the commands table.

```
BRK>.H$$
.IP .CP .DP .FP .SPO .SPI .LPO .LPI .VPO .VPI .XSO XSI
    << PROGRAM MEMORY COMMAND >>
.ID .CD .DD .D
    << DATA MEMORY COMMAND >>
.R .RN .BG .BK .CA .S
    << EMULATION COMMAND >>
.CC .CT .DC .DT .DM .SCO .SC1 .LCO .LC1 .VCO .VC1
    << BREAK , TRACE CONDITION COMMAND >>
.IG .CG .DG .EG .SGO .SG1 .LGO .LG1 .VGO .VG1
    << PULSE GENERATER COMMAND >>
```

CHAPTER 6. PROGRAMMABLE PATTERN GENERATOR (PPG)

The IE-17K has a 14-bit parallel output PPG function. The output pattern can be set up to 8192 steps. The step rate can be set from approximately 1 us to 13333 us in approximately 1 us steps.

6.1 PPG DATA DISPLAY AND MODIFICATION

PPG data display and modification are performed by .DG and .CG command. The .DG and .CG commands stop the PPG.

(Example 1) Display the PPG data of addresses 0 to 3.

```
BRK>0,3.DG$$  
0000 : 0000000000000000  
0001 : 0000000000000000  
0002 : 0000000000000000  
0003 : 0000000000000000
```

(Example 2) Modify the address 0 PPG data to '0011111111111111'

```
BRK>.CG$$  
0000:0000000000000000: - 0011111111111111 ↵  
0001:0000000000000000: - $  
BRK>
```

6.2 PPG PROBE VALID BITS AND STEP RATE SETTING

Valid bits setting and step rate modification can be performed by selecting SELECT(3) by .EG command. When .EG SELECT(3) is selected, the PPG stops.

Of the 14 bits output pins, the output of which pins is valid can be set for each bit.

When the PPG stopped, high level is output and when the PPG is operating, the PPG data is output at the output pins set at the valid bits.

Low level is output at output pins without valid bit without regard to the PPG data.

For the step rate, the time per step can be set from approximately 1 us to 13333 us in approximately 1 us units.

One step is approximately 1 us, but when set step rate is small, one step becomes smaller than 1 us.

(Example) Enable output of bits 0 to 7 and set the step rate to 100 us/step.

```
BRK> .EG$$  
PSG RUN (1) / RESET(2) / SELECT(3) : ? 3 ↵  
SELECT BIT : 0000000011111111 ↵  
STEP RATE : 100 ↵  
BRK>
```

6.3 PPG STARTING AND STOPPING

.EG is used to start and stop the PPG.

PPG execution can be started only when the PPG is stopped.

(Example 1) Start PPG execution.

```
BRK> .EG$$  
PSG RUN (1) / RESET (2) / SELECT (3) : ? 1 ↵  
BRK>
```

(Example 2) Stop the PPG.

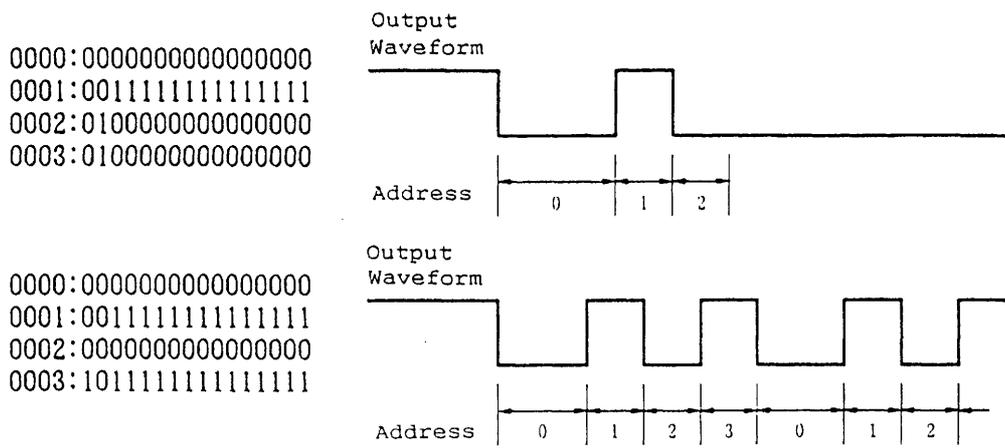
```
BRK> .EG$$  
PSG RUN (1) / RESET(2) / SELECT(3) : ? 2 ↵  
BRK>
```

6.4 PPG USAGE PRECAUTIONS

The data set at PPG address 0000H is output at a time of twice the set step rate.

When using the PPG stop control bit to stop the PPG, set the stop control bit to 1 for two consecutive steps.

(Example)

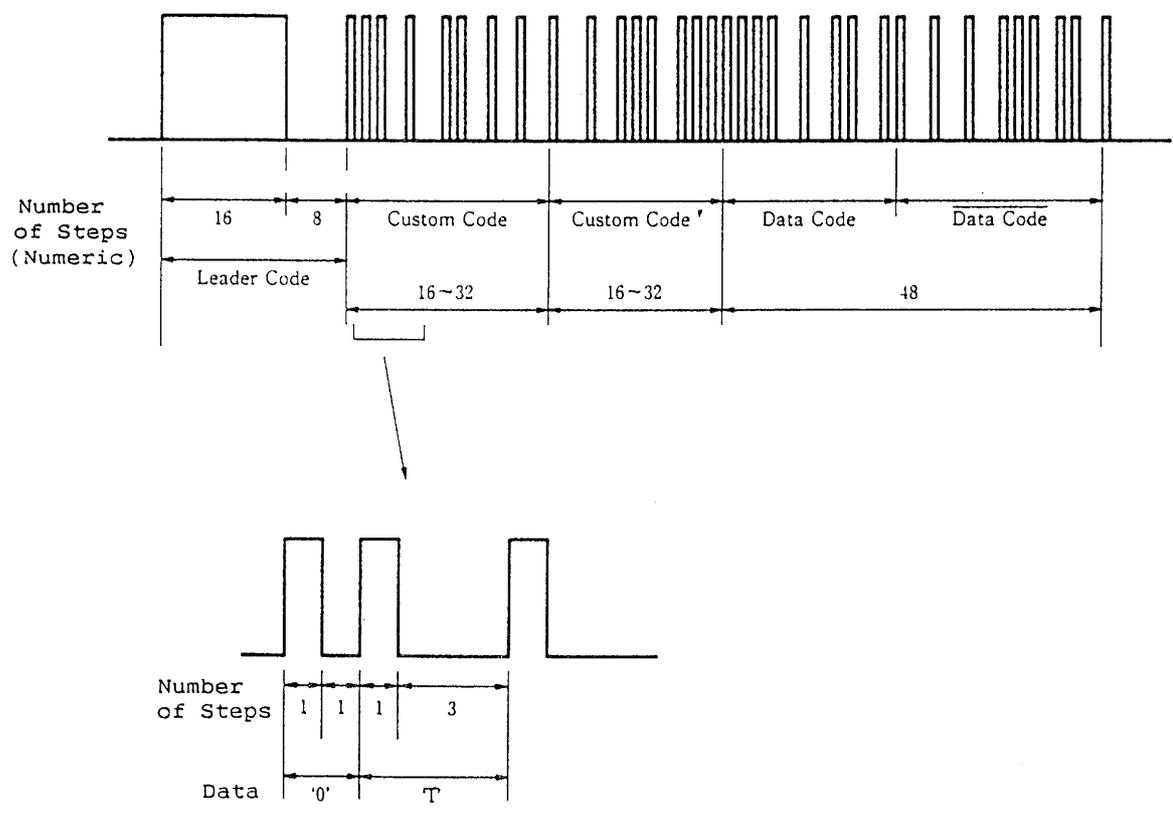


6.5 PPG EXAMPLES

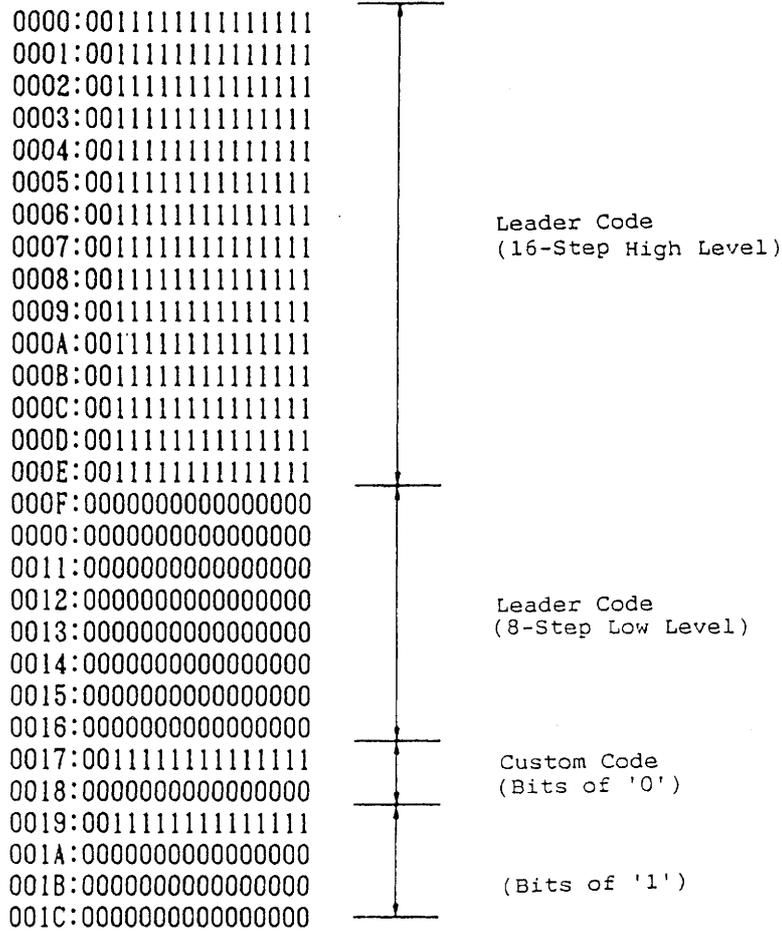
Remote transmission IC uPD6122G output data can be simulated by using the PPG. When the step rate is set to 563, one transmit data can be generated by using steps 106 to 138.

A uPD6122G output waveform example and part of the pattern generator data are shown below.

(Output waveform example)



PPG data (partial)



⌋

Phase-out/Discontinued

CHAPTER 7. PROGRAM EXECUTION

The following program execution methods are available.

- (1) Real-time emulation
- (2) 1 step emulation

7.1 REAL-TIME EMULATION

When desiring to run the program at the same speed as the actual product, use `.RN`. Break at an arbitrary condition is possible by setting the break point.

Execution can also be aborted by `.BK`.

(Example 1) Execute after resetting the CPU.

```
BRK>.R$$  
BRK>.RN$$  
RUN>
```

(Example 2) Resume after real-time emulation is broken.

```
RUN>.BK$$  
ADDR INSTRUCTION  
0027 1E7F2 BREAK  
0028 0C026 OVERRUN  
0029 070E0 NEXT  
PC SP AR WR BR MP IX  
0029 0 9999 * * *** **  
PSW :DB CP CY Z IXE MPE JG  
0 1 0 1 * * 0  
RP 0123456789ABCDEF  
*0 8D98D99999FFAD9D  
  
BRK>.RN$$  
  
RUN>
```

7.2 BREAK POINT SETTING

Execution can be broken by an arbitrary condition by setting the break point. At the break condition, program memory address, data memory address, data write to data memory, logic analyzer probe input level change, etc. can be set. (See Figure 7-1.)

The break condition can not only be used alone, but can also be set so that execution is broken when multiple break conditions are established simultaneously or multiple break conditions are established continuously. (See Figure 7-2.)

Figure 7-1 Break Condition Setting

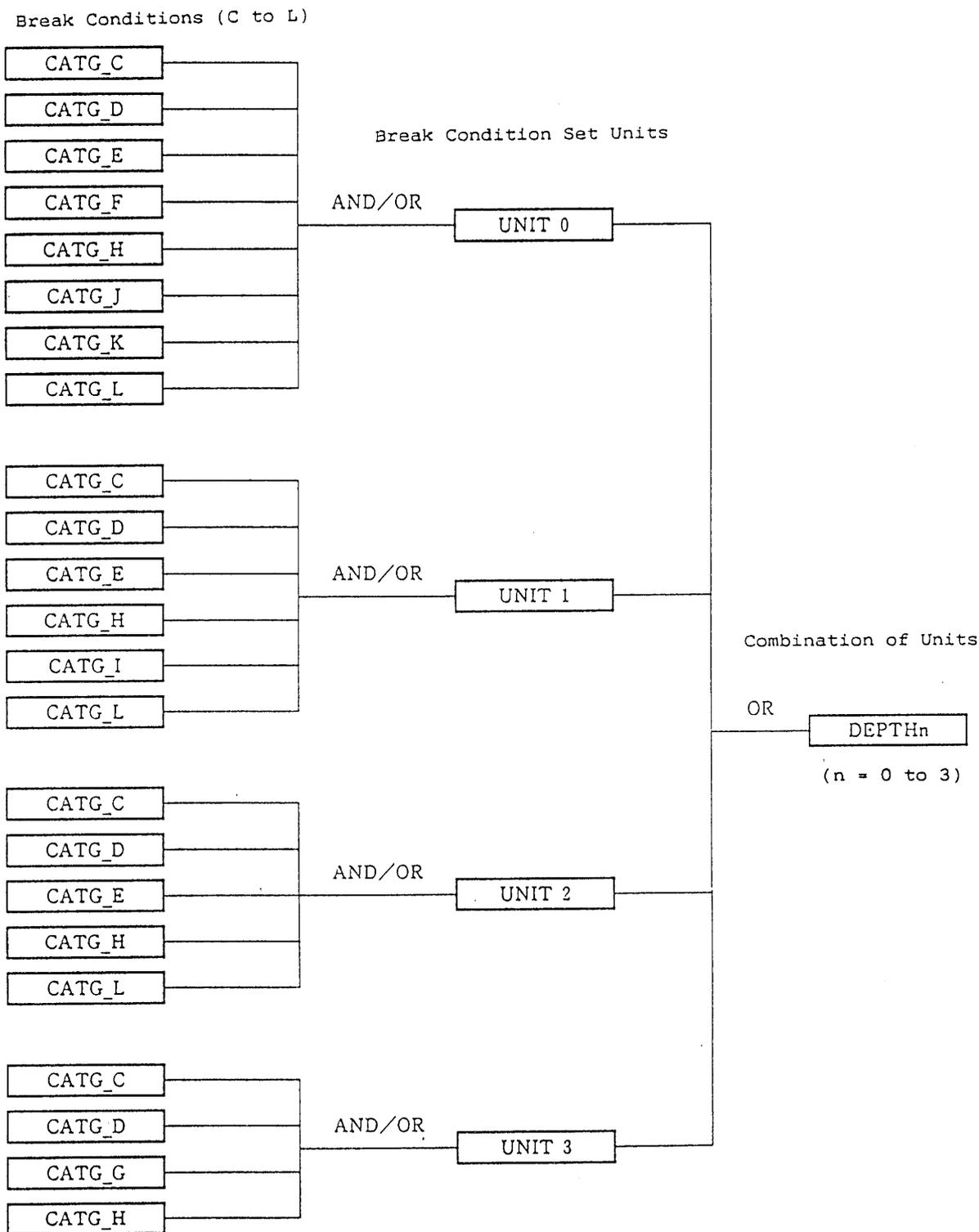
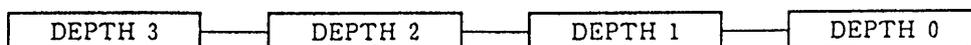


Figure 7-2 Break by Break Condition Sequence



7.2.1 BREAK BY PROGRAM ADDRESS

Break by program address is performed by specifying the address by .CC command.

(Example 1) Break when the program address reaches OOF0H.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) 0 : ? 0
D) PROG ADDR UPER      : FFFF ? OOF0
   PROG ADDR LOWER     : 0000 ? OOF0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 1000
INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
OOF0 1D069 BREAK
OOF1 1D059 OVERRUN
OOF2 1D045 NEXT
PC SP AR WR BR MP IX
OOF2 0 0000 * * *** ***
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
BRK>
```

(Example 2) Break when the program address enters the address 00F0H to 00FFH range.

```
BRK>.CC$$  
A) LEVEL(1 , 2) : ? 1  
B) UNIT (0 - 3) : ? 0  
   CATG (C - L) : ? C  
C) CONDITION AND(1) / OR(0) 0 : ? 0  
D) PROG ADDR UPER      : FFFF ? 00FF  
   PROG ADDR LOWER    : 0000 ? 00F0  
   MATCH(1) / UNMATCH(0) 0 ? 1  
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$  
A) LEVEL(1 , 2) : ? 2  
B) LEVEL2      : 0123  
   DEPTH-3     : 1101 ? 0000  
   DEPTH-2     : 1101 ? 0000  
   DEPTH-1     : 1101 ? 0000  
   DEPTH-0     : 1101 ? 1000  
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$  
BRK>.RN$$  
ADDR INSTRUCTION  
00F0 1D069 BREAK  
00F1 1D059 OVERRUN  
00F2 1D045 NEXT  
PC SP AR WR BR MP IX  
00F2 0 0000 * * *** ***  
PSW :DB CP CY Z IXE MPE JG  
      0 0 0 0 * * 0  
RP 0123456789ABCDEF  
*0 0000099990008005  
BRK>
```

(Example 3) Break when the program address exceeds the address 0000H to 00EFH range.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) 0 : ? 0
D) PROG ADDR UPER      : FFFF ? 00EF
   PROG ADDR LOWER    : 0000 ? 0000
   MATCH(1) / UNMATCH(0) 0 ? 0
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 1000
INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
00F0 1D069 BREAK
00F1 1D059 OVERRUN
00F2 1D045 NEXT
PC SP AR WR BR MP IX
00F2 0 0000 * * *** ***
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
BRK>
```

(Example 4) Break when program address 00F0H is executed five times.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0
   CATG (C - L) : ? C
C) CONDITION AND(1) 7 OR(0) 0 : ? 0
D) PROG ADDR UPER      : FFFF ? 00F0
   PROG ADDR LOWER     : 0000 ? 00F0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0
   CATG (C - L) : ? L
L) COUNTER SOURCE SELECT
   NO(0)/INST(1)/CONDITION(2)/INST AFTER CONDITION(3) : 0 ? 2
   TERMINAL COUNTER      : 00 ? 5
   COUNTER MASK          : 00 ? FF
   MATCH(1)/UNMATCH(0)  : 0 ? 1
                                     Break when condition
                                     above establishes 5
                                     times
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 1000
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
00F0 1D069 BREAK
00F1 1D059 OVERRUN
00F2 1D045 NEXT
PC SP AR WR BR MP IX
00F2 0 0000 * * *** ***
PSW :DB CP CY Z IXE MPE JG
     0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
BRK>
```

7.2.2 BREAK BY DATA MEMORY MODIFICATION

Break by data memory modification can be performed by specifying the data memory address and modification data by .CC command.

(Example 1) Break when data memory address 1.30H is modified.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1
   CATG (C - L) : ? E
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? 0
   DATA ADDR      : 000 ? 130      Set data memory address to 1.30H
   DATA ADDR MASK : 000 ? F7F    Enable all bits of address
                                       specification
   MATCH(1)/UNMATCH(0) : 0 ? 1    Break at an address above
   CURRENT DATA     : 0 ? 0
   CURRENT MASK      : 0 ? 0      Unrelated to write data
   MATCH(1)/UNMATCH(0) : 0 ? 1
   PREVIOUS DATA DISABLE YES(1)/NO(0) : 0 ? 1
   PREVIOUS DATA    : 0 ? 0      Unrelated to data before
   MATCH(1)/UNMATCH(0) : 0 ? 1    modification
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 0100
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
OOF0 1D309 BREAK
OOF1 1D059 OVERRUN
OOF2 1D045 NEXT
PC SP AR WR BR MP IX
OOF2 0 0000 0 1 000 000
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 0 0 0
RP 0123456789ABCDEF
OO 0000099990008005
BRK>
```

(Example 2) Break when 5H is written to data memory address 1.30H.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1
   CATG (C - L) : ? E
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? 0
   DATA ADDR      : 000 ? 130      Set data memory address to 1.30H
   DATA ADDR MASK : 000 ? F7F      Enable all bits of address
                                       specification
   MATCH(1)/UNMATCH(0) : 0 ? 1
   CURRENT DATA    : 0 ? 5         Break when written value is 5H
   CURRENT MASK     : 0 ? F
   MATCH(1)/UNMATCH(0) : 0 ? 1
   PREVIOUS DATA DISABLE YES(1)/NO(0) : 0 ? 1
   PREVIOUS DATA   : 0 ? 0         Unrelated to data before
                                       modification
   MATCH(1)/UNMATCH(0) : 0 ? 1
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 0100
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
OOF0 1D305 BREAK
OOF1 1D059 OVERRUN
OOF2 1D045 NEXT
PC SP AR WR BR MP IX
OOF2 0 0000 0 1 000 000
PSW :DB CP CY Z IXE MPE JG
     0 0 0 0 0 0 0
RP 0123456789ABCDEF
OO 0000099990008005
BRK>
```

(Example 3) Break when bit 0 of data memory address 1.3xH is set.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1
   CATG (C - L) : ? E
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? 0
   DATA ADDR      : 000 ? 130      Set data memory address to 1.30H
   DATA ADDR MASK : 000 ? F70      Column address can be anything
   MATCH(1)/UNMATCH(0) : 0 ? 1
   CURRENT DATA   : 0 ? 1          Break when bit 0 is set
   CURRENT MASK    : 0 ? 1
   MATCH(1)/UNMATCH(0) : 0 ? 1
   PREVIOUS DATA DISABLE YES(1)/NO(0) : 0 ? 1
   PREVIOUS DATA  : 0 ? 0          Unrelated to data before
   MATCH(1)/UNMATCH(0) : 0 ? 1      modification
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 0100
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
OOF0 1D309 BREAK
OOF1 1D059 OVERRUN
OOF2 1D045 NEXT
PC SP AR WR BR MP IX
OOF2 0 0000 0 1 000 000
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 0 0 0
RP 0123456789ABCDEF
OO 0000099990008005
BRK>
```

(Example 4) Break when data memory address 1.30H is changed from 1 to 5.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1
   CATG (C - L) : ? E
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? 0
   DATA ADDR      : 000 ? 130      Set data memory address to 1.30H
   DATA ADDR MASK : 000 ? F7F      Enable all bits of address
   MATCH(1)/UNMATCH(0) : 0 ? 1      specification
   CURRENT DATA   : 0 ? 5         Break when write value is 5H
   CURRENT MASK    : 0 ? F
   MATCH(1)/UNMATCH(0) : 0 ? 1
   PREVIOUS DATA DISABLE YES(I)/NO(O) : 0 ? 0
   PREVIOUS DATA  : 0 ? 1         Data before change is 1
   MATCH(1)/UNMATCH(0) : 0 ? 1
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 0000
   DEPTH-0     : 1101 ? 0100
   INITIAL DEPTH : 0 ? 0
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
OOF0 1D305 BREAK
OOF1 1D059 OVERRUN
OOF2 1D045 NEXT
PC SP AR WR BR MP IX
OOF2 0 0000 0 1 000 000
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 0 0 0
RP 0123456789ABCDEF
OO 0000099990008005
BRK>
```

7.2.3 BREAK BY MULTIPLE BREAK CONDITION

(Example 1) Break when program address reaches address 00F0H or 01F0H.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0          Set 00F0H at unit 0
   CATG (C - L) : ? C
C) CONDITION AND(1) 7 OR(0) 0 : ? 0
D) PROG ADDR UPER   : FFFF ? 00F0
   PROG ADDR LOWER  : 0000 ? 00F0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1          Set 01F0H at unit 1
   CATG (C - L) : ? C
C) CONDITION AND(1) 7 OR(0) 0 : ? 0
D) PROG ADDR UPER   : FFFF ? 01F0
   PROG ADDR LOWER  : 0000 ? 01F0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2       : 0123
   DEPTH-3      : 1101 ? 0000
   DEPTH-2      : 1101 ? 0000
   DEPTH-1      : 1101 ? 0000
   DEPTH-0      : 1101 ? 1100
   INITIAL DEPTH : 0 ? 0          Break when unit 0 or unit
                                   1 condition establishes
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
00F0 1D069  BREAK
00F1 1D059  OVERRUN
00F2 1D045  NEXT
PC SP AR WR BR MP IX
00F2 0 0000 * * *** ***
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
BRK>
```

(Example 2) Break when program address 01F0H is executed after address 00F0H.

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 0           Set 00F0H at unit 0
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) 0 : ? 0
D) PROG ADDR UPER      : FFFF ? 00F0
   PROG ADDR LOWER    : 0000 ? 00F0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 1
B) UNIT (0 - 3) : ? 1           Set 01F0H at unit 1
   CATG (C - L) : ? C
C) CONDITION AND(1) / OR(0) 0 : ? 0
D) PROG ADDR UPER      : FFFF ? 01F0
   PROG ADDR LOWER    : 0000 ? 01F0
   MATCH(1) / UNMATCH(0) 0 ? 1
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? $
```

```
BRK>.CC$$
A) LEVEL(1 , 2) : ? 2
B) LEVEL2      : 0123
   DEPTH-3     : 1101 ? 0000
   DEPTH-2     : 1101 ? 0000
   DEPTH-1     : 1101 ? 1000   To DEPTH0 when unit 0 establishes
   DEPTH-0     : 1101 ? 0100   Break when unit 1 establishes
   INITIAL DEPTH : 0 ? 1
```

```
BRK>.R$$
BRK>.RN$$
ADDR INSTRUCTION
01F0 1D069 BREAK
01F1 1D059 OVERRUN
01F2 1D045 NEXT
PC SP AR WR BR MP IX
01F2 0 0000 * * *** ***
PSW :DB CP CY Z IXE MPE JG
      0 0 0 0 * * 0
RP 0123456789ABCDEF
*0 0000099990008005
BRK>
```

7.3 1-STEP EMULATION

One-step emulation is used when desiring to verify the processing flow by executing the program one instruction at a time.

(Example 1) Execute one program execution.

```
BRK>.S$$  
BR RP PC INSTRUCTION  
* *0 0034 0C03D  $$
```

BRK>

The instruction of displayed address 0034H is not executed.

When a numeric is described in front of the .S command, the number of steps can be specified.

(Example 2) Execute the two instructions of address 33H and address 34H.

```
BRK>33.CA$2.S$$  
BR RP PC INSTRUCTION  
* *0 0034 0C03D  
* *0 003D 11001  $$
```

BRK>

After the .S command is executed, the next instruction is executed by entering a space.

(Example 3) Execute the two instructions of address 33H and address 34H.

BRK>33.CA\$.S\$\$

BR RP PC INSTRUCTION

* *0 0034 0C03D

* *0 003D 11001 \$

.....When a space is input, the program is executed 1 step

BRK>

Phase-out/Discontinued

CHAPTER 8. SE BOARD PROM CREATION

Programs corrected by IE-17K can output the PROM file format hexadecimal codes output by AS17K to channel 0 or channel 1 by using .XS0 or .XS1. Connecting PROM programmer to channel 1 makes it possible to create SE board PROM.

Phase-out/Discontinued

CHAPTER 9. ERROR MESSAGES

The IE-17K generates error messages for command errors and for IE-17K hardware errors.

9.1 COMMAND ERRORS

Command errors are error messages which are displayed when there is an error in the command name and when the number of arguments do not match at command input.

The error messages and their meanings are shown below.

(1) ?ANF ARY NOT FOUND

Array not found.

Specify the correct array.

(2) ?ARG IMPROPER ARGUMENT

Incorrect argument.

Input the correct argument.

(3) ?ASC ARGUMENT STACK COUNTER ERROR

Number of arguments is incorrect.

Adjust the commands so that the number of arguments is correct.

(4) ?AST ARGUMENT STACK OVERFLOW

Number of arguments is too large.

Reduce the number of arguments.

(5) ?AUN ARGUMENT STACK UNDERFLOW

No value in argument stack.

Assign a value to the argument stack.

(6) ?FAP FAIL TO ACCESS PPG

When programmable pattern generator data is saved, a verify error is generated.

Save again.

(7) ?ILL ILLEGAL COMMAND

An unsupported command is used or a command is used incorrectly.

Use the correct command.

(8) ?INA ILLEGAL NUMBER OF ARGUMENTS

Number of macro command arguments is small.

Correct input the macro command arguments.

(9) ?IPE INPUT ERROR

Illegal value is set by .CC command.

Reset the correct value.

(16) ?MRA MISSING>

'>' in loop is less than the number of '<'.

Correct the < > correspondence.

(17) ?MRP MISSING(

')' is less than the number of '('.

Correct the () correspondence.

(18) ?MVQ NO VALUE IN Q-REGISTER

A Q-register that is not macro-defined is executed as a macro.

Specify the correct macro-defined Q-register name.

(19) ?NAE NO ARGUMENT BEFORE=

No = command argument.

Input the argument.

(20) ?NAQ NO ARGUMENT BEFORE"

No " command argument.

Input the argument.

(21) ?NVQ NO VALUE IN Q-REGISTER

There is no value (numeric or character string) in the Q-register.

Assign a value to the Q-register.

(22) ?POS INVALID ADDRESS

Address specification exceeds the program memory address range of the target product.

Specify a value within the last program memory address.

(23) ?QNF QUOTATION NOT FOUND

' not found.

Input the '.

(24) ?QRO Q-REGISTER AREA OVERFLOW

Q-register area is full.

Delete the contents of the unwanted Q-register.

(25) ?QST Q-STACK OVERFLOW

Q-stack area is full.

Delete the unwanted contents of the Q-register.

(26) ?QUN Q-STACK UNDERFLOW

There is no value at the Q-stack area.

Assign at value to the Q-stack area.

(27) ?RNE CPU RUN ERROR

An unexecutable command is input during program execution.

Stop the program by .BK command and re-input the command.

(28) ?RNG NUMERIC RANGE OVER

Numeric exceeds the effective range.

Set the numeric to within the effective range.

(29) ?RSE CPU RESET ERROR

An unexecutable command is input during program execution.

Stop the program by .BK command and re-input the command.

(30) ?RTE CPU RESET ERROR

An unexecutable command is input during program execution.

Stop the program by .BK command and re-input the command.

(31) ?SYN INVALID SYNTAX

Syntax is incorrect.

Use the correct syntax.

(32) ?TAG MISSING TAG !XXX!

Tag is not found.

Specify the correct tag.

(33) ?UPE UNPRINTABLE ERROR

An error that is not available with the CLICE system is generated. There is an error in the system. Please restart it.

(34) ?UPO UNTERMINATED POINTER @ OR _

@ command or _ command not terminated by V.

Terminate with V.

(35) ?UTG UNTERMINATED TAG

Tag not enclosed in !.

Enclose tag in !.

(36) ?WRE WRITE ERROR

When writing to memory, a verify error is generated.

Execute the command again.

9.2 HARDWARE ERRORS

Hardware errors are error messages that are displayed when the IE-17K malfunctions during program execution.

The messages and their meanings are shown below.

(1) SYSTEM REGISTER ACCESS ERROR

Displayed when an uninstalled bit is set at a system register other than the AR register.

(2) STACK OVER/UNDER FLOW

Displayed when the stack pointer underflows or overflows.

(3) RAM NOT INITIALIZE

Displayed when an instruction which reads a data memory, other than a port, etc., which is not written even once or data memory without an initial value is executed.

(4) ILLEGAL RAM WRITE

Displayed when a nonexistent data memory is written.

(5) ?IOS INVALID OPTION SWITCH AT 0000

Displayed when the option switch specification described when a program is loaded to the IE-17K or at program execution is different from the setting of the option switch on the SE board. The last number is the number of the SE board switch with the different setting.

(6) ?ISE INVALID SE BOARD NUMBER [00-00]

Displayed when the device file used by the assembler when a program is loaded to the IE-17K or at program execution and the SE board are different. It is also displayed when the SE board installation state is bad. The left side of the last number is the SE board number and the right side is the number included at the device file.

(7) ?IDI INVALID DEVICE ID NUMBER [00-00]

Displayed when the device file used by the assembler when a program is loaded to the IE-17K or at program execution and the device on the SE board are different. It is also displayed when the SE board installation state is bad. The left side of the last number is the device number on the SE board and the right side is the number included at the device file.

(8) ----- NO SWITCH OPTION -----

Displayed when the option information is not loaded normally when a program is loaded to the IE-17K.

(9) PC ERROR!

Displayed when the program counter does not operate as expected due to a malfunction of the device on the SE board.

Error message shown from (10) to (18) below are displayed when there is an error in the result of IE-17K self-diagnosis when the power is turned on and at system reset. When these error messages are displayed, the hardware is fault and must be repaired.

- (10) MEMORY ERROR → 0000:0000 to 7000:FFFF

Displayed when there is an error in the memory used by the IE-17K.

- (11) MEMORY ERROR → E000:0000 to E000:FFFF

Displayed when there is an error used at the pulse generator.

- (12) DEVICE ERROR → PTC (UPD71054) #0

Displayed when there is an error at programmable timer 0 (uPD71054).

- (13) DEVICE ERROR → PTC (UPD71054) #1

Displayed when there is an error at programmable timer 1 (uPD71054).

- (14) DEVICE ERROR → PIU (UPD71055) #0

Displayed when there is an error at parallel interface 0 (uPD71055).

- (15) DEVICE ERROR → PIU (UPD71055) #1

Displayed when there is an error at parallel interface 1 (uPD71055).

- (16) DEVICE ERROR → SCU (UPD71051) #0

Displayed when there is an error at serial control unit 0 (uPD71051).

(17) DEVICE ERROR → SCU (UPD71051) #1

Displayed when there is an error at serial control unit 1 (uPD71051).

(18) DEVICE ERROR → ICU (UPD71059)

Displayed when there is an error at the interrupt controller (uPD71059).

Error messages (19) to (24) below are displayed when IE-17K CPU runs away or otherwise malfunctions.

(19) <<DIVIDE BY ZERO>>

Divided by '0'.

(20) <<CHECK FIELD>>

Memory boundary crossed.

(21) <<SINGLE STEP>>

Single step is performed.

(22) <<BREAK MODE>>

Break instruction is executed.

(23) <<OVERFLOW>>

Overflow is generated during operation.

(24) <<NMI>>

NMI is generated.

Phase-out/Discontinued

APPENDIX A. PRIMITIVE COMMANDS

The primitive command is used when creating a user original macro command. However, for IE-17K CLICE 1.6, supports 52 built-in macro commands that are combined with this primitive command. The primitive command is not necessary for normal work. This chapter should be read if the advanced built-in commands are to be used.

A.1 PRIMITIVE COMMANDS TABLE

Table A-1 Commands Table

Command	ASCII Code	Function	Command	ASCII Code	Function
↑ @	00	Dummy	↑ V	16	Pointer
↑ A	01	Character string display	↑ W	17	<Undefined>
↑ B	02	Binary constant prefix	↑ X	18	1 line deletion
↑ C	03	Abort	↑ Y	19	<Undefined>
↑ D	04	Decimal constant prefix	↑ Z	1A	<Undefined>
↑ E	05	Editor start	↑ [1B	Dummy
↑ F	06	Array address function	↑ ¥	1C	<Undefined>
↑ G	07	Current line display	↑]	1D	<Undefined>
↑ H	08	1 character deletion	↑ ^	1E	Remainder operator
↑ I	09	Dummy	↑ _	1F	<Undefined>
↑ J	0A	Dummy	SPACE	20	Dummy
↑ K	0B	<Undefined>	!	21	Tag
↑ L	0C	<Undefined>	"	22	Conditional branch
↑ M	0D	Dummy	#	23	OR operator
↑ N	0E	<Undefined>	\$	24	Dummy
↑ O	0F	<Undefined>	%	25	Q-register increment
↑ P	10	14344 constant prefix	&	26	AND operator
↑ Q	11	User-defined function call	'	27	Conditional branch
↑ R	12	Control character input	(28	Priority indicator
↑ S	13	Screen display pause)	29	Priority indicator
↑ T	14	Key input function	*	2A	Integration operator
↑ U	15	1 line deletion	+	2B	Addition operator

(to be continued)

Table A-1 Commands Table (cont'd)

Command	ASCII Code	Function	Command	ASCII Code	Function
'	2C	Dummy	B	42	Hexadecimal constant
-	2D	Subtraction operator	C	43	Hexadecimal constant
.	2E	Built-in macro prefix	D	44	Hexadecimal constant
/	2F	Division operator	E	45	Hexadecimal constant
0	30	Numeric constant	F	46	Hexadecimal constant
1	31	Numeric constant	G	47	<Undefined>
2	32	Numeric constant	H	48	<Undefined>
3	33	Numeric constant	I	49	<Undefined>
4	34	Numeric constant	J	4A	<Undefined>
5	35	Numeric constant	K	4B	<Undefined>
6	36	Numeric constant	L	4C	<Undefined>
7	37	Numeric constant	M	4D	Macro definition command
8	38	Numeric constant	N	4E	<Undefined>
9	39	Numeric constant	O	4F	GOTO
:	3A	Argument pop	P	50	<Undefined>
;	3B	Loop abort	Q	51	Q-register prefix
<	3C	Loop start	R	52	<Undefined>
=	3D	Display command	S	53	<Undefined>
>	3E	Loop end	T	54	<Undefined>
?	3F	Error display	U	55	Q-register assignment
@	40	8-bit pointer	V	56	<Undefined>
A	41	Hexadecimal constant	W	57	<Undefined>

(to be continued)

Table A-1 Commands Table (cont'd)

Command	ASCII Code	Function	Command	ASCII Code	Function
X	58	Array assignment		6E	<Undefined>
Y	59	<Undefined>		6F	<Undefined>
Z	5A	Wait		70	<Undefined>
[5B	Q-register pop		71	<Undefined>
¥	5C	Numeric function		72	<Undefined>
]	5D	Q-register push		73	<Undefined>
^	5E	Control character prefix		74	<Undefined>
_	5F	16-bit pointer		75	<Undefined>
	60	<Undefined>		76	<Undefined>
	61	<Undefined>		77	<Undefined>
	62	<Undefined>		78	<Undefined>
	63	<Undefined>		79	<Undefined>
	64	<Undefined>		7A	<Undefined>
	65	<Undefined>	{	7B	Left shift operator
	66	<Undefined>		7C	Exclusive-OR operator
	67	<Undefined>	}	7D	Right shift operator
	68	<Undefined>	~	7E	Negate operator
	69	<Undefined>	DEL	7F	1 character deletion
	6A	<Undefined>			
	6B	<Undefined>			
	6C	<Undefined>			
	6D	<Undefined>			

A.2 ARRAY TABLE

When referencing an array shown in Table A-2, use ↑FVER=H (CLICE version displayed in hexadecimal), etc.

Table A-2 Array Table

Array Name	Function
VER	CLICE version
WRK	CLICE work area top address
PRM	Program memory top address
TRM	Trace memory top address (be careful of the current bank) BANK#0: PC, BANK#1:PORT, BANK#2:WA, DB, JG, BANK#3: PC, BANK#4:TMO, BANK#5:TM1
PCV	PC coverage memory top address
DCV	Data coverage memory top address
CND	Condition memory top address
ERG	Emulator register top address
PSG	PPG memory top address
CRG	Condition register top address
CRO	Condition register unit #0
CR1	Condition register unit #1
CR2	Condition register unit #2
CR3	Condition register unit #3
SRG	System register top address
DTM	Data memory top address
RGF	Register file top address
RN1	RAM NOT INITIALIZE break register
SPE	STACK OVERFLOW/UNDERFLOW break register

(to be continued)

Table A-2 Array Table (cont'd)

Array Name	Function
IRW	ILLEGAL RAM WRITE break register
PC <input type="checkbox"/>	System register PC address
SP <input type="checkbox"/>	System register SP address
ARO	System register ARO address
AR1	System register AR1 address
AR2	System register AR2 address
AR3	System register AR3 address
WR <input type="checkbox"/>	System register WR address
BNK	System register BANK address
IXH	System register IXH address
IXM	System register IXM address
IXL	System register IXL address
RPH	System register RPH address
RPL	System register RPL address
PSW	System register PSW address
JG <input type="checkbox"/>	System register JG address
ICU	INTERRUPT CONTROL (uPD71059)
TC0	TIMER CONTROL#0 (uPD71054)
TC1	TIMER CONTROL#1 (uPD71054)
SC0	SERIAL CONTROL#0 (uPD71051)
SC1	SERIAL CONTROL#1 (uPD71051)
PGC	PPG CONTROL
PGL	PPG BIT SELECT (bit 0 to bit 7)
PGH	PPG BIT SELECT (bit 8 to bit 15)

(to be continued)

Table A-2 Array Table (cont'd)

Array Name	Function
PIO	PARALLEL INTERFACE#0 (uPD71055)
PI1	PARALLEL INTERFACE#1 (uPD71055)
DSW	DIP SWITCH
RES	RESET Mam'Chip
SEN	SE BOARD NUMBER

Remarks: indicates a space.

A.3 CONDITION REGISTER OFFSET ADDRESS

The offset addresses for the array CRG are shown in Table A-3.

When referencing an array, use $\uparrow\text{FCRG} + \uparrow\text{FDPO} = \text{H}$ (address that stores the break condition DEPTH0 in hexadecimal), etc.

Table A-3 Condition Register Offset Address

Array Name	Function
DPO	Break condition DEPTH0
DP1	Break condition DEPTH1
DP2	Break condition DEPTH2
DP3	Break condition DEPTH3
LVC	Break condition DEPTH counter
CNB	Condition break enable flag
RIB	RAM NOT INITIALIZE break enable flag
IWB	ILLEGAL RAM WRITE break enable flag
SOB	SP OVER/UNDERFLOW break enable flag
TTM	Trace timer
TTP	Trace timer prescaler
TAD	Status trace address trace switching flag
XPD	Logic/analyzer probe data
XPE	Logic analyzer probe data edge/level switching flag
XPM	Logic analyzer probe data mask
XPU	Logic analyzer probe break MATCH/UNMATCH switching flag
PDB	PREVIOUS DATA break enable flag
PCA	PC break address
PCB	PC break enable flag

A.4 CONDITION UNIT REGISTER OFFSET ADDRESS

Table A-4 shows the offset addresses for arrays CR0 to CR3.

When referencing an array, use $\uparrow\text{FCR0} + \uparrow\text{FPAU} = \text{H}$ (address of unit 0 which stores the upper limit of the program address break/trace condition range in hexadecimal), etc.

Table A-4 Condition Unit Register Offset Address

Array Name	Function
PAU	Upper limit of program address break/trace condition range
PAL	Lower limit of program address break/trace condition range
PAM	Program address break/trace condition range MATCH/UNMATCH specification
DTA	Data memory address break/trace condition
DMK	Data memory address break/trace condition mask
DTU	Data memory address break/trace condition MATCH/UNMATCH specification
CRD	Data memory write data break/trace condition
CRM	Data memory write data break/trace condition mask
CRU	Data memory write data break/trace condition MATCH/UNMATCH specification
PDT	Break/trace condition of data previously written to data memory
PDM	MATCH/UNMATCH specification of break/trace condition of data previously written to data memory
SPU	Upper limit of stack pointer break/trace condition range
SPL	Lower limit of stack pointer break/trace condition range
SPM	Stack pointer break/trace condition range MATCH/UNMATCH specification

(to be continued)

Table A-4 Condition Unit Register Offset Address (cont'd)

Array Name	Function
ISD	Break/trace condition by execution of specified instruction
ISM	Mask of break/trace condition by execution of specified instruction
ISU	MATCH/UNMATCH specification of break/trace condition by execution of specified instruction
PTD	Logic analyzer probe data break/trace condition
PTM	Logic analyzer probe data break/trace condition mask
PTE	Logic analyzer probe data edge/level specification
PTU	Logic analyzer probe data break/trace condition MATCH/UNMATCH specification
MAR	Break/trace condition of data specified by MAR (Monitor Address Register)
MAM	Mask of break/trace condition of data specified by MAR
MAU	MATCH/UNMATCH specification of break/trace condition of data specified by MAR
INT	Break/trace condition by interrupt
INM	Mask of break/trace condition by interrupt
INU	MATCH/UNMATCH specification of break/trace condition by interrupt
DMA	Break/trace condition by DMA
DMM	Mask of break/trace condition by DMA
DMU	MATCH/UNMATCH specification of break/trace condition by DMA
AND	AND/OR specification of break/trace condition
CNT	Number of times break/trace condition established count condition
CND	Break/trace condition by number of times break/trace condition established
CNM	Mask of number of times break/trace condition established

(to be continued)

Table A-4 Condition Unit Register Offset Address (cont'd)

Array Name	Function
CDR	Current number of times break/trace condition established
CNU	MATCH/UNMATCH specification of number of times break/trace condition established
TRS	Trace condition ON/OFF condition specification

A.5 PRIMITIVE COMMANDS DESCRIPTIONS

This section describes the functions of the primitive commands. Primitive commands offer a more advanced debugging method for those experienced in the development of programs using the IE-17K.

New commands (macro commands) can be added to the IE-17K or a series of debugging procedures can be programmed.

A.5.1 POINTER

Pointer reads the resources indicated by array elements.

(1) Byte pointer

@ ↑ V

(2) Word pointer

_ ↑ V

@ ↑V Byte Pointer

Format : @ α ↑V
α : Expression

[Function] Reads one byte from the array element.

The byte pointer reads the one byte (8 bits) data from the array element indicated by the evaluated value of the expression described between @ and ↑V.

(Example)

@ 100 ↑V ... Refer to contents of array element number 100H.

@ ↑FDTM ↑V ... Refer to contents of data memory start address (address 0).

Here, ↑FDTM is a function which returns the array address that indicates the data memory start address.

`_ ↑V` Word Pointer

Format : <code>_ α ↑V</code>
<code>α</code> : Expression

[Function] Reads one word from the array element.

The word pointer reads one word (16 bits) of data by reading one byte (8 bits) of data from the array element indicated by the evaluated value of the expression described between `_` and `↑V` and making it the high-order byte and reading one byte (8 bits) of data from the array elements specified by evaluated value + 1 and making it the low-order byte.

(Example 1)

`_ ↑FDTM ↑V ...` Refer to 16-bit data by making the contents of the data memory start address (address 0) the high-order byte and the contents of address 1 the low-order byte.

Here, `↑FDTM` is a function which returns the data memory start address.

(Example 2)

_ ↑FPRM+ (10H {1) ↑V=P

... Display the contents of
program memory address
10H.

Here, ↑FPRM is a function
which returns the array
address that indicates the
program memory start
address.

A.5.2 FUNCTION

CLICE has two kinds of functions, "built-in functions" which are built into the system and "user-defined functions" ($\uparrow Q$ commands) which are defined by the user.

Various built-in functions, including array address function ($\uparrow F$ command), key input function ($\uparrow T$ command), and numeric function (\forall command), are available.

(1) User-defined function

$\uparrow Q$

(2) Array address function

$\uparrow F$

(3) Key input function

$\uparrow T$

(4) Numeric function

\forall

↑Q User-Defined Function

Format : ↑Q α
α : Q-register name

[Function] Executes the contents of the Q-register and returns a value.

The ↑Q command executes the command string (character string) stored in the Q-register indicated by α and returns the defined value in it.

For a detailed description of this command, see Appendix A.5.6 "Macro".

↑F Array Address Function

Format : ↑F α

α : Array element name

[Function] Returns the array address of an array element name.

The ↑F command returns the array address indicated by the array element name α specified by the three characters following it.

The array element names table is shown in Appendix A.2.

(Example) ↑FPRM

This command returns the array address corresponding to program memory address 0.

Therefore, when the program memory address uses array address.

↑FPRM+1 ... Low-order 8 bits of
address 0

↑ FPRM+n ... When n an even number:
High-order 8 bits of
address $n/2$

When n an odd number:
Low-order 8 bits of
address $(n-1)/2$

↑T Key Input Function

Format : ↑T

[Function] Returns the code of the keyed in character.

Returns the ASCII code of the keyed in character. When there is no key input, this command waits until a key is input.

When the ↑C key is input two consecutive times, the ↑T command is terminated even if it is being executed.

(Example) ↑T1↑T2↑T3
① ② ③

When the command string above is executed, first the program enters the input wait state at ↑T ①. When ↑C is keyed in here, ↑T ① returns 03H and assigns 03H to Q1. (See "U command (Assignment to Q-register)" in Appendix A.5.3 "Assignment")

Then, command execution shifts to ↑T ②.

When ↑C is keyed in here,
execution of the command string is
terminated by ↑T ② .

In short, two consecutive ↑C
cannot be input at ↑T.

The input characters are displayed
automatically.

¥ Numeric Function

Format : ¥ α
α : Q-register name

[Function] Converts a character string to a numeric.

Returns the numeric which represents the character string stored in the Q-register specified by α .

At this time, the number of digits of the numeric is the number of digits from the specified Q-register start address up to the appearance of the first character other than a numeric or from the start of the Q-register to the last character of the Q-register.

(Example) When the contents of Q-register 1 are as shown below, for ¥1,

① ↑B1234 ... Returns 1.

(2 is not a binary number)

② BACK ... Returns 0BACH.

(K is not a hexadecimal number.)

③ ↑D16*2 ... Returns 10H.

(* is not a decimal number.)

When 1 character cannot be converted as a numeric and when the range which can be represented as a constant is exceeded as shown below, an error is recognized.

④ ↑DECIMAL ... Error

⑤ 12345678901234567890
... Error

A.5.3 ASSIGNMENT

A U command and XB, XC and XW commands as available with CLICE to assign a numeric or character string to a variable.

- (1) Assignment to Q-register

U

- (2) Byte data assignment to array

XB

- (3) Word data assignment to array

XW

- (4) Character string assignment to array

XC

U Q-register Assignment

Format :	① $\beta U \alpha$
	② $U \alpha \gamma \$$ (macro definition)
	③ $\delta, \epsilon U \alpha$

α :	Q-register name
β :	Expression
γ :	Character string
δ :	Array start address
ϵ :	Array end address

[Function] Assigns a numeric or character string to the Q-register.

Format ① assigns the evaluated value of expression β to the Q-register specified by α .

(Example) 3UA ... Assign 3 to Q-register A.

@ ↑FDTM+1 ↑VU3

... Assign the contents of data memory address 1 to Q-register 3.

Format ② assigns the character string γ up to $\$$ to the Q-register specified by α .

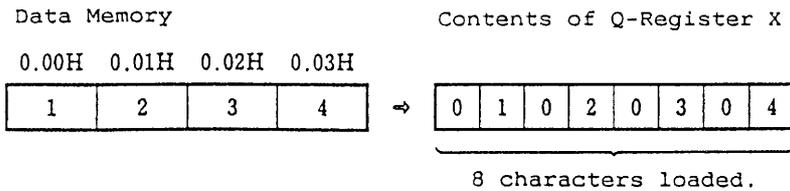
When a command string is assigned as a character string γ , it can be used as a macro.

(Example) U1ABC\$... Assign ABC to Q-register
1.

Format ③ assigns the contents from the array address specified by the evaluated value of expression δ to the array address specified by the evaluated value of expression ϵ to the Q-register specified by α as a character string.

(Example) \uparrow FDTM, \uparrow FDTM+3UX

... Assign the contents of data memory addresses 0 to 3 to Q-register X as character string.



 XB Byte Data Assignment to Array

Format :	α , β XB
α :	Expression (only low-order 8 bits valid)
β :	Array address

[Function] Assigns 8-bit data α to array β .

Assigns the low-order 8 bits of the evaluated value of expression α to the elements of the array address indicated by the evaluated value of expression β .

(Example 1) Q1, \uparrow FDTMXB

... Assign low-order 8 bits of Q-register 1 to data memory address 0.

(Example 2) A sample program which stores the keyed in characters to the array work area is shown below.

$$\begin{array}{l} \uparrow\text{FWRKUP} < \uparrow\text{TU2} \quad \text{D-Q2} ; \quad \text{Q2, QPXB} \quad \%P> \\ \text{QP} = \text{FWRK} \quad \text{Q2} = \text{key} \quad \text{if CR ?} \quad \text{ARRAY (Q1)} = \text{Q2} \quad \text{QP} = \text{QP} + 1 \end{array}$$

This program exits from the loop when the keyed characters are ASCII code 0DH or less. (Also see "<> command (loop)" in Appendix A.5.7 "Control".)

XW Word Data Assignment to Array

Format : α , β XW

α : Expression (only low-order 16 bits valid)

β : Array address

[Function] Assigns 16-bits data α to array β .

Assigns the low-order 16 bits of the evaluated value of expression α to the elements of the array address indicated by the evaluated value of expression β .

(Example) Q1, \uparrow FPRMXW

... Assign low-order 16-bits of Q-register 1 to data memory addresses 0 and 1.

This example can also be described as follows by using the XB command.

Q1}8, \uparrow FDTMXBQ1, \uparrow FDTM+1XB

XC Character String Assignment to Array

Format : β XC α
α : Q-register name
β : Array address

[Function] Assigns the character string stored in Q-register α to array β .

The character string stored in the Q-register specified by α is assigned to the array range of that number of characters sequentially from the array address element showing the evaluated value of expression β .

(Example) U1ABC\$ ↑FWRKXC1
 ① ②

This example assigns the character string ABC to Q-register 1 at ① and assigns its value to A, B and C sequentially from the start of the array work area at ②.

That is, A is assigned to array address \uparrow FWRK, B is assigned to \uparrow FWRK+1, and C is assigned to \uparrow FWRK+2.

This example can also be described as follows
by using the XB command.

41, ↑FWRKXB42, ↑FWRK+1XB43, ↑FWRK+2XB

However, 41H, 42H, 43H shows A, B, C by ASCII
code.

A.5.4 ARGUMENT STACK

CLICE has an argument stack as a temporary memory for passing command numeric arguments.

A numeric argument consists of a constant, function value, or other expression evaluated value. It is stored in an argument stack and used by reading it by command.

How an argument stack is used is shown below.

```
3, 4*2, 5+1$
```

The numeric argument 3 is pushed to the argument stack and the numeric arguments 8 and 6 separated by a comma (,) are then pushed to the argument stack in order.

Then, the command is used by popping the numeric arguments from the argument stack in 6,8,3 order.

The argument stack is also initialized at the end of execution of the command string.

- (1) Argument push
 Numeric
- (2) Argument pop
 :

Argument Push

Format : α [, α]

α : Expression

[Function] Pushes a numeric to the argument stack.

The value of the expression is pushed to the argument stack.

The level of the argument stack is incremented (+1).

(Example) 12, 34, 56\$

The values 12H and 34H and 56H are pushed to the argument stack in order.

: Argument Pop

Format : : α
α : Q-register name

[Function] Pops the numeric from the argument stack and stores it to the Q-register.

Pops the numeric from the argument stack and stores it to the Q-register specified by α .

The level of the argument stack is decremented (-1).

(Example) UM:A:BQA*QB=D\$\$
↑D12, 3MM\$\$36

Stores a command string to Q-register M and then executes macro M, which makes 12 and 3 arguments.

The result 36 is displayed.

(For a description of the commands used, see Appendix A.5.6 "Macro" and Appendix A.5.8 "Display".)

A.5.5 Q-STACK

The Q-stack is a stack for saving the contents stored in the Q-registers.

The Q-stack is initialized at the end of execution of a command string.

(1) Q-register push

[

(2) Q-register pop

]

[Q-register Push

Format : [α
α : Q-register name

[Function] Pushes the contents of Q-register α to the Q-stack.

Pushes the contents (numeric or character string) of the Q-register specified by α to the Q-stack.

The level of Q-stack is incremented (+1).

(Example) 34UN [N

Assigns 34H to Q-register N and pushes 34H to the Q-stack.

] Q-register Pop

Format :] α
α : Q-register name

[Function] Stores the contents popped from the Q-stack to Q-register α .

Stores the contents (numeric or character string) popped from the Q-stack to the Q-register specified by α .

The level of the Q-stack is decremented (-1).

(Example)] NQN=H

The contents popped from Q-register N are displayed as a hexadecimal value.

A.5.6 MACRO

A macro is a function which executes a character string stored in a Q-register as a command.

There are two kinds of macro execution, macro command (M command) and user function (\uparrow Q command).

(1) Macro command execution

M

(2) User function execution

\uparrow Q

M Macro Command Execution

Format : M α
α : Q-register name

[Function] Executes the contents of Q-register α as a command string.

Executes the contents stored in the Q-register specified by α as a command.

The parameters (arguments) are passed to the character string in the Q-register through variable, Q-stack, and argument stack.

Macro commands can also be nested.

A macro which displays the sum of two arguments in hexadecimal is shown below.

(Example 1) Pass parameters through a variable.

```
UMQ1+Q2=H$$  
100U1200U2MM$$300
```

Stores a command which adds and hexadecimal displays the contents of Q-register 1 and Q-register 2 and stores the result to Q-register M.

Next, it assigns 100H and 200H to Q-register 1 and Q-register 2, respectively, and passes the parameters by executing macro M.

The result 300 is displayed.

When the same operation as the example above is performed through an array.

```
UM _↑FWRK ↑V+ _↑FWRK+2 ↑V=H$$
100, ↑FWRKXW200, ↑FWRK+2XWMM$$300
```

(Example 2) Pass parameters through the Q-stack

When macros are nested, parameters may be passed through variables by stacking the variables (Q-register, etc.) used.

One method of solving this problem is to use the Q-stack.

```
UM] 1Q1+] 1Q1=H$$
100U1 [1 200U1 [1 MM$$300
```

Stores a command string that adds and hexadecimal displays the numerics popped to Q-register 1 to Q-register M.

Next, the numeric assigned to Q-register 1 is pushed to the Q-stack and parameters can be passed by executing macro M.

(Example 3) Pass parameters through argument stack

This is the most simple method of passing parameters.

```
UM:1Q1+ :1Q1=H$$  
100, 200MM$$300
```

Stores a command string which adds and hexadecimal displays the numerical values popped to Q-register 1 to Q-register M.

Next, the numeric is pushed to the argument stack and the parameters can be passed by executing macro M.

↑Q User Function Execution

Format : ↑Q α
α : Q-register name

[Function] Executes the contents of Q-register α as a command string (function) and returns a value.

Executes the contents stored in the Q-register specified by α as a command.

The value which can be returned as a function in this command is pushed to the Q-stack.

A value is fetched automatically from the Q-stack when execution of this function is completed by means of this.

Parameters (arguments) are passed to the command string in the Q-register through variable, Q-stack, or argument stack, the same as a macro command.

User-defined functions can also be nested.

(Example) An example of definition of a function that returns a program memory address is shown below.

This function returns the logical address (1 address/8 bits) on an array, with a program memory address (1 address/16 bits) as the input.

UM:1 ↑ FPRM+ (Q1*2) ↑ VU1 [1\$

This macro M is used as follows:

_100 ↑ QM ↑ V=H\$\$

This hexadecimal displays the contents of program memory address 100H.

A.5.7 CONTROL

CLICE can control the execution order of each command.

- (1) Loop
 <>
- (2) Loop abort
 ;
- (3) Tag
 !!
- (4) Unconditional branch
 0\$
- (5) Conditional branch
 " ,

<> Loop

Format : [α] < β >
α : Number of repetitions
β : Command string

[Function] Executes command string β α times.

When α is $2^{32}-1$, it can be omitted.

Command string β enclosed in < > is executed the number of times indicated by the value of the numeric argument specified by α .

Loops can be nested.

When the number of repetitions α is omitted, $2^{32}-1$ times is assumed.

(Example) 10<command string A 5 <command string B> command string C>

In this case, command string A is executed once, command string B is executed 5 times, and command string C is executed once so that processing is repeated 16 times.

; Loop Abort

Format : α ;

α : End conditional expression

[Function] Aborts a repetitive loop.

When the value of the numeric argument specified by α becomes 0 or more before the end of the specified number of repetitions of a repetitive loop, repetition of the command string ends and the program exits from the loop.

In other words, control shifts to the command described at the immediate right of the > that indicates the end of the loop containing;.

When the numeric argument specified by α is smaller than 0, this command is ignored.

A ; command outside a loop generates an error.

(Example) 100<1F- ↑T;command string>

Executes the command string repeatedly until the loop reaches 256 repetitions or a control key (ASCII code 00H to 1FH) is input.

!! Tag

Format : ! α !
α : Tag

[Function] Shows the tag in a command string.

A comment is described to explain processing contents in the command string or specify the jump destination of an O command.

The character string enclosed in ! has no affect on execution of the command.

(Example) \uparrow FWRKU1!Q1=WORK POINTER!

Assigns the array WRK address to Q-register 1.

The character string enclosed in ! is treated as a comment and is not executed as a command.

O\$ Unconditional Branch

Format : O α \$

α : Tag

[Function] Unconditionally shifts control to the specified tag.

The O command changes the command flow at the tag described by the same character string as the character string (tag) described at the right side of the command.

When the same tag is not described, the tag that appears first is valid.

(Example) !LOOP! command string OLOOP\$

Executes the command string infinitely.

" ' Conditional Branch

Format : α " β γ ' δ

α : Conditional expression

β : Condition

γ : Command string 1

δ : Command string 2

[Function] Changes the command to be executed according to the condition.

Judges the value of conditional expression α and controls the command execution flow.

When the value of the numeric argument specified by α satisfies condition β described at the right side of " β ", command string γ following the condition is executed, then command string δ is executed.

If the condition is not satisfied, only command string δ following ' is executed.

That is, command string γ is skipped.

This conditional branch command can be nested.

The following four conditions can be described:

- E: Equal zero (n=0)
- N: Not equal zero (n≠0)
- L: Less than zero (n<0)
- G: Greater than zero (n>0)

(Example) Q1-3" EOU2'Q1+Q2U1

In this example, if Q1-3 is 0 (that is, Q1=3), after 0 is assigned to Q-register 2, the next command Q1+Q2U1 is executed.

If Q1≠3, OU2 is not skipped and Q1+Q2U1 following ' is executed.

A.5.8 DISPLAY

CLICE has commands which display characters and numerics.

- (1) Numeric binary display
=B
- (2) Numeric decimal display
=D
- (3) Numeric hexadecimal display
=H
- (4) Q-register contents character display
=C
- (5) Character string display
↑A
- (6) 1-4-3-4-4 bit format display
=P

=B Numeric Binary Display

Format : [β ,] α =B
α : Numeric data
β : Number of display digits

[Function] Binary displays the value of the numeric argument specified by numeric data α .

The value of the numeric argument specified by β indicates the number of display digits. When it is 0 or is omitted, the data is displayed by suppressing the leading zeros.

(Example 1) A=B\$\$1010

Binary displays 0AH. At this time, the command is executed by \$\$, but line feed is not performed and the value is displayed unchanged.

(Example 2) 8, A=B\$\$00001010

Binary displays 0AH in 8 digits.

=D Numeric Decimal Display

Format : [β ,] α =D
α : Numeric data
β : Number of display digits

[Function] Decimal displays the value of the numeric argument specified by numeric data α .

The value of the numeric argument specified by β indicates the number of display digits. When it is 0 or is omitted, the value is displayed by suppressing the leading zeros.

(Example 1) A=D\$\$10

Decimal displays 0AH. At this time, the command is executed by \$\$, but line feed is not performed and the value is displayed as is.

(Example 2) 4, A=D\$\$0010

Decimal display 0AH in 4 digits.

=H Numeric Hexadecimal Display

Format : [β ,] α =H
α : Numeric data
β : Number of display digits

[Function] Hexadecimal displays the value of the numeric argument specified by numeric data α .

The value of the numeric argument specified by β indicates the number of display digits. When it is 0 or is omitted, the value is displayed by suppressing the leading zeros.

(Example 1) \uparrow D10=H\$\$A

Hexadecimal displays the decimal number 10. At this time, the command is executed by \$\$, but line feed is not performed and the value is displayed unchanged.

(Example 2) 4, A=H\$\$000A

Hexadecimal displays 0AH in 4 digits.

=C Q-register Contents Character Display

Format : =C α
α : Q-register name

[Function] Displays the contents stored in Q-register α as a character string.

(Example) U1ABC\$=C1\$\$ABC

↑A Character String Display

Format : = ↑A α ↑A

α : Character string

[Function] Displays character string α enclosed in ↑A.

(Example) ↑ ABELL ↑ R ↑ GBELL TWICE ↑ R ↑ G ↑ R
↑ G ↑ A\$\$

Display BELL and sound one beep and displays
BELL TWICE and sounds two beeps.

=P 1-4-3-4-4 Bit Format Display

Format : α =P
α : Numeric data

[Function] Displays the value of the numeric argument specified by α in 1-4-3-4-4 bit format.

(Example) \uparrow B0011110011110000=P\$\$078F0

Display 0011110011110000B(NOP) in 1-4-3-4-4 format. At this time, the command is executed by \$\$, but line feed is not performed and the value is displayed unchanged.

A.5.9 OTHERS

CLICE has the following commands, besides those described above.

- (1) Q-register increment
%
- (2) Wait
Z
- (3) Assignment to Q-register
*
- (4) Error display
?
- (5) Data memory read from device
Y
- (6) Data memory write to device
W

% Q-register Increment

Format : % α
α : Q-register name

[Function] Increment (+1) the contents of the Q-register.

Increments (+1) the contents stored in Q-register α as a numeric.

(Example) 100U1%1

After 100H is assigned to Q-register 1, the contents of Q-register 1 are incremented (+1) by % command.

As a result, the contents of Q-register 1 become 101H.

Z Wait

Format : α Z

α : Wait time

[Function] Halts execution of the next command for the time indicated by the value of the numeric argument specified by α . The time units are 10 msec.

(Example) \uparrow D1000Z

Halts execution for 10 seconds.

* Assignment to Q-register

Format : * α
α : Q-register name

[Function] Assigns the contents of the command buffer to Q-register α .

When command string input is interrupted by $\uparrow C$ and when command execution is interrupted by $\uparrow C\uparrow C$ or when a prompt is displayed by interrupt by an error during command execution, the contents of the command buffer can be stored to the Q-register by entering * at the 1st character and specifying the Q-register at the 1st character following the prompt.

(Example)

```
xxx>:1 _  $\uparrow$ FPRM+ (Q1 {1)  $\uparrow$ V=H$  $\uparrow$ C ... ①  
xxx> *2 ... ②  
xxx> M2$$ ... ③
```

At ①, a command string is input and the command is terminated by \$ and input is interrupted by $\uparrow C$.

At ②, the command string of ① is assigned to Q-register 2 by *2.

At this time, if assignment is performed normally, a prompt is displayed as shown at ③ .

The command string stored to Q-register 2 can be executed by M command.

Macros can be easily created by using the * like this.

When command execution is interrupted by an error, the error can be corrected and the command can be re-executed by .ED command by assigning the contents of the command buffer to a Q-register by * command.

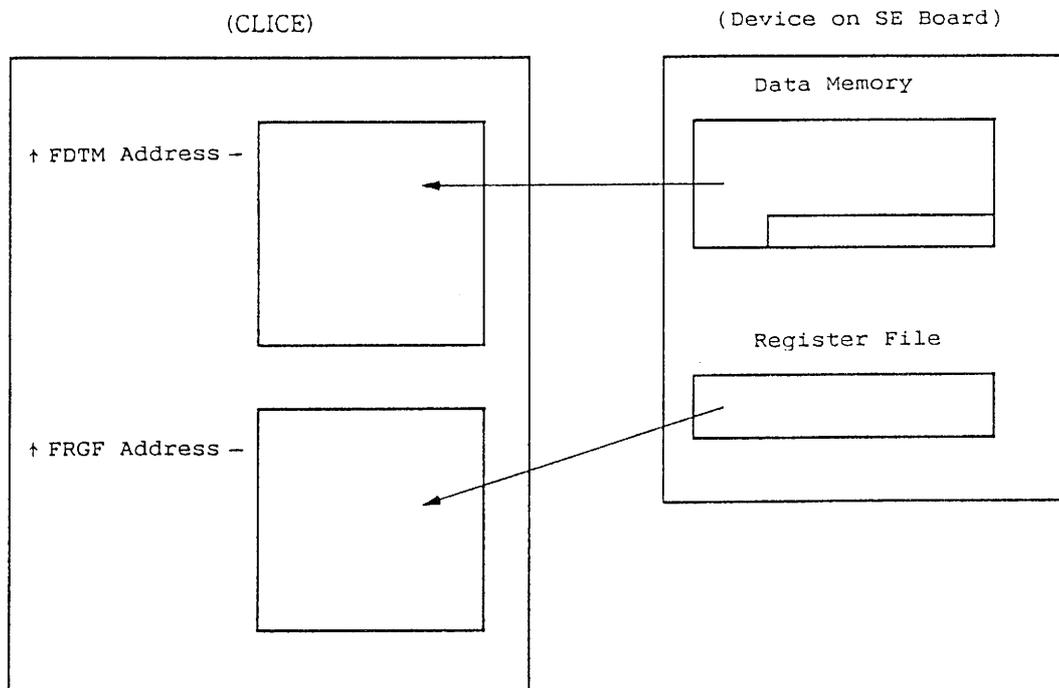
This function allows efficient command input without the need to re-input the command from the beginning even when a long command string is input and an error is generated.

Y Data Memory Read Command

Format : Y

[Function] Reads the contents of the data memory and register file of a device on the SE board to CLICE.

Reads the contents of the data memory of the device on the SE board to the address indicated by array DTM and succeeding addresses. The contents of a register file are read to the address indicated by array RGF and succeeding addresses.

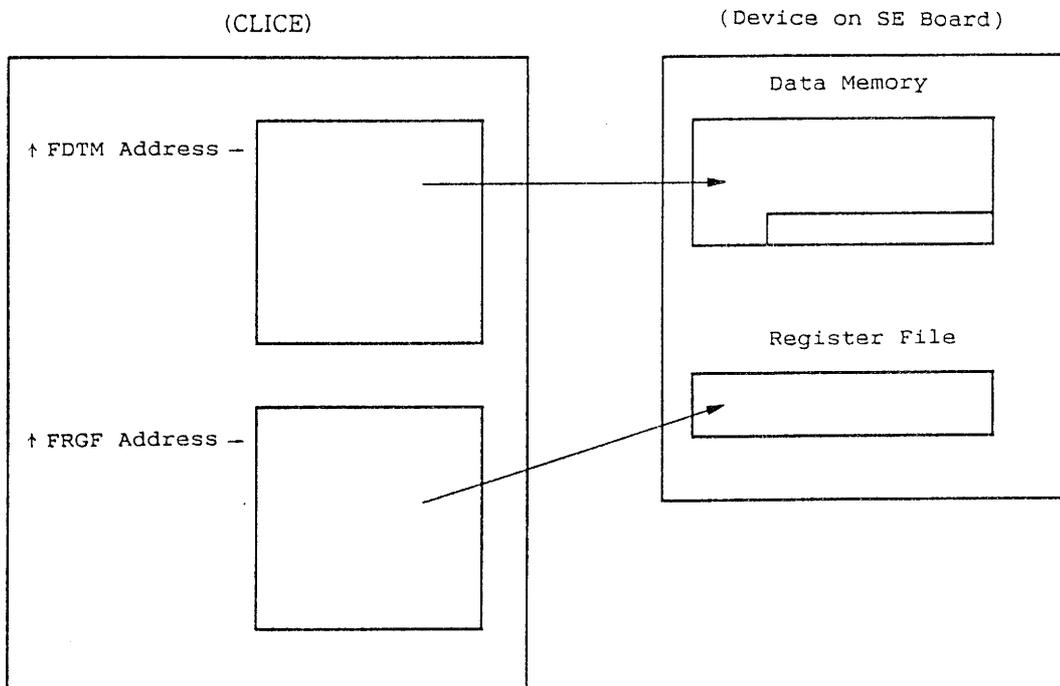


W Data Memory Write Command

Format : W

[Function] Writes the data on CLICE to the data memory and register file of the device on the SE board.

Writes the data beginning from the address indicated by array DTM to the data memory of the device on the SE board. Writes the data beginning from the address indicated by array RGF to the register file of the device on the SE board.



A.6 EDITOR

CLICE has editor functions for editing command strings.

The editor can add, change, and delete all, or part, of a command string (character string).

A.6.1 COMMAND BUFFER EDITING

[Function] Edits the command buffer.

[Format] ↑E

When ↑E is input during command string input, CLICE enters the editor mode which edits the input command string.

The editor command to be described later can be used in editor mode.

(Example) xxx>U1 ↑ASTRINGS ↑A\$ ↑E
> ... Enters editor mode

A.6.2 Q-REGISTER EDITING

[Function] Edits the contents of the Q-register.

[Format] .ED α

CLICE enters the editor mode which edits the contents stored in the Q-register specified by α as a character string.

The editor commands to be described later are used in the editor mode.

(Example) `xxx>U1 ↑ ASTRINGS ↑ A$$`
`xxx>.ED1$$`
`>`

... Enters the editor mode.

[Note] When Q-register α is not defined, the error message

`?NVQ NO VALUE IN Q-REGISTER`

is displayed.

A.6.3 EDITOR COMMANDS

The editor commands include move, insert, delete, search, replace, display and end.

In some commands, the number of times the command is to be executed can be specified in front of the command.

The number of times is specified by a 2-digit decimal.

When 0 is specified or specification is omitted, 1 is assumed.

Each editor command consists of one uppercase alphabetic character. Note that an editor command is not displayed even if input.

When an editor command generates an error, a beep is sounded and the input command is ignored.

(1) Cursor control

The cursor is moved to the right and the character to be edited is displayed with the space key.

The cursor is moved to the left and the displayed character is detected with the BS or DEL key.

The cursor can be moved to the position to be edited by these operations.

Editing is performed at the position indicated by the cursor.

(Example) >■ The first character is edited.

>S■ When the cursor is moved to the right with the space key, a character string is displayed.

>ST■

>STRING■

>STRIN■

When the cursor is moved to the left with the BS or DEL key, the displayed character is deleted.

■ indicates the cursor.

(2) Insertion

[Command] I

Inserts the keyed in character string at the current cursor position.

Insertion is ended by pressing the ESC key.

(Example) >ABCGHI ■ Original character string.

>ABC ■ Cursor is moved to G.

>ABCDEF ■ IDEF(ESC) is input.

>ABCDEFGHI ■

When the cursor is moved, it can be seen that a character string have been inserted.

[Command] X

After moving the cursor to the end of the line, this command operates the same as the I command.

Insertion is ended by pressing the ESC key.

(Example) > ■ Original character string.

>ABCDEF ■

When X is input, the entire character string is displayed.

>ABCDEFGHI ■

GHI(ESC) is inserted and insertion ends.

[Command] P

Inserts the keyed in character at the current cursor position.

(Example) >ABCEF■ Original character string.

>ABC■ Cursor is moved to E.

>ABCD■ PD is input

>ABCDEF■

When the cursor is moved, it can be seen that characters have been inserted.

(3) Deletion

[Command] D

Deletes one character from the current cursor position.

When a count is specified, that number of characters are deleted.

(Example 1)

>ABCDEF■ Original character string.

>ABCD■ The cursor is moved to E.

>ABCD~~Y~~E~~Y~~■

When D is input, the deleted character is displayed.

(Example 2)

>ABCDEF■ Original character string.

>ABC■ The cursor is moved to D.

>ABC¥DEF¥■

When 3D is input, the three
deleted characters are
displayed.

[Command] H

Deletes the characters from the current cursor
position to the end of the line and operates the same
as the I command.

Insertion is ended by pressing the ESC key.

(Example) >ABC■ Original character string.

>¥ABC¥■ The characters deleted by
the H command are
displayed.

>¥ABC¥DEF■ DEF (ESC) are input.

(4) Search

[Command] S

Searches for the keyed in character from the current cursor position and moves the cursor to the found position.

If the character is not found, moves the cursor to the end and sounds a beep.

When a count is specified, searches for that number of characters.

(Example 1) >ABCDEFGHI■

Original character string.

>■

The cursor is moved to the top.

>ABCDEF■

When SG is input, the cursor moves to G.

(Example 2) >ABCABCABC■

Original character string.

>■

The cursor is moved to the top.

>ABCABCA■

When 3SB is input, the cursor moves to the 3rd B.

(5) Replace

[Command] C

Replaces the character at the current cursor position with the keyed in character.

When a count is specified, characters are replaced by the number of key inputs.

(Example 1)

>ABC*EF■ Original character string.

>ABC■ Cursor is moved to the *.

>ABCD■

When CD is input, the * is replaced by D.

(Example 2)

>ABC1234HI■

Original character string.

>ABC■ Cursor is moved to 1.

>ABCDEFGH■

When 4CDEFG is input, 1234 are replaced by DEFG.

[Command] R

Replaces the character string from the current cursor position with the keyed in character string.

Replacement is ended with the ESC key.

(Example) >ABC123GHI■

Original character string.

>ABC■ Cursor is moved to 1.

>ABCDEF■ RDEF (ESC) are input.

>ABCDEFGHI■

When the cursor is moved, the replaced character string can be seen.

(6) Display

[Command] L

Displays the character string from the current cursor position to the end of the line and moves the cursor to the top of the line.

(Example) >ABCDEFGH■ Original character string.

>ABC■ Cursor is moved.

>ABCDEFGH The character string is displayed by L command.

>■ Automatic line fed cursor moves to the top of the line.

[Command] T

Displays the character string from the current cursor position to the last character string being edited and moves the cursor to the top of the character string being edited.

(Example) >ABCDEFG

HIJKL ■ Original character string.

>ABC ■ Cursor is moved.

>ABCDEFG

HIJKL

Character string displayed by
T command.

> ■ Automatically line fed cursor
moves to top.

(7) END

The editor mode is ended with the ESC key.

APPENDIX B. BUILT-IN MACRO COMMANDS

Built-in macro commands are listed in the following tables. These can be used for program development. These tables can also be used as an index.

B.1 PROGRAM MEMORY CONTROL COMMANDS

Command Name	Function	Outline	Page
.LPO	Program memory load (channel 0)	When IE-17K starts up, downloads xxx.ICE file using RS-232-C channel 0.	5-21
.LP1	Program memory load (channel 1)	When IE-17K starts up, downloads xxx.ICE file using RS-232-C channel 1.	5-21
.VPO	Program memory verify (channel 0)	Using RS-232-C channel 0, verify program memory and ICE file downloaded in IE-17K.	5-22
.VP1	Program memory verify (channel 1)	Using RS-232-C channel 1, verify program memory and ICE file downloaded in IE-17K.	5-22
.IP	Program memory initialization	Rewrites specified data at a time in arbitrary range of IE-17K program area (1-4-3-4-4 format).	5-24
.CP	Program memory change	Rewrites an instruction in arbitrary addresses of IE-17K program area (1-4-3-4-4 format).	5-25
.AP	Assemble command (change by mnemonic)	Rewrites an instruction from mnemonic in arbitrary addresses of IE-17K program area. It is convenient if used together with UP command.	5-27
.DP	Program memory dump	Displays maximum 3FH-byte data from arbitrary address in IE-17K program area (1-4-3-4-4 format).	5-30
.UP	Disassemble command (dump by mnemonic)	Displays maximum 10 steps (in mnemonic format) from arbitrary address of IE-17K program area.	5-32
.FP	Program memory search	Searches for contents of program memory in 1-4-3-4-4 format.	5-35

(to be continued)

(cont'd)

Command Name	Function	Outline	Page
.SPO	Program memory save (channel 0)	Using RS-232-C channel 0, saves an IE-17K program. Convenient to save a patched program.	5-36
.SP1	Program memory save (channel 1)	Using RS-232-C channel 1, saves an IE-17K program. it is convenient to save a patched program.	5-36
.XS0	PROM data output (channel 0)	Using RS-232-C channel 0, converts and outputs IE-17K program to PROM file.	5-37
.XS1	PROM data output (channel 1)	Using RS-232-C channel 1, converts and outputs IE-17K program to PROM file.	5-37
.Q	Restart	If prompt is ">@@@" and this command is executed, a program downloaded once is restarted.	5-38

B.2 DATA MEMORY CONTROL COMMANDS

Command Name	Function	Outline	Page
.ID	Data memory initialization	Initializes specified data in arbitrary range of data memory at a time.	5-41
.CD	Data memory change	Rewrites data in arbitrary address of data memory in an address unit.	5-42
.DD	Data memory dump	Dumps data memory in arbitrary range.	5-44
.D	All data memory dump	Dumps all data memory.	5-46

B.3 PERIPHERAL CIRCUIT CONTROL COMMANDS

Command Name	Function	Outline	Page
.GD	Peripheral register contents display	Displays peripheral register contents.	5-48
.GE	Peripheral register contents read	Assigns peripheral register contents to Q-register. Using a primitive command, this command is used for editing a user macro.	5-49
.PD	Peripheral register write	Assigns a value to the peripheral register.	5-51
.PU	Indirect peripheral register write	Assigns Q-register contents to the peripheral register. Using a primitive command, this command is used for editing a user macro.	5-52

B.4 EMULATION COMMANDS

Command Name	Function	Outline	Page
.R	Reset	Resets an SE board. Program execution address becomes OH, and data memory and register file, etc. become the status equal to the reset status of target products.	5-54, 7-1
.RN	Program execution	Executes a program from specified program execution start address. The break/trace conditions are not changed.	5-55, 7-1
.BG	Program execution (partial reset of break/trace reset)	Executes a program from specified program execution start address. The break/trace conditions are partially reset.	5-56
.BK	Break	Stops program execution.	5-57, 7-1
.CA	Program start address change	Changes program execution start address.	5-58
.S	Step operation	Using space key, executes program according to the number of times specified by keypad input in an instruction unit.	5-59, 7-14
.DS	Display	This is a command associated with certain products with LCD controller.	5-61

B.5 BREAK/TRACE CONDITION CONTROL COMMANDS

Command Name	Function	Outline	Page
.CC	Break/trace condition change	Sets/changes break/trace condition.	5-63, 7-4
.CT	Trace on/off condition change	Sets trace start/end and trace one shot.	5-84
.DC	Break/trace condition dump	Dumps on display the break/trace condition.	5-89
.DT	Trace table dump	Dumps trace result.	5-91
.SCO	Break/trace condition save (channel 0)	Output to RS-232-C channel 0 the break/trace condition set by .CC level 1.	5-95
.SC1	Break/trace condition save (channel 1)	Output to RS-232-C channel 1 the break/trace condition set by .CC level 1.	5-95
.LCO	Break/trace condition load (channel 0)	Downloads the break/trace condition from RS-232-C channel 0.	5-96
.LC1	Break/trace condition load (channel 1)	Downloads the break/trace condition from RS-232-C channel 1.	5-96
.VCO	Break/trace condition verify (channel 0)	Verifies data from RS-232-C channel 0 using break/trace condition.	5-97
.VC1	Break/trace condition verify (channel 1)	Verifies data from RS-232-C channel 1 using break/trace condition.	5-97

B.6 COVERAGE DISPLAY COMMAND

Command Name	Function	Outline	Page
.DM	Coverage memory dump	Displays program memory change history and data memory write history.	5-99

B.7 PROGRAM PATTERN GENERATOR (PPG) CONTROL COMMANDS

Command Name	Function	Outline	Page
.IG	PPG data initialization	Initializes PPG data.	5-102
.CG	PPG data setting /change	Sets/changes PPG data.	5-103, 6-1
.DG	PPG data dump	Dumps PPG data.	5-105, 6-1
.EG	PPG execution/ stop & Operating mode setting	Executes PPG, issues stop directive, and sets output pin and execution speed.	5-106, 6-1
.SG0	PPG data save (channel 0)	Outputs PPG data to RS-232-C channel 0.	5-108
.SG1	PPG data save (channel 1)	Outputs PPG data to RS-232-C channel 1.	5-108
.LG0	PPG data load (channel 0)	Downloads data from RS-232-C channel 0 using PPG data.	5-109
.LG1	PPG data load (channel 1)	Downloads data from RS-232-C channel 1 using PPG data.	5-109
.VG0	PPG data verify (channel 0)	Verify data from RS-232-C channel 0 using PPG data.	5-110
.VG1	PPG data verify (channel 1)	Verify data from RS-232-C channel 1 using PPG data.	5-110

B.8 HELP COMMAND

Command Name	Function	Outline	Page
.H	Support command display	Displays command list.	5-112

Phase-out/Discontinued